PA04 - InterfaceClass

Generated by Doxygen 1.8.6

Wed Feb 17 2016 11:24:01

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=NULL, DataNode< DataType > *inNextPtr=NULL)

  Implementation of parameterized constructor for DataNode class.

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)

Implementation of parameterized constructor for DataNode class.

Initializers used to set data members

Precondition

Assumes an uninitialized DataNode object

Postcondition

Initialized DataNode object

Algorithm

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

Exceptions

None	

Parameters

in	*dataItem*	Reference parameter of type DataType which will be the item the node holds (DataType)
in	*previous*	DataNode pointer that points to the previous node in the list (DataNode<DataType>*)
in	*next*	DataNode pointer that points to the next node in the list (DataNode<DataType>*)

Returns

None

Note

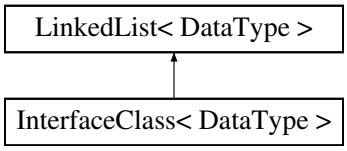
None

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

- LinkedList.h
- LinkedList.cpp

4.2 InterfaceClass< DataType > Class Template Reference

Inheritance diagram for InterfaceClass< DataType >:



Public Member Functions

- InterfaceClass ()
  Implementation of InterfaceClass default constructor.
- InterfaceClass (const InterfaceClass< DataType > &copiedList)
  Implementation of InterfaceClass copy constructor.
- virtual ~InterfaceClass ()
  Implementation of InterfaceClass destructor.
- virtual bool operator++ ()
  Implementation of InterfaceClass overloaded increment operator.
- virtual bool operator-- ()
  Implementation of InterfaceClass overloaded decrement operator.
- virtual const DataType & operator* ()
  Implementation of InterfaceClass overloaded dereference operator.

Additional Inherited Members

4.2.1 Constructor & Destructor Documentation

4.2.1.1 template<class DataType > InterfaceClass< DataType >::InterfaceClass ( )

Implementation of InterfaceClass default constructor.

The base class default sets data members to default values

Precondition

Assumes an uninitialized InterfaceClass object

Postcondition

The InterfaceClass object's data members are set to default values

Algorithm

The base class default constructor is called implicitly

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.2.1.2 template<class DataType > InterfaceClass< DataType >::InterfaceClass (const InterfaceClass< DataType > & *copiedList*)

Implementation of InterfaceClass copy constructor.

The nodes of the InterfaceClass object passed in as a parameter are copied into the calling object

Precondition

Assumes an uninitialized InterfaceClass object

Postcondition

The calling object has the same nodes as the object passed in as a parameter

Algorithm

The overloaded assignment operator is called on the local object and the InterfaceClass object passed in as a parameter

Exceptions

None	

Parameters

copiedList	A const reference InterfaceClass object that is to be copied into the calling object (InterfaceClass<DataType>)

Returns

None

Note

None

4.2.1.3 template<class DataType > InterfaceClass< DataType >::~InterfaceClass ( ) [virtual]

Implementation of InterfaceClass destructor.

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

Precondition

Assumes an initialized InterfaceClass object

Postcondition

All memory allocated to the list freed and data members set to default values

Algorithm

The method clearList is called to delete all nodes in the list and set data members to default values

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.2.2 Member Function Documentation

4.2.2.1 template<class DataType > const DataType & InterfaceClass< DataType >::operator*( ) [virtual]

Implementation of InterfaceClass overloaded dereference operator.

Returns the item of type DataType at a particular place in the list

Precondition

Assumes an initialized InterfaceClass object

Postcondition

The item of type DataType that is at a particular place in the the list is returned and the InterfaceClass is unchanged

Algorithm

The local object is dereferenced and its method getAtCurrent is called in a return statement to retrieve the item of type DataType at that place in the list

Exceptions

None	

Parameters

None	

Returns

A value or object of type DataType that is in the list at a given place in the list (DataType)

Note

None

4.2.2.2 template<class DataType > bool InterfaceClass< DataType >::operator++( ) [virtual]

Implementation of InterfaceClass overloaded increment operator.

