PA09 - Heaps

Generated by Doxygen 1.8.6

Wed Apr 13 2016 14:31:44

Contents

1 Class Index

1.1 Class List

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

ClassType	**2**
DataNode< DataType >	**3**
HeapType< DataType >	**4**
RoomType	**17**
SimpleTimer	**18**
SimpleVector< DataType >	**20**

2 File Index

2.1 File List

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

ClassType.cpp **Implementation file for ClassType class**	**31**
ClassType.h **Definition file for ClassType class**	**31**
HeapType.cpp **HeapType class implementation**	**32**
HeapType.h **Definition file for HeapType class**	**32**
PA09.cpp **Driver program to implement the recursive backtracking operation**	**33**
RoomType.cpp **Implementation file for RoomType class**	**34**
RoomType.h **Definition file for RoomType class**	**34**

3 Class Documentation

3.1 ClassType Class Reference

Public Member Functions

- **ClassType** (char ∗inClsName, int inClsSize)
- const ClassType & **operator=** (const ClassType &rhClass)
- void **setClassData** (char ∗inClsName, int inClsSize)
- void **setClassAvailable** (bool flagState)
- bool **classIsAvailable** ()
- int **getSizeRequest** ()
- int **compareTo** (const ClassType &otherClass) const
- int **compareKey** (const ClassType &otherClass) const
- void **toString** (char ∗outString) const
- int **toInt** () const

Static Public Attributes

- static const int **NO_CLASS** = -1
- static const int **STD_STR_LEN** = 80
- static const char **NULL_CHAR** = '\0'

Private Member Functions

- void **copyString** (char ∗destination, const char ∗source) const
- int **compareStrings** (const char ∗oneStr, const char ∗otherStr) const
- int **getStrLen** (const char ∗str) const
- char **toLower** (char testChar) const

Private Attributes

- char **className** [STD_STR_LEN]
- int **classSize**
- bool **classAvailable**

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

- ClassType.h
- ClassType.cpp

3.2 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**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 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

3.3 HeapType< DataType > Class Template Reference

Public Member Functions

- HeapType ()

  Implementation of HeapType class default constructor.
- HeapType (const HeapType< DataType > &copiedVector)

  Implementation of HeapType class copy constructor.
- ∼HeapType ()

  Implementation of HeapType class destructor.
- const HeapType< DataType > & operator= (const HeapType< DataType > &rhVector)

  Implementation of HeapType class overloaded assignment operator.
- void showHPStructure (char IDChar)

  Implementation of HeapType class method to print the heap to the screen in the form of a tree.
- int getSize () const

  Implementation of HeapType class method to return the size of the heap.
- bool isEmpty () const

  Implementation of HeapType class method to check if the heap is empty.
- void add (const DataType &inData)

  Implementation of HeapType class method to add an item to the heap.
- bool remove (DataType &removeData)

  Implementation of HeapType class method to remove an item from the heap.
- void clear ()

  Implementation of HeapType class method to set the size of the heap to zero.

Static Public Attributes

- static const int **DEFAULT_CAPACITY** = 10
- static const int **BASE_TWO** = 2
- static const char **SPACE** = ' '

Private Member Functions

- void shiftUp (int currentIndex)

  Implementation of HeapType class method to shift an item up in the heap if necessary.
- void shiftDown (int currentIndex)

  Implementation of HeapType class method to shift an item down in the heap if necessary.
- void checkForResize ()

  Implementation of HeapType class method to resize the heap if it becomes full.
- void copyHeapVector (DataType ∗destination, const DataType ∗source, int count)

  Implementation of HeapType class method to copy one vector into another.
- void swap (int one, int other)

  Implementation of HeapType class method to swap two items in the heap.
- int getHeight () const

  Implementation of HeapType class method to determine the height of the tree representation of the heap.
- void getSpacing (int row, int &firstSpaces, int &dividerSpaces) const

  Implementation of HeapType class method to calculate the proper spacing for the showHPStructure.
- int toPower (int base, int exponent) const

  Implementation of HeapType class method to calculate the result of raising a number to a power.
- void displayInt (int valueIndex) const

  Implementation of HeapType class method to printed out an item in the heap to the screen.
- void displayChars (int numChars, char outChar) const

  Implementation of HeapType class method to print a char to the screen in an amount specificied by an input parameter.

