PA07 Extended BSTClass

Generated by Doxygen 1.8.6

Thu Mar 17 2016 19:39:47

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 BSTClass< DataType > Class Template Reference

Public Member Functions

- BSTClass ()

  Implementation of BSTClass class default constructor.
- BSTClass (const BSTClass< DataType > &copied)

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

  Implementation of BSTClass class destructor.
- const BSTClass & operator= (const BSTClass< DataType > &rhData)

  Implementation of BSTClass class overloaded assignment operator.
- void clear ()

  Implementation of BSTClass class method to clear the tree.
- void insert (const DataType &newData)

  Implementation of BSTClass class method to insert an item into the tree.
- bool find (DataType &searchDataItem) const

  Implementation of BSTClass class to search for an item in the tree.

- bool [remove](const DataType &dataItem)

  Implementation of [BSTClass] class that removes a node in the tree.
- bool [isEmpty] () const

  Implementation of [BSTClass] class method that determines whether or not the tree is empty.
- void [preOrderTraversal] () const

  Implementation of [BSTClass] class to print the values in the tree to the screen.
- void [inOrderTraversal] () const

  Implementation of [BSTClass] class to print the values in the tree to the screen.
- void [postOrderTraversal] () const

  Implementation of [BSTClass] class to print the values in the tree to the screen.
- void [showBSTStructure] (char IDChar)

  Implementation of [BSTClass] method to print out the structure of the tree to the screen.
- int [getHeight] ([BSTNode]< DataType > ∗workingPtr)

  Implementation of [BSTClass] method to get the height of the tree or subtree.
- void [showBSTStructureHelper] ([BSTNode]< DataType > ∗workingPtr, int treeHeight, int currentRow, int print-Row, bool &newLine)

  Implementation of [BSTClass] method to print out the structure of the tree to the screen.
- int [powerFunction] (int base, int exponent)

  Implementation of [BSTClass] method to calculate the result of raising a number to a power.

Static Public Attributes

- static const char **TAB** = '\t'
- static const char **SPACE** = ' '
- static const int **MAX_DATA_LEN** = 80
- static const int **INITIALIZE** = 101
- static const int **AT_BOTTOM** = 102
- static const int **GET_RESULT** = 103

Private Member Functions

- void [copyTree] ([BSTNode]< DataType > ∗&workingPtr, const [BSTNode]< DataType > ∗sourcePtr)

  Implementation of [BSTClass] class method that copies the structure and values from one tree into another.
- void [clearHelper] ([BSTNode]< DataType > ∗workingPtr)

  Implementation of [BSTClass] class method to clear the tree.
- void [insertHelper] ([BSTNode]< DataType > ∗&workingPtr, const DataType &newData)

  Implementation of [BSTClass] class method to insert a new item into the tree.
- bool [findHelper] ([BSTNode]< DataType > ∗workingPtr, DataType &searchDataItem) const

  Implementation of [BSTClass] class to search for an item in the tree.
- bool [removeHelper] ([BSTNode]< DataType > ∗&workingPtr, const DataType &removeDataItem)

  Implementation of [BSTClass] class that removes a node in the tree.
- void [preOrderTraversalHelper] ([BSTNode]< DataType > ∗workingPtr) const

  Implementation of [BSTClass] class method to traverse the tree and print its structure to the screen.
- void [inOrderTraversalHelper] ([BSTNode]< DataType > ∗workingPtr) const

  Implementation of [BSTClass] class method to traverse the tree and print its structure to the screen.
- void [postOrderTraversalHelper] ([BSTNode]< DataType > ∗workingPtr) const

  Implementation of [BSTClass] class method to traverse the tree and print its structure to the screen.
- int [getHeightHelper] ([BSTNode]< DataType > ∗workingPtr)

  Implementation of [BSTClass] method to get the height of the tree or subtree.
- void [spacePrinter] (int height, bool &newLine)

  Implementation of [BSTClass] method to print spaces before the nodes in the tree.
- void [dashPrinter] (bool &newLine, int height, int dashes)

  Implementation of [BSTClass] method to print out dashes to the screen when a node isn't present in the tree.

Private Attributes

- BSTNode< DataType > ∗ **rootNode**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 template<class DataType > BSTClass< DataType >::BSTClass ( )

Implementation of BSTClass class default constructor.