Iterates forward through the list toward the end

Precondition

Assumes an initialized InterfaceClass object

Postcondition

The current location in the list is incremented if it's not currently already at the end of the list

Algorithm

An if statement checks whether the current location in the list is at the end or not, if so false is returned, otherwise the method nextNode is called in a return statement which moves to the next location in the list

Exceptions

None	

Parameters

None	

Returns

A bool is returned corresponding to whether or not the incrementor operator was successful in iterating through the list (bool)

Note

None

4.2.2.3 template<class DataType > bool InterfaceClass< DataType >::operator-- ( ) [virtual]

Implementation of InterfaceClass overloaded decrement operator.

Iterates back through the list toward the beginning

Precondition

Assumes an initialized InterfaceClass object

Postcondition

The current location in the list is decremented if it's not currently already at the beginning of the list

Algorithm

An if statement checks whether the current location in the list is at the beginning or not, if so false is returned, otherwise the method prevNode is called in a return statement which moves to the previous location in the list

Exceptions

None	

Parameters

None	

Returns

A bool is returned corresponding to whether or not the decrementor operator was successful in iterating through the list (bool)

Note

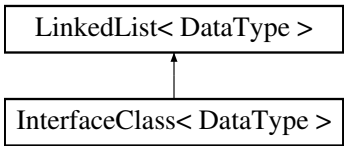
None

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

- InterfaceClass.h
- InterfaceClass.cpp

4.3 LinkedList< DataType > Class Template Reference

Inheritance diagram for LinkedList< DataType >:



Public Member Functions

- LinkedList ()

  Implementation of LinkedList default constructor.
- LinkedList (const LinkedList< DataType > &copiedList)

  Implementation of LinkedList copy constructor.
- ~LinkedList ()

  Implementation of LinkedList destructor.
- const LinkedList< DataType > & operator= (const LinkedList< DataType > &rhList)

  Implementation of LinkedList overloaded assignment operator.
- void clearList ()

  Implementation of LinkedList method to clear the list of all nodes.
- void insertAtCurrent (const DataType &inData)

  Implementation of LinkedList method to insert an item of type DataType into the list.
- bool removeAtCurrent (DataType &removedItem)

  Implementation of LinkedList method to remove an item of type DataType from the list.
- bool nextNode ()

  Implementation of LinkedList method to move the DataNode pointer currentPtr to the next node.
- bool prevNode ()

  Implementation of LinkedList method to move the DataNode pointer currentPtr to the previous node.
- void toEnd ()

  Implementation of LinkedList method to move the DataNode pointer currentPtr to the end of the list.
- void toBeginning ()

  Implementation of LinkedList method to move the DataNode pointer currentPtr to the beginning of the list.
- bool atEnd () const

  Implementation of LinkedList method that determines whether or not the DataNode pointer currentPtr is at the end of the list.
- bool atBeginning () const

  Implementation of LinkedList method that determines whether or not the DataNode pointer currentPtr is at the beginning of the list.
- const DataType & getAtCurrent () const

  Implementation of LinkedList method to retrieve the item held in the node at the DataNode pointer currentPtr.
- void showLLStructure (char IDChar) const

  Implementation of LinkedList method to print the list to the screen.

Static Public Attributes

- static const int **LARGE_STR_LEN** = 100

Private Member Functions

- void copyList (const LinkedList< DataType > &copiedList)

  Implementation of LinkedList private method to copy on LinkedList object into another.
- void clearListHelper (DataNode< DataType > *workingPtr)

  Implementation of LinkedList private method to delete a node in the list.

Private Attributes

- DataNode< DataType > * **listHead**
- DataNode< DataType > * **currentPtr**

4.3.1 Constructor & Destructor Documentation

4.3.1.1 template<class DataType > LinkedList< DataType >::LinkedList ( )

Implementation of LinkedList default constructor.

Initializers used to set data members to NULL

Precondition

Assumes an uninitialized LinkedList object

Postcondition

Initialized LinkedList object with data members set to default values

Algorithm

Initializers used to set data members of type DataNode pointer to NULL

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.3.1.2 template<class DataType > LinkedList< DataType >::LinkedList (const LinkedList< DataType > & *copiedList*)

Implementation of LinkedList copy constructor.