Private Attributes

- int **heapCapacity**
- int **heapSize**
- DataType ∗ **heapVector**

3.3.1 Constructor & Destructor Documentation

3.3.1.1 template<class DataType > HeapType< DataType >::HeapType ( )

Implementation of HeapType class default constructor.

Sets data members to default values and allocates memory for heap

Precondition

> Assumes an uninitialized HeapType object

Postcondition

> An initialized HeapType object with data members set to default values

Algorithm

> Initializers are used to set data members to default values and memory is allocated for the heap

Exceptions

None	

Parameters

None	

Returns

> None

Note

> Initializers used

3.3.1.2 template<class DataType > HeapType< DataType >::HeapType (const HeapType< DataType > & *copiedVector*)

Implementation of HeapType class copy constructor.

Copies the data members and values from the heap passed in as a parameter into the calling heap

Precondition

> Assumes an uninitialized HeapType object

Postcondition

> The heap is created with all values as the heap passed in as a parameter

Algorithm

> Initializers are used to set data members to default values and then the overloaded assignment operator is called on the local object

Exceptions

None	

Parameters

`in`	*copiedVector*	A reference parameter of type HeapType which corresponds to the heap to be copied into the local object (HeapType<DataType>)

Returns

None

Note

Initializers used

3.3.1.3 template<class DataType > HeapType< DataType >::∼HeapType ( )

Implementation of HeapType class destructor.

Deletes heap and sets data members to default values

Precondition

Assumes initialized HeapType object

Postcondition

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

Algorithm

Data members are set to default values and if the heap pointer is not NULL then the heap is deleted

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.3.2 Member Function Documentation

3.3.2.1 template<class DataType > void HeapType< DataType >::add (const DataType & *inData*)

Implementation of HeapType class method to add an item to the heap.

The item is added to the last place in the heap and then is recursively shifted up with a call to shiftUp if necessary

Precondition

Assumes an initialized HeapType object holding items of a type that has an assignment operator

Postcondition

The item is added to the heap and is shifted up if necessary, possibly changing the order of the heap

Algorithm

A call to checkForResize increases the size of the heap if it's full, the item is added to the last position in the heap, shiftUp is called on that index to shift it up if necessary and the size of the heap is incremented

Exceptions

None	

Parameters

`in`	*inData*	A reference parameter of type DataType which corresponds to the item to be added to the heap (DataType)

Returns

None

Note

None

3.3.2.2 template<class DataType > void HeapType< DataType >::checkForResize ( ) `[private]`

Implementation of HeapType class method to resize the heap if it becomes full.

The vector is dynamically resized to 1.25 the size if it becomes full

Precondition

Assumes an initialized HeapType object holding items of a type that has an assignment operator

Postcondition

The capacity of the heapVector is increased by 1.25 times

Algorithm

An if statement checks that heapVector is not NULL and that the heap is full and if those conditions are met then an array of type DataType is created with a new capacity 1.25 times larger and then a counter controlled loop copies the items from the old vector into the new and then deletes the old vector and points heapVector to the new one

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.3.2.3 template<class DataType > void HeapType< DataType >::clear ( )

Implementation of HeapType class method to set the size of the heap to zero.

The data member heapSize is set to zero

Precondition

Assumes an initialized HeapType object

Postcondition

The size of the heap is changed to zero and any items in the heap are lost

Algorithm

The data member heapSize is set to zero

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.3.2.4 template<class DataType > void HeapType< DataType >::copyHeapVector (DataType ∗ *destination,* const DataType ∗ *source,* int *count*) [private]

Implementation of HeapType class method to copy one vector into another.

Takes in two vectors and copies one into the other

Precondition

Assumes an positive int for the capacity of the source vector

Postcondition

The destination pointer points to a vector of the same size, holding the same values, as that of the source pointer and the heapVector data member points to destination

Algorithm

An if statement checks whether destination points to a vector, if so its deleted, then memory is allocated to of size count, and then a counter controlled loop copies in the items from the source vector into the destination vector and then the data member heapVector points to the destination vector

Exceptions

None	

Parameters

`in`	*destination*	A DataType pointer which points to the vector which will have the values of the other vector copied into it (DataType∗)
`in`	*source*	A DataType pointer which points to the vector which will have its values copied into the other vector (DataType∗)
`in`	*count*	An int corresponding to the capacity of the source vector (int)

Returns

None

Note

None

3.3.2.5 template< class DataType > void HeapType< DataType >::displayChars (int *numChars,* char *outChar*) const `[private]`

Implementation of HeapType class method to print a char to the screen in an amount specificied by an input parameter.

