PA10 - HashClass

Generated by Doxygen 1.8.6

Wed Apr 20 2016 13:14:28

Contents

1 Class Index

1.1 Class List

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

2 File Index

2.1 File List

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

3 Class Documentation

3.1 DataNode< DataType > Struct Template Reference

Public Member Functions

- DataNode ()

  Implementation of DataNode struct default constructor.

Public Attributes

- NodeState **usedState**
- DataType **nodeData**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 template<typename DataType > DataNode< DataType >::DataNode ( )

Implementation of DataNode struct default constructor.

An initializer is used to set one of the struct data members to its default value

Precondition

Assumes a DataNode struct

Postcondition

The data member usedState is set to its default value

Algorithm

An initializer is used to set the data member usedState to its default value

Exceptions

None	

Parameters

None	

Returns

None

Note

Initializer used

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

- HashClass.h
- HashClass.cpp

3.2 HashClass< DataType > Class Template Reference

Public Member Functions

- HashClass ()
  Implementation of HashClass default constructor.
- HashClass (const HashClass< DataType > &copied)
  Implementation of HashClass copy constructor.
- ~HashClass ()
  Implementation of HashClass destructor.
- const HashClass & operator= (const HashClass< DataType > &rhData)
  Implementation of HashClass overloaded assignment operator.
- bool setTableLength (int newTableLength, bool clearListFlag, int &maxProbes, int &totalProbes)
  Implementation of HashClass method to resize the hash table.
- void setHashLetterCount (int newHashLetterCount)
  Implementation of HashClass method to set the data member hashLetterCount.
- void setProbeAttempts (int newNumProbeAttempts)
  Implementation of HashClass method to set the data member maxProbeAttempts.
- bool addItem (const DataType &newData, int &probeAttempts)
  Implementation of HashClass to add and item to the table.
- int findItem (const DataType &dataItem, int &probeAttempts) const
  Implementation of HashClass method to find and item in the table.
- bool removeItem (const DataType &dataItem, int &probeAttempts)
  Implementation of HashClass method to remove an item from the hash table.
- void clearList ()
  Implementation of HashClass method to clear the hash table.
- bool isEmpty () const
  Implementation of HashClass method to determine whether the hash table is empty.
- void showStructure () const
  Implementation of HashClass method to print out the hash table to screen.

Static Public Attributes

- static const char **TAB** = '\t'
- static const int **DEFAULT_HASH_TABLE_LENGTH** = 10
- static const int **DEFAULT_HASH_LETTER_COUNT** = 3
- static const int **DEFAULT_PROBE_ATTEMPT_LIMIT** = 10
- static const int **FAILED_PROBE_PROCESS** = -1
- static const int **STD_STR_LEN** = 50
- static const int **LARGE_STR_LEN** = 100
- static const bool **CLEAR_LIST** = true
- static const bool **NO_LIST_CLEAR** = false

Private Member Functions

- int hash (const DataType &dataItem, int workingTableLength) const

  Implementation of HashClass method to calculate the hash value of an item to be added to the table.
- int addItemHelper (DataNode< DataType > *destHashTable, const DataType &newData)

  Implementation of HashClass method to assist in adding an item to the table.
- bool resizeList (int newSize, bool clearFlag, int &maxProbes, int &totaProbes)

  Implementation of HashClass method to resize the hash table.
- void copyList (const DataNode< DataType > *copiedList)

  Implementation of HashClass class method to copy a hash table into the local object.
- int toPower (int base, int exponent) const

  Implementation of HashClass class method to calculate the result of raising a number to a power.

Private Attributes

- int **tableLength**
- int **maxProbeAttempts**
- int **hashLetterCount**
- DataNode< DataType > * **hashList**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 template<typename DataType > HashClass< DataType >::HashClass ( )

Implementation of HashClass default constructor.

Initializers are used to set data members to default values

Precondition

Assumes an uninitialized HashClass object

Postcondition

Data members are set to default values and memory is allocated for the hash table

Algorithm

Initializers are used to set data members to default values and allocate memory for the hash table

Exceptions

None	

Parameters

None	

Returns

None

Note

Initializers used

3.2.1.2 template<typename DataType > HashClass< DataType >::HashClass (const HashClass< DataType > & *copied*)

Implementation of HashClass copy constructor.