The private method copyList is called to copy the list passed in as a parameter into the calling list

Precondition

Assumes an uninitialized LinkedList object

Postcondition

LinkedList object with the same nodes and data members as the list passed in as a parameter

Algorithm

Initializers are used to set default values to data members and then the private method copyList is called on the list passed in as a parameter

Exceptions

None	

Parameters

in	*copiedList*	A const LinkedList reference parameter which will be copied into the calling list (LinkedList<DataType>)

Returns

None

Note

None

4.3.1.3 template<class DataType > LinkedList< DataType >::~LinkedList ( )

Implementation of LinkedList destructor.

The member method clearList is called to delete all nodes in the list

Precondition

Assumes an initialized LinkedList object

Postcondition

All nodes contained in the LinkedList object deleted and data members set to default values

Algorithm

The member method clearList is called to remove all nodes from the list

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.3.2 Member Function Documentation

4.3.2.1 template<class DataType > bool LinkedList< DataType >::atBeginning ( ) const

Implementation of LinkedList method that determines whether or not the DataNode pointer currentPtr is at the beginning of the list.

If the DataNode pointer currentPtr is at the beginning of the list then a corresponding bool is returned

Precondition

Assumes an initialized LinkedList object

Postcondition

A bool corresponding to the position of currentPtr is returned and the LinkedList object is unchanged

Algorithm

An if statement checks whether or not the current node is listHead and if so then true is returned, otherwise false is returned

Exceptions

None	

Parameters

None	

Returns

A bool corresponding to whether or not currentPtr is at the beginning of the list (bool)

Note

None

4.3.2.2 template<class DataType > bool LinkedList< DataType >::atEnd ( ) const

Implementation of LinkedList method that determines whether or not the DataNode pointer currentPtr is at the end of the list.

If the DataNode pointer currentPtr is at the end of the list then a corresponding bool is returned

Precondition

Assumes an initialized LinkedList object

Postcondition

A bool corresponding to the position of currentPtr is returned and the LinkedList object is unchanged

Algorithm

An if statement checks whether or not the next node is listHead and if so then true is returned, otherwise false is returned

Exceptions

None	

Parameters

None	

Returns

A bool corresponding to whether or not currentPtr is at the end of the list (bool)

Note

None

4.3.2.3 template<class DataType > void LinkedList< DataType >::clearList ( )

Implementation of LinkedList method to clear the list of all nodes.

Memory for all nodes is freed and data members are set to default values

Precondition

Assumes an initialized LinkedList object

Postcondition

All nodes are deleted and data members are set to default values

Algorithm

An if statement checks whether of not listHead is NULL and if not then the nodes are cleared recursively by checking whether there is more one node, the recursive case in which the method clearListHelper is on currentPtr and clearList is called again, or there is only one node, the base case in which listHead is deleted and data members are set to default values

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.3.2.4 template<class DataType > void LinkedList< DataType >::clearListHelper (DataNode< DataType > * *workingPtr*) [private]

Implementation of LinkedList private method to delete a node in the list.

Takes in a pointer to a node as a parameter, deletes it and links up adjacent nodes

Precondition

Assumes an initialized LinkedList object

Postcondition

The node to which the DataNode pointer passed in as a parameter points to is deleted and adjacent nodes are linked

Algorithm

An if statement checks that the parameter is not NULL and that it's not listHead, if not then, after an if statement checks whether the parameter pointing at currentPtr and if so then prevNode is called, the nodes adjacent to the one the parameter points to are linked and the node is deleted

Exceptions

None	

Parameters

in	*workingPtr*	DataNode pointer which corresponds to the node to be deleted (DataNode<-DataType>)

Returns

None

Note

None

4.3.2.5 template<class DataType > void LinkedList< DataType >::copyList (const LinkedList< DataType > & *copiedList*) [private]

Implementation of LinkedList private method to copy on LinkedList object into another.