A char parameter is printed to the screen in the amount specified by a parameter

Precondition

Assumes an initialized HeapType object

Postcondition

A char is printed to the screen some number of times and the heap is unchanged

Algorithm

A counter controlled loop prints to the screen the char input to the method as a parameter

Exceptions

None	

Parameters

in	*numChars*	An int corresponding to the number of a particular char that is to be printed to the screen (int)
in	*outChar*	A char that is to be printed to the screen (char)

Returns

None

Note

None

3.3.2.6 template<class DataType > void HeapType< DataType >::displayInt (int *valueIndex*) const [private]

Implementation of HeapType class method to printed out an item in the heap to the screen.

The value specified by the input parameter is formatted and printed to the screen

Precondition

Assumes an initialized HeapType object that holds items of a type that can be compared to an int using the less than and greater than operators

Postcondition

The value passed in as a parameter is printed to the screen and the heap is unchanged

Algorithm

If statements check how many digits are in the parameter passed in to be printed to the screen, if one digit it's centered between spaces, if two digits then a leading zero is added, if three then it's simply printed to the screen

Exceptions

None	

Parameters

in	*valueIndex*	An int corresponding to the value in the heap at which is to be printed to the screen (int)

Returns

None

Note

None

3.3.2.7 template<class DataType > int HeapType< DataType >::getHeight ( ) const [private]

Implementation of HeapType class method to determine the height of the tree representation of the heap.

Returns an int corresponding to the height of the tree form of the heap based on indices

Precondition

Assumes an initialized HeapType object with a size greater than zero

Postcondition

Returns the height of the tree form of the heap and the heap is unchanged

Algorithm

An if statement checks that the heap is not empty and if it's not then an event controlled loop considers the final index in the heap and while an int, result, is not greater than the final index it's assigned the value of $2^{\wedge}$(height) - 1 as height starts at zero and then is incremented and once result is greater than last index height is decremented and it's returned

Exceptions

None	

Parameters

None	

Returns

An int corresponding to the height of the tree representation of the heap (int)

Note

None

3.3.2.8 template<class DataType > int HeapType< DataType >::getSize ( ) const

Implementation of HeapType class method to return the size of the heap.

Returns the size of the heap

Precondition

Assumes an initialized HeapType object

Postcondition

The size of the heap is returned and the heap is unchanged

Algorithm

The int heapSize is returned

Exceptions

None	

Parameters

None	

Returns

An int corresponding to the size of the heap (ine)

Note

None

3.3.2.9 template<class DataType > void HeapType< DataType >::getSpacing (int *row*, int & *firstSpaces*, int & *dividerSpaces*) const [private]

Implementation of HeapType class method to calculate the proper spacing for the showHPStructure.

The parameters hold values corresponding to the leading and diving spaces in the tree representation of the heap

Precondition

Assumes that the parameter row is a positive int

Postcondition

The parameters hold the values of the calculation and the heap is unchanged

Algorithm

Both parameters are assigned the value of the calculations 2^(row) - 1 for leading spaces and 2^(row + 1) - 3 for dividing spaces

Exceptions

None	

Parameters

out	*firstSpaces*	An int corresponding to the number of leading spaces (int)
out	*dividerSpaces*	An int corresponding to the number of dividing spaces (int)

Returns

None

Note

None

3.3.2.10 template<class DataType > bool HeapType< DataType >::isEmpty ( ) const

Implementation of HeapType class method to check if the heap is empty.

Returns a bool corresponding to whether or not the heap is empty

Precondition

Assumes an initialized HeapType object

Postcondition

A bool is returned and the heap is unchanged

Algorithm

An if statement checks whether or no the heap pointer is NULL or if the size of the heap is zero, if so then true is returned, otherwise false

Exceptions

None	

Parameters

None	

Returns

A bool corresponding to whether or not the heap is empty (bool)

Note

None

3.3.2.11 template<class DataType > const **HeapType**< DataType > & **HeapType**< DataType >::operator= (const **HeapType**< DataType > & *rhVector*)

Implementation of HeapType class overloaded assignment operator.