The data member has a default value assigned to it

Precondition

Assumes an uninitialized BSTClass object

Postcondition

The data member has a default value

Algorithm

An initializer is used to set the root to NULL

Exceptions

None	

Parameters

None	

Returns

None

Note

Intializer used

3.1.1.2 template<class DataType > BSTClass< DataType >::BSTClass (const BSTClass< DataType > & *copied*)

Implementation of BSTClass class copy constructor.

Creates a tree with the same structure as the object passed in as parameter

Precondition

Assumes an uninitialized BSTClass object

Postcondition

The local object has the same structure as the parameter passed in

Algorithm

An initializer is used to set the root to NULL and then the method copyTree is called to copy the structure of the parameter into the into the local object

Exceptions

None	

Parameters

in	*copied*	A const reference parameter of type BSTClass which will have its values copied into the local object (BSTClass$<$DataType$>$)

Returns

> None

Note

> Intializers used

3.1.1.3 template$<$**class DataType** $>$ **BSTClass**$<$ **DataType** $>$**::**$\sim$**BSTClass ( )**

Implementation of BSTClass class destructor.

Deletes all nodes in the tree and sets the data member to its default value

Precondition

> Assumes an initialized BSTClass object

Postcondition

> All nodes in the tree deleted and the data member set to NULL

Algorithm

> The method clear is called to delete all nodes and set the root to NULL

Exceptions

None	

Parameters

None	

Returns

> None

Note

> None

3.1.2 Member Function Documentation

3.1.2.1 template$<$**class DataType** $>$ **void BSTClass**$<$ **DataType** $>$**::clear ( )**

Implementation of BSTClass class method to clear the tree.

Deletes all nodes in the tree and sets the root to NULL

Precondition

> Assumes an initialized BSTClass object

Postcondition

> All nodes in the tree deleted and the data member set to NULL

Algorithm

> An if statement checks that the tree is not empty and if its not then the method clearHelper is called to delete all nodes and the root is set to NULL

Exceptions

None	

Parameters

None	

Returns

> None

Note

> None

3.1.2.2   template$<$class DataType $>$ void BSTClass$<$ DataType $>$::clearHelper (BSTNode$<$ DataType $>$ ∗ *workingPtr*) `[private]`

Implementation of BSTClass class method to clear the tree.

Deletes the nodes in the tree recursively

Precondition

> Assumes an initialized BSTClass object

Postcondition

> All nodes in the tree are deleted

Algorithm

> An if statement checks whether the left pointer is NULL, if not then the method is called recursively on the left, then another if statement checks if the right is NULL, and if not the method is called recursively on the right, and the node is deleted

Exceptions

None	

Parameters

in	*workingPtr*	A pointer of type DataType corresponding to the current node under consideration (BSTNode<DataType>∗)

Returns

 None

Note

 None

3.1.2.3 template<class DataType > void BSTClass< DataType >::copyTree (BSTNode< DataType > ∗& *workingPtr,* const BSTNode< DataType > ∗ *sourcePtr*) `[private]`

Implementation of BSTClass class method that copies the structure and values from one tree into another.

Recursive calls to the method copy the structure and values from the tree passed in as a parameter into the local object

Precondition

 Assumes an initialized BSTClass object with no nodes

Postcondition

 The local object has the same structure as the parameter passed in

Algorithm

 An if statement checks whether the pointer pointing to the parameter is NULL and if not then then pointer to the local object has a node created for it and then the method is called recursively on the left and right pointers of the tree passed in as a parameter

Exceptions

None	

Parameters

in	*workingPtr*	A reference BSTNode pointer which points to the current node under consideration in the local tree (BSTNode<DataType>∗)
in	*sourcePtr*	A BSTNode pointer which points to the current node under consideration in the tree passed in as a parameter (BSTNode<DataType>∗)

Returns

 None

Note

 None

3.1.2.4 template<class DataType > void BSTClass< DataType >::dashPrinter (bool & *newLine,* int *height,* int *dashes*) `[private]`

Implementation of BSTClass method to print out dashes to the screen when a node isn't present in the tree.