The calling LinkedList object has the nodes and data members of the list passed in as parameter

Precondition

Assumes a LinkedList object

Postcondition

LinkedList object with the same nodes and data members as the list passed in as a parameter

Algorithm

An if statement checks that the local object and the LinkedList object passed in as a parameter are not the same, if they're not then a call to the method clearList clears the calling object's nodes and if the object passed in as a parameter is not empty then an event controlled loop goes through it and new nodes are created for the calling object

Exceptions

None	

Parameters

in	*copiedList*	A const LinkedList reference parameter which will be copied into the calling list (LinkedList<DataType>)

Returns

None

Note

None

4.3.2.6 template<class DataType > const DataType & LinkedList< DataType >::getAtCurrent ( ) const

Implementation of LinkedList method to retrieve the item held in the node at the DataNode pointer currentPtr.

If there are nodes in the list then the item at currentPtr is returned

Precondition

Assumes an initialized LinkedList object and that currentPtr isn't NULL

Postcondition

The item at currentPtr is returned and the LinkedList object is unchanged

Algorithm

An if statement checks whether or not currentPtr is NULL and if not then the item at currentPtr is returned

Exceptions

None	

Parameters

None	

Returns

The item of type DataType at the currentPtr is returned (DataType)

Note

None

4.3.2.7 template<class DataType > void LinkedList< DataType >::insertAtCurrent (const DataType & *inData*)

Implementation of LinkedList method to insert an item of type DataType into the list.

The item passed in as a parameter is added to the list at the node pointed to by currentPtr

Precondition

Assumes an initialized LinkedList object

Postcondition

The LinkedList object has the item of type DataType passed in as a parameter inserted into the list at currentPtr

Algorithm

An if statement checks whether there are nodes in the list and if there aren't then the first one is made at listHead, otherwise a new one is made and then linked up with currentPtr and the node before it

Exceptions

None	

Parameters

`in`	*inData*	A const reference parameter of type DataType that is to be inserted into the list (DataType)

Returns

None

Note

None

4.3.2.8 template<class DataType > bool LinkedList< DataType >::nextNode ( )

Implementation of LinkedList method to move the DataNode pointer currentPtr to the next node.

If the list is not empty and there is more than one node, the DataNode pointer next is moved to the next node

Precondition

Assumes an initialized LinkedList object

Postcondition

The LinkedList data member currentPtr points to the next node in the list

Algorithm

An if statement checks whether or not the list is empty and if there is more than one node in the list, if so then currentPtr is moved to next node

Exceptions

None	

Parameters

None	

Returns

A bool corresponding to whether or not the DataNode pointer was successfully moved (bool)

Note

None

4.3.2.9 template<class DataType > const LinkedList< DataType > & LinkedList< DataType >::operator= (const LinkedList< DataType > & *rhList*)

Implementation of LinkedList overloaded assignment operator.

The private method copyList is called to copy the list passed in as a parameter into the calling list

Precondition

Assumes a LinkedList object

Postcondition

LinkedList object with the same nodes and data members as the list passed in as a parameter

Algorithm

A call to the private method copyList by the calling object copies the nodes and data members from the list passed in as a parameter into the local object

Exceptions

None	

Parameters

`in`	*rhList*	A const LinkedList reference parameter which will be copied into the calling list (LinkedList<DataType>)

Returns

The local LinkedList object (LinkedList<DataType>)

Note

None

4.3.2.10 template<class DataType > bool LinkedList< DataType >::prevNode ( )

Implementation of LinkedList method to move the DataNode pointer currentPtr to the previous node.

If the list is not empty and there is more than one node, the DataNode pointer previous is moved to the previous node

Precondition

Assumes an initialized LinkedList object

Postcondition

The LinkedList data member currentPtr points to the previous node in the list

Algorithm

An if statement checks whether or not the list is empty and if there is more than one node in the list, if so then currentPtr is moved to previous node

Exceptions

None	

Parameters

None	

Returns

A bool corresponding to whether or not the DataNode pointer was successfully moved (bool)

Note

None

4.3.2.11 template<class DataType > bool LinkedList< DataType >::removeAtCurrent (DataType & *removedItem*)

Implementation of LinkedList method to remove an item of type DataType from the list.