Initializers are used to set data members to default values and allocate memory for the hash table and then the values of one HashClass object are copied into the other

Precondition

Assumes an uninitialized HashClass object

Postcondition

The local object has the values of the HashClass object passed in as a parameter copied into it

Algorithm

Initializers are used to set data members to default values and allocate memory for the hash table and the method copyList copies over the values from the HashClass object passed in as a parameter into the local object

Exceptions

None	

Parameters

in	*copied*	A reference parameter of type HashClass corresponding to the hash table to be copied into the local object (HashClass<DataType>)

Returns

None

Note

Initializers used

3.2.1.3 template<typename DataType > HashClass< DataType >::~HashClass ( )

Implementation of HashClass destructor.

The memory allocated to the hash table is freed

Precondition

Assumes an initialized HashClass object

Postcondition

The memory allocated to the hash table is freed

Algorithm

An if statement checks whether hashList is NULL and if it isn't then the memory allocated to it is freed and then it's set to NULL

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.2.2 Member Function Documentation

3.2.2.1 template<typename DataType > bool HashClass< DataType >::addItem (const DataType & *newData,* int & *probeAttempts*)

Implementation of HashClass to add and item to the table.

Calls addItemHelper to add an item to the list and then returns a corresponding to whether or not the item was used

Precondition

Assumes an initialized HashClass object

Postcondition

The item is added to the hash table if, after quadratic probing if necessary, it finds an available place in the table

Algorithm

The method addItemHelper is called and the result is stored in probeAttempts, if the value in probeAttempts is less than the max maximum possible attempts then true is returned, otherwise false is returned

Exceptions

None	

Parameters

in	*newData*	A reference parameter of type DataType which corresponds to the item to be added to the table (DataType)
in	*probeAttempts*	An int corresponding to the number of times the table has been probed (int)

Returns

A bool is returned corresponding to whether or not the item was added to the table

Note

None

3.2.2.2 template<typename DataType > int HashClass< DataType >::addItemHelper (DataNode< DataType > * *destHashTable,* const DataType & *newData*) [private]

Implementation of HashClass method to assist in adding an item to the table.

Called by addItem to assist in adding an item to the hash table and the number of probes is returned

Precondition

Assumes an initialized HashClass object

Postcondition

An item is added to the table if there is an available place for it after the maximum number of probe attempts

Algorithm

The method has is called and assigned to hashValue, then a counter controlled loop attempts to place the item at the index corresponding to hashValue, checking its usedState data member with an if statement, placing it there if possible and if not calculating the next index with call to toPower, in either case the number of probe attempts is returned

Exceptions

None	

Parameters

in	*destHashHelper*	A pointer of type DataNode which points to the hash table (DataNode<DataType>)
in	*newData*	A reference parameter of type DataType which corresponds to the item to be added to the table (DataType)

Returns

An int is returned corresponding to the number of probe attempts (int)

Note

None

3.2.2.3 template<typename DataType > void HashClass< DataType >::clearList ( )

Implementation of HashClass method to clear the hash table.

All the nodes in the table have their usedState set to UNUSED

Precondition

Assumes an initialized HashClass object

Postcondition

All the items in the table are set to UNUSED thereby clearing the table

Algorithm

An if statement checks whether or not the pointer to the table is NULL, if not then a counter controlled loop moves through the table and sets all the nodes' usedState data members to UNUSED

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.2.2.4 template<typename DataType > void HashClass< DataType >::copyList (const DataNode< DataType > * *copiedList*) [private]

Implementation of HashClass class method to copy a hash table into the local object.

The contents of one hash table are copied into the other

Precondition

Assumes an initialized HashClass object, both tables are of the same size, and that the DataType that the node holds has an overloaded assignment operator

Postcondition

The hash table holds the same nodes as those of the parameter passed in

Algorithm

A counter controlled loop moves through both hash lists and copies the nodes from one to the other by accessing the struct node data members directly

Exceptions

None	

Parameters

in	*copiedList*	A pointer to a DataNode object which points to the hash table to be copied to the local object (DataNode<DataType>*)

Returns

None

Note

None

3.2.2.5 template<typename DataType > int HashClass< DataType >::findItem (const DataType & *dataItem,* int & *probeAttempts*) const

Implementation of HashClass method to find and item in the table.