Uses spacePrinter to print out the appropriate number of spaces and then prints out dashes in the amount specified

---

Precondition

Assumes valid parameters

Postcondition

Spaces and dashes printed out to the screen in the amount specified and the tree is unchanged

Algorithm

A counter controlled loop calls spacePrinter and then prints dashes equal to the number of times passed in by the parameter dashes

Exceptions

None	

Parameters

in	*newLine*	A reference parameter of type bool which corresponds to whether or not the first node in the row under consideration has been printed out yet or not (bool)
in	*height*	An int corresponding to the height of the tree or subtree at the node under consideration (int)
in	*dashes*	An int corresponding to the number of sets of dashes to print out to the screen (int)

Returns

None

Note

None

3.1.2.5 template<**class DataType** > **bool BSTClass**< **DataType** >**::find (DataType &** *searchDataItem* **) const**

Implementation of BSTClass class to search for an item in the tree.

The item passed in as a parameter is searched for in the tree and true is returned if it is found

Precondition

Assumes an initialized BSTClass object

Postcondition

A bool corresponding to whether or not the item is found is returned and the tree is unchanged

Algorithm

An if statement checks whether the tree is empty and if not then the method findHelper is called to recursively search for the item

Exceptions

	None	

Parameters

in	*searchDataItem*	A reference parameter of type DataType which corresponds to the item searched for in the tree (DataType)

Returns

A bool corresponding to whether or not the item was found in the tree (bool)

Note

None

3.1.2.6 template< class DataType > bool BSTClass< DataType >::findHelper (BSTNode< DataType > ∗ *workingPtr,* DataType & *searchDataItem*) const `[private]`

Implementation of BSTClass class to search for an item in the tree.

An item is searched for in the tree and if found true is returned

Precondition

Assumes an initialized BSTClass object and that the item searched for has a compareTo method

Postcondition

A bool corresponding to whether or not the item is found is returned and the tree is unchanged

Algorithm

If statements check whether the item passed in is equal to, less than, or greater than the item at the node under consideration, if it's equal then true is returned, if it's either less than or greater than it then in either case an if statement checks whether there is a node to the left or right and if so the method is called on that node otherwise false is returned

Exceptions

	None	

Parameters

in	*workingPtr*	A parameter of type BSTNode pointer which points to the current node under consideration in the recursion (BSTNode<DataType>∗)
in	*searchDataItem*	A const reference parameter of type DataType corresponding to the item searched for in the tree (DataType)

Returns

A bool corresponding to whether or not the item was found in the tree (bool)

Note

The item searched for must have a compareTo method

3.1.2.7   template<**class DataType** > **int BSTClass**< **DataType** >**::getHeight (BSTNode**< **DataType** > ∗ *workingPtr* **)**

Implementation of BSTClass method to get the height of the tree or subtree.

Calls a helper method to recursively determine the height of the tree or subtree

Precondition

> Assumes an initialized BSTClass object

Postcondition

> The height of the tree or subtree under consideration is calculated and the tree is unchanged

Algorithm

> The helper method is called and the result is returned

Exceptions

None	

Parameters

in	*workingPtr*	A pointer of type BSTNode that points to the node currently under consideration (BSTNode<DataType>∗)

Returns

> An int corresponding to the height of the tree (int)

Note

> None

3.1.2.8   template<**class DataType** > **int BSTClass**< **DataType** >**::getHeightHelper (BSTNode**< **DataType** > ∗ *workingPtr* **)** `[private]`

Implementation of BSTClass method to get the height of the tree or subtree.

Recursively determines the height of the tree or subtree at a given node passed in as a parameter

Precondition

> Assumes an initialized BSTClass object

Postcondition

> The height of the tree or subtree under consideration is calculated and the tree is unchanged

Algorithm

> An if statement checks if the workingPtr is NULL, the base case, and if so zero is returned, then the method is called recursively on the left and then the right node pointers, then the values each returns are compared and the greater one plus one is returned

Exceptions

None	

Parameters

in	*workingPtr*	A pointer of type BSTNode that points to the node currently under consideration (BSTNode<DataType>∗)

Returns

An int corresponding to the height of the tree or subtree currently under consideration (int)

Note

None

3.1.2.9 template<class DataType > void BSTClass< DataType >::inOrderTraversal ( ) const

Implementation of BSTClass class to print the values in the tree to the screen.

The tree is traversed in order and the values in the nodes are printed to the screen

Precondition

Assumes an initialized BSTClass object

Postcondition

The values in the tree are printed and the tree is unchanged

Algorithm

An if statement checks whether or not the tree is empty and if not then the method inOrderTraversalHelper is called with the root as a parameter

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.1.2.10 template<class DataType > void BSTClass< DataType >::inOrderTraversalHelper (BSTNode< DataType > ∗ workingPtr) const `[private]`

Implementation of BSTClass class method to traverse the tree and print its structure to the screen.