The item at the node pointed to by currentPtr is removed and stored in parameter of type DataTye passed into the method

Precondition

Assumes an initialized LinkedList object

Postcondition

The LinkedList object has the item of type DataType at currentPtr removed and stored in the parameter passed into the method

Algorithm

An if statement checks whether there are nodes in the list and if there are then a function call to the getAtCurrent gets the item and assigns it to the reference parameter passed in and then if that was the only node the clearList is called, otherwise the node at currentPtr is deleted and adjacent nodes are linked together

Exceptions

None	

Parameters

out	*removedItem*	A reference parameter of type DataType that accepts the item at the node pointed to by currentPtr (DataType)

Returns

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

Note

None

4.3.2.12 template<class DataType > void LinkedList< DataType >::showLLStructure (char *IDChar*) const

Implementation of LinkedList method to print the list to the screen.

The items in the list are printed to the screen along with a char that identifies the list

Precondition

Assumes an initialized LinkedList object

Postcondition

The structure of the LinkedList object is printed to the screen and the list is unchanged

Algorithm

An if statement checks whether there are nodes in the list, if not then an indication that the list is empty is printed to the screen, otherwise the list identifier is printed out and an event controlled moves through the list and prints each item to the screen

Exceptions

None	

Parameters

in	*IDChar*	A char which acts as an identifier for the list to be printed to the screen (char)

Returns

None

Note

None

4.3.2.13 template<class DataType > void LinkedList< DataType >::toBeginning ( )

Implementation of LinkedList method to move the DataNode pointer currentPtr to the beginning of the list.

If the DataNode pointer currentPtr isn't already at the beginning of the list then it is moved to the first node

Precondition

Assumes an initialized LinkedList object

Postcondition

The LinkedList data member currentPtr points to the first node in the list

Algorithm

An if statement checks whether or not currentPtr is already at the beginning of the list and if not then the method prevNode is called in and event controlled loop until it reaches the beginning

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.3.2.14 template<class DataType > void LinkedList< DataType >::toEnd ( )

Implementation of LinkedList method to move the DataNode pointer currentPtr to the end of the list.

If the DataNode pointer currentPtr isn't already at the end of the list then it is moved to the last node

Precondition

Assumes an initialized LinkedList object

Postcondition

The LinkedList data member currentPtr points to the last node in the list

Algorithm

An if statement checks whether or not currentPtr is already at the end of the list and if not then the method nextNode is called in an event controlled loop until it reaches the end

Exceptions

None	

Parameters

None	

Returns

None

Note

None

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

- LinkedList.h
- LinkedList.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

5 File Documentation

5.1 InterfaceClass.cpp File Reference

Implementation for InterfaceClass class.

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

5.1.1 Detailed Description

Implementation for InterfaceClass class.

Author

Bryan Kline

Implements all member methods for the InterfaceClass class

Version

1.00 Bryan Kline (15 February 2016)

None

5.2 InterfaceClass.h File Reference

Definition file for InterfaceClass class.

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

Classes

- class InterfaceClass< DataType >

5.2.1 Detailed Description

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

Version

1.00 Michael Leverington (06 February 2016) Original code

None

5.3 LinkedList.cpp File Reference

Implementation for LinkedList class.

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

5.3.1 Detailed Description

Implementation for LinkedList class.

Author

Bryan Kline

Implements all member methods for the LinkedList class

Version

1.00 Bryan Kline (15 February 2016)

None

5.4 LinkedList.h File Reference

Definition file for LinkedList class.

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

Classes

- class DataNode< DataType >
- class LinkedList< DataType >

5.4.1 Detailed Description

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

Version

1.00 Michael Leverington (06 February 2016) Original code

None

5.5 StudentType.cpp File Reference

Implementation file for StudentType class.

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

5.5.1 Detailed Description

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

Version

1.00 (07 September 2015)

Requires StudentType.h

Index