Copies the data members and values from the heap passed in as a parameter into the calling heap

Precondition

Assumes an uninitialized HeapType object

Postcondition

The calling heap has the values of the heap passed in as a parameter copied into it

Algorithm

An if statement checks that the calling object and the parameter are not the same object and if not then a call to copyHeapVector copies the vector and then the local data members are assigned to those of the parameter and then the local object is returned

Exceptions

None	

Parameters

`in`	*rhVector*	A reference parameter of type HeapType which corresponds to the heap to be copied into the local object (HeapType<DataType>)

Returns

A the local object of type HeapType is returned (HeapType<DataType>)

Note

None

3.3.2.12 template<class DataType > bool **HeapType**< DataType >::remove (DataType & *removeData*)

Implementation of HeapType class method to remove an item from the heap.

The top item of the heap is removed, then the last is placed on top and shifted down if necessary and a bool is returned corresponding to whether it was successful

Precondition

Assumes an initialized HeapType object holding items of a type that has an assignment operator

Postcondition

The largest item is removed from the top of the heap and then the last item is placed on the top and shifted down with a call to shiftDown if necessary

Algorithm

An if statement checks that the heap is not empty and if not then the item at the top of the heap is assigned to the parameter removeData, the item at the last position in the heap is placed on the top, the size of the heap is decremented and then shiftDown is called to shift it down if necessary and true is returned if that was successful, otherwise false is returned

Exceptions

None	

Parameters

out	*removeData*	A reference parameter of type DataType which accepts the item removed if it's there (DataType)

Returns

A bool is returned corresponding to whether or not an item could be removed from the heap (bool)

Note

None

3.3.2.13 template<class DataType > void HeapType< DataType >::shiftDown (int *currentIndex*) [private]

Implementation of HeapType class method to shift an item down in the heap if necessary.

Recursively shifts the item at the specified index down the heap if it is smaller than either of its children

Precondition

Assumes an initialized HeapType object

Postcondition

The item at the specified index is swapped with the larger of its children if it's smaller

Algorithm

An if statement checks that the left child of currentIndex is within the size of the vector, if not it's the base case, if so then another if checks that the right child is also within the size of the vector, if so then both right and left are compared to the parent, and then to each other, if statements swap the item at currentIndex with either the left or right child depending on which is larger if the parent is smaller, and if the right child is not within the size of the vector then the parent is swapped with the left child if it's smaller, lastly the function is called recursively on the child it was swapped with in either case

Exceptions

None	

Parameters

in	*currentIndex*	An int corresponding to the index of the item to be shifted down if necessary (int)

Returns

None

Note

None

3.3.2.14 template<class DataType > void HeapType< DataType >::shiftUp (int *currentIndex*) [private]

Implementation of HeapType class method to shift an item up in the heap if necessary.

Recursively shifts the item at the specified index up the heap if it is larger than its parent

Precondition

Assumes an initialized HeapType object

Postcondition

The item at the specified index is swapped with its parent if it's larger

Algorithm

An if statement checks that currentIndex isn't at the top of the heap, if so then it's the base case, and if not it enters the recursive case which checks if the item at the index is greater than its parent and if so a call to swap swaps them and then the method is called recursively on the parent of the currentIndex

Exceptions

None	

Parameters

in	*currentIndex*	An int corresponding to the index of the item to be shifted up if necessary (int)

Returns

None

Note

None

3.3.2.15 template<class DataType > void HeapType< DataType >::showHPStructure (char *IDChar*)

Implementation of HeapType class method to print the heap to the screen in the form of a tree.

The heapVector is printed out in such a way so as to resemble a tree

Precondition

Assumes an initialized HeapType object

Postcondition

The heap is printed out in the form of a tree and the heap is unchanged

Algorithm

A nested counter controlled loop calls getSpacing to determine the proper spacing, prints out leading spaces with a call to displayChars, both in the outter loop, then the inner loop prints out items in the tree and dividing spaces with calls to displayChars and displayInt

Exceptions

None	

Parameters

`in`	*IDChar*	A char which acts as an identifier of the heap (char)

Returns

None

Note

None

3.3.2.16 template<class DataType > void **HeapType**< DataType >::swap (int *one,* int *other*) `[private]`

Implementation of HeapType class method to swap two items in the heap.