Recursively moves through the tree and prints the value at each node

Precondition

> Assumes an initialized [BSTClass](#) object

Postcondition

> The values in the nodes of the tree are printed to the screen and the tree is unchanged

Algorithm

> An if statement checks whether the pointer is NULL and if not then the method is called on the left branch, the value at the node is printed to the screen, and then the method is called on the right branch

Exceptions

None	

Parameters

in	*workingPtr*	A parameter of type [BSTNode](#) pointer which points to the current node under consideration in the recursion (BSTNode<DataType>∗)

Returns

> None

Note

> A helper method to preOrderTraversal

3.1.2.11 template<**class DataType** > **void BSTClass**< **DataType** >**::insert (const DataType &** *newData* **)**

Implementation of [BSTClass](#) class method to insert an item into the tree.

Inserts an item into the appropriate location in the tree with a call to a recursive helper method

Precondition

> Assumes an initialized [BSTClass](#) object

Postcondition

> The tree has the item inserted into the appropriate location

Algorithm

> An if statement checks whether the tree is empty, if so then a new node is created at the root, otherwise the recursive helper method is called

Exceptions

None	

Parameters

in	*newData*	A reference parameter of type DataType corresponding to the item to be inserted (DataType)

Returns

> None

Note

> None

3.1.2.12 template<class DataType > void BSTClass< DataType >::insertHelper (BSTNode< DataType > ∗& *workingPtr,* const DataType & *newData*) `[private]`

Implementation of [BSTClass](#) class method to insert a new item into the tree.

The item to be inserted is added to tree with recursive calls to the method

Precondition

> Assumes an initialized [BSTClass](#) object and that the item to be inserted has a compareTo method

Postcondition

> The tree has the item inserted into the appropriate location

Algorithm

> An if statement checks whether the current node pointer is NULL, if not then if statements check whether the item to be inserted is greater or less than the item at the current node, going right or left in either respective case, and then if that node is a leaf then a new node is created, otherwise the method is called on the child node

Exceptions

None	

Parameters

`in`	*workingPtr*	A pointer of type [BSTNode](#) pointer which corresponds to the current node under consideration (BSTNode<DataType>∗)
`in`	*newData*	A reference parameter of type DataType corresponding to the item to be inserted (DataType)

Returns

> None

Note

> The inserted item must have a compareTo method

3.1.2.13 template<class DataType > bool BSTClass< DataType >::isEmpty ( ) const

Implementation of [BSTClass](#) class method that determines whether or not the tree is empty.

If there are no nodes in the tree then true is returned

Precondition

> Assumes an initialized [BSTClass](#) object

Postcondition

> A bool is returned based on whether or not the tree is empty

Algorithm

> An if statement checks whether the root is NULL and if so true is returned

Exceptions

None	

Parameters

None	

Returns

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

Note

None

3.1.2.14 template<class DataType > const BSTClass< DataType > & BSTClass< DataType >::operator= (const BSTClass< DataType > & *rhData*)

Implementation of BSTClass class overloaded assignment operator.

Assigns one BSTClass object to the local object

Precondition

Assumes an initialized BSTClass object

Postcondition

The local object has the same structure as the parameter passed in

Algorithm

An if statement checks whether the local object and the parameter passed in are the same, if not then the method copyTree copies the structure and values of the parameter into the local object

Exceptions

None	

Parameters

in	*rhData*	A const reference parameter of type BSTClass which will be the object which has its structure and values copied into the local object (BSTClass<Data-Type>)

Returns

The local object is returned with this deferenced (BSTClass<DataType>)

Note

None

3.1.2.15 template<class DataType > void BSTClass< DataType >::postOrderTraversal ( ) const

Implementation of BSTClass class to print the values in the tree to the screen.

The tree is traversed in post-order and the values in the nodes are printed to the screen

Precondition

> Assumes an initialized BSTClass object

Postcondition

> The values in the tree are printed and the tree is unchanged

Algorithm

> An if statement checks whether or not the tree is empty and if not then the method postOrderTraversalHelper
> is called with the root as a parameter

Exceptions

None	

Parameters

None	

Returns

> None

Note

> None

3.1.2.16   template$<$ class DataType $>$ void BSTClass$<$ DataType $>$::postOrderTraversalHelper (BSTNode$<$ DataType $>$ $*$ workingPtr) const `[private]`

Implementation of BSTClass class method to traverse the tree and print its structure to the screen.