Returns the index of the table if the item is found, otherwise -1 is returned

Precondition

Assumes an initialized HashClass object

Postcondition

The index of the item is returned if it's in the table and it's unchanged

Algorithm

An event controlled loop, while the number of rehash attempts is below the maximum, goes through the table checking first the initial hash index and then the quadratically probed index after that if not found, an if statement returning the index if it's found at any point, otherwise -1 is returned

Exceptions

None	

Parameters

in	*dataItem*	A reference parameter of type DataType corresponding to the item to be searched for in the table (DataType)
in	*probeAttempts*	An int corresponding to the number of probe attempts, needed for the hash method

Returns

An int corresponding to the index at which an item was found in the table (int)

Note

None

3.2.2.6 template<typename DataType > int HashClass< DataType >::hash (const DataType & *dataItem,* int *workingTableLength*) const [private]

Implementation of HashClass method to calculate the hash value of an item to be added to the table.

The method of one of the parameters is called to get the hash value of the item to be added to the table

Precondition

Assumes an initialized HashClass object and that the parameter dataItem has a hashing method

Postcondition

The hash value of the item to be added to the table is returned and the hash table is unchanged

Algorithm

The hashing method for the parameter dataItem is called with a default number of letters to be considered for the calculation the length of the table passed in as parameters

Exceptions

None	

Parameters

in	*dataItem*	A reference parameter of type DataType which corresponds to the item to be hashed and added to the table (DataType)
in	*workingTable-Length*	A int corresponding to the length of the hash table for the hash calculation (int)

Returns

An int corresponding to the value returned from the call to the parameter dataItem's hashing method (int)

Note

None

3.2.2.7 template<typename DataType > bool HashClass< DataType >::isEmpty ( ) const

Implementation of HashClass method to determine whether the hash table is empty.

Returns a bool based on whether or not the hash table is empty

Precondition

Assumes an initialized HashClass object

Postcondition

A bool is returned based on whether or not the table is empty and it's unchanged

Algorithm

An if statement checks whether or not the pointer to the table is NULL, if not then a counter controlled loop moves through the table and if at any point there is a node with its data member usedState set to USED then false is returned, if not then true is returned

Exceptions

None	

Parameters

None	

Returns

A bool is returned corresponding to whether or not the hash table is empty (bool)

Note

None

3.2.2.8 template<typename DataType > const HashClass< DataType > & HashClass< DataType >::operator= (const HashClass< DataType > & *rhData*)

Implementation of HashClass overloaded assignment operator.

The data members and hash table from one HashClass object are copied into another

Precondition

Assumes an initialized HashClass object

Postcondition

The data members and hash table from one HashClass object are copied into the other

Algorithm

An if statement checks whether the local object is the same as the parameter and if not then the method resizeList is called, data members are assigned the values of the HashClass object passed in as a parameter, the method copyList is called to copy the hash table, and then the local object is returned

Exceptions

None	

Parameters

in	*rhHashTable*	A reference parameter of type HashClass which corresponds to the HashClass object to be copied into the local object (HashClass)

Returns

None

Note

None

3.2.2.9 template<typename DataType > bool HashClass< DataType >::removeItem (const DataType & *dataItem,* int & *probeAttempts*)

Implementation of HashClass method to remove an item from the hash table.

The method findItem is called and if it's found then it is removed from the table

Precondition

Assumes an initialized HashClass object

Postcondition

If the item to be removed is in the table it is removed

Algorithm

The method findItem is called and the result is stored in index, if that value is -1 then false is retuned as it's not in the list, otherwise the struct node at that index is set to UNUSED and true is returned

Exceptions

None	

Parameters

`in`	*dataItem*	A reference parameter of type DataType which corresponds to the item to be removed from the table (DataType)
`in`	*probeAttempts*	An int which corresponds to the number of times the item will be rehashed by the findItem method (int)

Returns

A bool is returned corresponding to whether or not the item was successfully removed from the hash table (bool)

Note

None

3.2.2.10 template<typename DataType > bool HashClass< DataType >::resizeList (int *newSize,* bool *clearFlag,* int & *maxProbes,* int & *totaProbes*) [private]

Implementation of HashClass method to resize the hash table.