Swaps two items in the heap at the indices indicated by the input parameters

Precondition

Assumes an initialized HeapType object holding items of a type that has an assignment operator

Postcondition

Two items in the heapVector at the specified indices are swapped

Algorithm

An if statement checks that the heap contains items and if so then it uses one a temporary position to swap the items at the specified indices

Exceptions

None	

Parameters

`in`	*one*	An int corresponding to an item in the heap at the first index (int)

in	*other*	An int corresponding to an item in the heap at the next index (int)

Returns

None

Note

None

3.3.2.17 template<class DataType > int HeapType< DataType >::toPower (int *base,* int *exponent*) const [private]

Implementation of HeapType 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 HeapType object and positive int parameters

Postcondition

The result of the calculation is returned and the heap 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:

- HeapType.h
- HeapType.cpp

3.4 RoomType Class Reference

Public Member Functions

- **RoomType** (char ∗bldgName, int roomNum, int roomCap)
- const RoomType & **operator=** (const RoomType &rhRoom)

- void **setRoomData** (char ∗inBldgName, int inRoomNumber, int inRoomCapacity, int inAssocClsIndex=NO_-CLASS)
- void **setAssociatedIndex** (int inAssocIndex)
- int **getAssociatedIndex** () const
- int **getRoomCapacity** () const
- int **compareTo** (const RoomType &otherRoom) const
- int **compareKey** (const RoomType &otherRoom) const
- void **toString** (char ∗outString) const
- int **toInt** () const

Static Public Attributes

- static const int **NO_CLASS** = -1
- static const int **STD_STR_LEN** = 80
- static const char **NULL_CHAR** = '\0'

Private Member Functions

- void **copyString** (char ∗destination, const char ∗source) const
- int **compareStrings** (const char ∗oneStr, const char ∗otherStr) const
- int **getStrLen** (const char ∗str) const
- char **toLower** (char testChar) const

Private Attributes

- char **buildingName** [STD_STR_LEN]
- int **roomNumber**
- int **roomCapacity**
- int **associatedClassIndex**

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

- RoomType.h
- RoomType.cpp

3.5 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**

3.5.1 Constructor & Destructor Documentation

3.5.1.1 SimpleTimer::SimpleTimer ( )

Default constructor.

Constructs Timer class

Parameters

None	

Note

set running flag to false

3.5.1.2 SimpleTimer::∼SimpleTimer ( )

Default constructor.

Destructs Timer class

Parameters

None	

Note

No data to clear

3.5.2 Member Function Documentation

3.5.2.1 void SimpleTimer::start ( )

Start control.

Takes initial time data

Parameters

None	

Note

None

3.5.2.2 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

3.6 SimpleVector< DataType > Class Template Reference

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**

3.6.1 Constructor & Destructor Documentation

3.6.1.1 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

3.6.1.2 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

3.6.1.3 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

3.6.1.4 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

3.6.2 Member Function Documentation

3.6.2.1 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

3.6.2.2 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

3.6.2.3 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

3.6.2.4 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

3.6.2.5 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

3.6.2.6 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

3.6.2.7 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

3.6.2.8 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

3.6.2.9 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

3.6.2.10 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

3.6.2.11 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

3.6.2.12 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 File Documentation

4.1 ClassType.cpp File Reference

Implementation file for ClassType class.

```
#include "ClassType.h"
```

4.1.1 Detailed Description

Implementation file for ClassType class. Implements the methods of the ClassType class

Version

1.10 Michael Leverington (11 March 2016) Update for use with room information

1.00 Michael Leverington (30 January 2016) Initial development

Requires ClassType.h

4.2 ClassType.h File Reference

Definition file for ClassType class.

```
#include <cstdio>
#include <iostream>
```

Classes

- class ClassType

4.2.1 Detailed Description

Definition file for ClassType class. Specifies all data of the ClassType class, along with the constructor

Version

1.20 Michael Leverington (11 March 2016) Updated for use with room information

1.00 Michael Leverington (07 September 2015) Original code

None

4.3 HeapType.cpp File Reference

HeapType class implementation.