Recursively moves through the tree and prints the value at each node

Precondition

> Assumes an initialized BSTClass object

Postcondition

> The values in the nodes of the tree are printed to the screen and the tree is unchanged

Algorithm

> An if statement checks whether the pointer is NULL and if not then the method is called on the left branch, then
> the right branch, and then the value at the node is printed to the screen

Exceptions

None	

Parameters

`in`	*workingPtr*	A parameter of type BSTNode pointer which points to the current node under consideration in the recursion (BSTNode$<$DataType$>*$)

Returns

> None

Note

> A helper method to postOrderTraversal

3.1.2.17   template<**class DataType** > **int BSTClass**< **DataType** >**::powerFunction (int** *base,* **int** *exponent* **)**

Implementation of BSTClass 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 positive integers

Postcondition

>   The result of the calculation is returned and the tree 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

3.1.2.18   template<**class DataType** > **void BSTClass**< **DataType** >**::preOrderTraversal (   ) const**

Implementation of BSTClass class to print the values in the tree to the screen.

The tree is traversed in pre-order and the values in the nodes are printed to the screen

Precondition

>   Assumes an initialized BSTClass object

Postcondition

>   The values in the tree are printed and the tree is unchanged

Algorithm

>   An if statement checks whether or not the tree is empty and if not then the method preOrderTraversalHelper is called with the root as a parameter

Exceptions

None	

Parameters

None	

Returns

> None

Note

> None

3.1.2.19   template<**class DataType** > **void BSTClass**< **DataType** >**::preOrderTraversalHelper (BSTNode**< **DataType** > ∗ *workingPtr* **) const** `[private]`

Implementation of [BSTClass](#) class method to traverse the tree and print its structure to the screen.

Recursively moves through the tree and prints the value at each node

Precondition

> Assumes an initialized [BSTClass](#) object

Postcondition

> The values in the nodes of the tree are printed to the screen and the tree is unchanged

Algorithm

> An if statement checks whether the pointer is NULL and if not then the the value at the node is printed to the screen, the method is called on the left branch, and then on the right branch

Exceptions

None	

Parameters

in	*workingPtr*	A parameter of type [BSTNode](#) pointer which points to the current node under consideration in the recursion (BSTNode<DataType>∗)

Returns

> None

Note

> A helper method to inOrderTraversal

3.1.2.20   template<**class DataType** > **bool BSTClass**< **DataType** >**::remove (const DataType &** *dataItem* **)**

Implementation of [BSTClass](#) class that removes a node in the tree.

If the item passed in as a parameter is found the in tree then it is removed

Precondition

Assumes an initialized [BSTClass](#) object

Postcondition

If the item to be removed is in the tree it is removed and the parameter holds the value of the item removed

Algorithm

An if statement checks that the tree is not empty and then calls the method removeHelper with the root and the parameter as arguments

Exceptions

None	

Parameters

`out`	*dataItem*	A const reference parameter of type DataType corresponding to the item to be removed from the tree (DataType)

Returns

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

Note

None

3.1.2.21   template<**class DataType** > **bool BSTClass**< **DataType** >**::removeHelper (BSTNode**< **DataType** > ∗**&** ***workingPtr,* const DataType &** *removeDataItem* **)** `[private]`

Implementation of [BSTClass](#) class that removes a node in the tree.

The item passed in as a parameter is found and removed recursively if it's in the tree

Precondition

Assumes an initialized [BSTClass](#) object

Postcondition

If the item to be removed is in the tree then it is removed and the DataType parameter holds the value of the removed item

Algorithm

If statements check whether the item passed in is equal to, less than, or greater than the item at the node under consideration, if it's less or great then the method is called recursively and if it's equal then it is removed, if it has two children then the greatest item in the left branch is put into its node, if it has only one child then it's linked with the child's child, and if it's a leaf then it is deleted

Exceptions

	None	

Parameters

in	*workingPtr*	A parameter of type BSTNode pointer which points to the current node under consideration in the recursion (BSTNode<DataType>∗)
out	*removeDataItem*	A const reference parameter of type DataType corresponding to the item to be removed from the tree (DataType)

Returns

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

Note

> None

3.1.2.22   template<**class DataType** > **void BSTClass**< **DataType** >**::showBSTStructure (char** *IDChar* **)**

Implementation of BSTClass method to print out the structure of the tree to the screen.