Either clears the table and resizes it to any size or the items in the table remain but the size can only be increased

Precondition

Assumes an initialized HashClass object

Postcondition

Either clears the list and resizes it or simply resizes it if the new size is greater than the current size

Algorithm

An if statement checks whether or not clearFlag is set, if so then the table is deleted, a table with the new size is created and true is returned, otherwise an if statement checks if the new size is greater than the current, if not then false is returned otherwise a new table is created with the old size, the table is copied into it with a counter conrolled loop, the old table is deleted then a new one is created with the new size and then a counter controlled loop calls addItem on each one to place them into the new table if it's not used and true is returned

Exceptions

None	

Parameters

in	*newSize*	An int corresponding to the new size of the table (int)
in	*clearFlag*	A bool corresponding to whether or not the list will be cleared (bool)
in	*maxProbes*	A reference parameter of type int corresponding to the maximum number of times the table can be probed (int)
in	*totaProbes*	A reference parameter of type int corresponding to the number of probe attempts (int)

Returns

None

Note

None

3.2.2.11 template<typename DataType > void HashClass< DataType >::setHashLetterCount (int *newHashLetterCount*)

Implementation of HashClass method to set the data member hashLetterCount.

Assigns the data member hashLetterCount to the value parameter

Precondition

Assumes an initialized HashClass object

Postcondition

The data member hashLetterCount is reset

Algorithm

The data member hashLetterCount assigned the value of the parameter passed in

Exceptions

None	

Parameters

in	*newHashLetterCount*	An int corresponding to the new value of the data member hashLetterCount (int)

Returns

None

Note

None

3.2.2.12 template<typename DataType > void HashClass< DataType >::setProbeAttempts (int *newNumProbeAttempts*)

Implementation of HashClass method to set the data member maxProbeAttempts.

Assigns the data member maxProbeAttempts to the value parameter

Precondition

Assumes an initialized HashClass object

Postcondition

The data member maxProbeAttempts is reset

Algorithm

The data member maxProbeAttempts assigned the value of the parameter passed in

Exceptions

None	

Parameters

in	*maxProbe-Attempts*	An int corresponding to the new value of the data member maxProbeAttempts (int)

Returns

None

Note

None

3.2.2.13 template<typename DataType > bool HashClass< DataType >::setTableLength (int *newTableLength,* bool *clearListFlag,* int & *maxProbes,* int & *totalProbes*)

Implementation of HashClass method to resize the hash table.

The result of a call to resizeList is returned

Precondition

Assumes an initialized HashClass object

Postcondition

The hash table is resized

Algorithm

The parameters are passed into the method resizeList and its result is returned

Exceptions

None	

Parameters

in	*newTableLength*	An int corresponding to the new length of the hash table (int)
in	*clearListFlag*	A bool corresponding to whether or not the table will be cleared (bool)
in	*maxProbes*	An int corresponding to the maximum number of probes (int)
in	*totalProbes*	An int corresponding to the number of times probed (int)

Returns

A bool is returned corresponding to whether or not the resize of the table was successful (bool)

Note

None

3.2.2.14 template<typename DataType > void HashClass< DataType >::showStructure ( ) const

Implementation of HashClass method to print out the hash table to screen.

The items in the hash table are printed to the screen, UNUSED is displayed if there is no item at a position in the table

Precondition

Assumes an initialized HashClass object and that the items in the table have a toString method

Postcondition

The hash table is printed to the screen and is unchanged

Algorithm

A counter controlled loop moves through the table and if the node at a given index has its data member used-State set to USED then it's printed to the screen with call to toString, otherwise UNUSED is printed to the screen

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.2.2.15 template<typename DataType > int HashClass< DataType >::toPower (int *base,* int *exponent*) const [private]

Implementation of HashClass class method to calculate the result of raising a number to a power.