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

Variables

- const int **BASE_TWO** = 2
- const int **TWO** = 2
- const int **THREE** = 3
- const int **TWO_DIGIT** = 10
- const int **THREE_DIGIT** = 100
- const float **RESIZE** = 1.25

4.3.1 Detailed Description

HeapType class implementation. Implementation of HeapType class methods

Version

1.00 Bryan Kline (10 April 2016) Oringial code

None

4.4 HeapType.h File Reference

Definition file for HeapType class.

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

Classes

- class HeapType< DataType >

4.4.1 Detailed Description

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

Version

1.00 Michael Leverington (02 April 2016) Original code

None

4.5 PA09.cpp File Reference

Driver program to implement the recursive backtracking operation.

```
#include <iostream>
#include <cstring>
#include "SimpleTimer.h"
#include "ClassType.h"
#include "RoomType.h"
#include "HeapType.cpp"
#include "SimpleVector.cpp"
```

Enumerations

- enum **FILE_DATA_CODES** { **STRING_CAPTURED**, **START_CLASSROOMS**, **END_CLASSROOMS**, **START_CLASS_REQUESTS**, **END_CLASS_REQUESTS** }

Functions

- int **getALine** (istream &consoleIn, char ∗inString, char delimiterChar=SEMI_COLON)
- bool **getANumber** (istream &consoleIn, int &number)
- void **displayList** (const HeapType< RoomType > &roomList, const SimpleVector< ClassType > &classList)
- bool **fitClassRooms** (const HeapType< RoomType > &roomHeap, const HeapType< ClassType > &classHeap, SimpleVector< ClassType > &classList)
- void **printSpaces** (int numSpaces)
- int **main** ()

Variables

- const int **MAX_STR_LEN** = 100
- const int **STD_STR_LEN** = 50
- const int **MAX_NUM_ROOMS** = 25
- const bool **SHOW_INPUT** = false
- const bool **SHOW_TIMER** = false
- const char **ENDLINE_CHAR** = '\n'
- const char **CARRIAGE_RETURN_CHAR** = '\r'
- const char **NULL_CHAR** = '\0'
- const char **SPACE** = ' '
- const char **COMMA** = ','
- const char **SEMI_COLON** = ';'

4.5.1 Detailed Description

Driver program to implement the recursive backtracking operation. None

Version

1.10 Bryan Kline (13 April 2016) Modified to include fitClassRooms function

1.00 Michael Leverington (02 April 2016) Original code

Requires Heaptype.cpp, SimpleVector, SimpleTimer.h, ClassType.h, RoomType.h, iostream, cstring

4.6 RoomType.cpp File Reference

Implementation file for RoomType class.

```
#include "RoomType.h"
```

4.6.1 Detailed Description

Implementation file for RoomType class. Implements the methods of the RoomType class

Version

1.10 Michael Leverington (11 March 2016) Update for use with room information

1.00 Michael Leverington (30 January 2016) Initial development

Requires RoomType.h

4.7 RoomType.h File Reference

Definition file for RoomType class.

```
#include <cstdio>
#include <iostream>
```

Classes

- class RoomType

4.7.1 Detailed Description

Definition file for RoomType class. Specifies all data of the RoomType class, along with the constructor

Version

1.20 Michael Leverington (11 March 2016) Updated for use with room information

1.00 Michael Leverington (07 September 2015) Original code

None

4.8 SimpleTimer.cpp File Reference

Implementation file for SimpleTimer class.

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

4.8.1 Detailed Description

Implementation file for SimpleTimer class.

Author

Michael Leverington

Implements member methods for timing

Version

1.00 (11 September 2015)

Requires SimpleTimer.h.

4.9 SimpleTimer.h File Reference

Definition file for simple timer class.

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

Classes

- class SimpleTimer

4.9.1 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

4.10 SimpleVector.cpp File Reference

Implementation file for SimpleVector class.

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

4.10.1 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

4.11 SimpleVector.h File Reference

Definition file for SimpleVector class.

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

Classes

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

4.11.1 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