An int is returned corresponding to the result of taking one parameter to the power of the other

Precondition

Assumes an initialized HashClass object and positive int parameters

Postcondition

The result of the calculation is returned and the hash table is unchanged

Algorithm

An if statement checks that the exponent is greater than zero and if so then a counter controlled loop multiplies the base by itself that many times and returns the result, otherwise one is returned

Exceptions

None	

Parameters

in	*base*	An int corresponding to the base in the calculation (int)
in	*exponent*	An int corresponding to the exponent in the calculation (int)

Returns

An int corresponding to the result of the power calculation (int)

Note

None

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

- HashClass.h
- HashClass.cpp

3.3 MedType Class Reference

Public Member Functions

- **MedType** (const char ∗patientName, const char ∗medCodeNum, char patientGender)
- **MedType** (const MedType &newMedObject)
- const MedType & **operator=** (const MedType &rhMed)
- void **setAccount** (const char ∗patientName, const char ∗medicalCodeNum, char patientGender)
- void **getAccount** (char ∗patientName, char ∗medicalCodeNum, char &patientGender) const
- int **compareTo** (const MedType &rhMed) const throw (logic_error)
- void **toString** (char ∗medStr)
- int **hash** (int numLetters, int hashTableLength)

Static Public Attributes

- static const char **NULL_CHAR** = '\0'
- static const char **COMMA** = ','
- static const char **SPACE** = ' '
- static const char **BASE_STR_LEN** = 20
- static const int **STD_NAME_LEN** = 100

Private Member Functions

- void **copyString** (char ∗destination, const char ∗source) const
- void **concatString** (char ∗destination, const char ∗source) const
- void **concatChar** (char ∗destination, const char source) const
- int **getStrLen** (const char ∗str) const
- char **toUpper** (char letter) const

Private Attributes

- char ∗ **name**
- char ∗ **medCodeNum**
- char **gender**

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

- MedType.h
- MedType.cpp

3.4 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**

3.4.1 Constructor & Destructor Documentation

3.4.1.1 SimpleTimer::SimpleTimer ( )

Default constructor.

Constructs Timer class

Parameters

None	

Note

set running flag to false

3.4.1.2 SimpleTimer::~SimpleTimer ( )

Default constructor.

Destructs Timer class

Parameters

None	

Note

No data to clear

3.4.2 Member Function Documentation

3.4.2.1 void SimpleTimer::start ( )

Start control.

Takes initial time data

Parameters

None	

Note

None

3.4.2.2 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 File Documentation

4.1 HashClass.cpp File Reference

Implementation file for HashClass class.

```
#include "HashClass.h"
```

Variables

- const int **TWO_POWER** = 2

4.1.1 Detailed Description

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

Version

1.20 Bryan Kine (20 April 2016) Implementation of remaining methods

1.10 Michael Leverington (06 April 2016) Updated with probing

1.00 Michael Leverington (06 November 2015) Original code

Requires HashClass.h

4.2 HashClass.h File Reference

Definition file for HashClass.

```
#include <iostream>
```

Classes

- struct DataNode< DataType >
- class HashClass< DataType >

Enumerations

- enum **NodeState** { **USED**, **UNUSED** }

4.2.1 Detailed Description

Definition file for HashClass. Specifies all data and other members of the HashClass

Version

1.10 Michael Leverington (06 April 2016) Updated with probing

1.00 Michael Leverington (06 November 2015) Original code

None

4.3 MedType.cpp File Reference

Implementation file for MedType class.

```
#include "MedType.h"
#include <iostream>
```

4.3.1 Detailed Description

Implementation file for MedType class. Implements member actions of the MedType class

Author

Michael Leverington

Version

1.00 (30 October 2015)

Requires MedType.h

4.4 MedType.h File Reference

Definition file for MedType class.

```
#include <ostream>
#include <stdexcept>
```

Classes

- class MedType

Variables

- const char **NAME_DEFAULT** [] = "Name Default"
- const char **CODE_NUM_DEFAULT** [] = "Code Num Default"

4.4.1 Detailed Description

Definition file for MedType class. Specifies all data of the MedType class, along with the constructor, MedType class is entered and stored as a string

Author

Michael Leverington

Version

1.00 (30 October 2015)

None

4.5 SimpleTimer.cpp File Reference

Implementation file for SimpleTimer class.

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

4.5.1 Detailed Description

Implementation file for SimpleTimer class.

Author

Michael Leverington

Implements member methods for timing

Version

1.00 (11 September 2015)

Requires SimpleTimer.h.

4.6 SimpleTimer.h File Reference

Definition file for simple timer class.

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

Classes

- class SimpleTimer

4.6.1 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

Index