A helper method is called recursively to print out the structure of the tree in a way the resembles the structure of a tree

Precondition

> Assumes an initialized BSTClass object

Postcondition

> The items in the tree are printed out to the screen and the tree is unchanged

Algorithm

> An if statement checks whether or not the tree is empty and if not then a counter controlled loop calls the helper function as many times as there are rows in the tree

Exceptions

	None	

Parameters

in	*IDChar*	A char corresponding to the identifier of the tree (char)

Returns

> None

Note

> None

3.1.2.23   template<**class DataType** > **void BSTClass**< **DataType** >**::showBSTStructureHelper (BSTNode**< **DataType** > ∗ *workingPtr,* **int** *treeHeight,* **int** *currentRow,* **int** *printRow,* **bool &** *newLine* **)**

Implementation of BSTClass method to print out the structure of the tree to the screen.

Recursively moves through the entire tree and prints out the nodes to the screen in a way that resemebles the actual structure of a tree

---

Precondition

> Assumes an initialized BSTClass object

Postcondition

> The items in the tree are printed out to the screen and the tree is unchanged

Algorithm

> An if statement checks whether the pointer to the current node is NULL or not and whether the current row is above the row to be printed, if not then the method ends, the base case, if so then if statements check whether the row to be printed is the current row, if so then the node is printed after spaces, else if there is a node to the left but not the right then the row is incremented, the method is called recursively with the node's left pointer, the row is decremented, calculations are performed with the print row versus the current row and that many dashes or spaces are printed with a call to dashPrinter, likewise, for if there is a node to the right but not to the right, else if there are no child nodes then the number of dashes is calculated and dashPrinter prints out the spaces and dashes, else the method is called recursively on both the left and right subtrees

Exceptions

None	

Parameters

in	*workingPtr*	A pointer to a BSTNode which corresponds to the current node under consideration (BSTNode<DataType>∗)
in	*treeHeight*	An int corresponding to the total height of the tree (int)
in	*currentRow*	An int corresponding to the current row (int)
in	*printRow*	An int corresponding to the current row to be printed (int)
in	*newLine*	A reference parameter of type bool which corresponds to whether or not the first node in the row has been printed (bool)

Returns

> None

Note

> None

3.1.2.24 template<**class DataType** > **void BSTClass**< **DataType** >**::spacePrinter (int** *height,* **bool &** *newLine* **)** `[private]`

Implementation of BSTClass method to print spaces before the nodes in the tree.

The number of spaces to print out is calculated based on the parameters passed in

Precondition

> Assumes valid parameters

Postcondition

> Spaces printed out to the screen in the appropriate number and the tree is unchanged

Algorithm

> If statements check whether the bool newLine is true, if so the first node of the row is to be printed and the number of leading spaces are calculated (2^n - 1), otherwise the number of dividing spaces is calculated ($2^{(n+1)}$ - 3), then a counter controlled loop prints out spaces in that amount

Exceptions

None	

Parameters

in	*height*	An int corresponding the number of sets of dashes to print out to the screen (int)
in	*newLine*	A reference parameter of type bool which corresponds to whether or not the first node in the row under consideration has been printed out yet or not (bool)

Returns

> None

Note

> None

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

- BSTClass.h
- BSTClass.cpp

3.2 BSTNode< DataType > Class Template Reference

Public Member Functions

- BSTNode (const DataType &nodeData, BSTNode ∗leftPtr, BSTNode ∗rightPtr)
  Implementation of BSTNode class parameterized constructor.

Public Attributes

- DataType **dataItem**
- BSTNode< DataType > ∗ **left**
- BSTNode< DataType > ∗ **right**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 template< **class DataType** > **BSTNode**< **DataType** >**::BSTNode (const DataType &** *nodeData,* **BSTNode**< **DataType** > ∗ *leftPtr,* **BSTNode**< **DataType** > ∗ *rightPtr* **)**

Implementation of BSTNode class parameterized constructor.

Initializers used to assign data members to the parameters passed in

Precondition

> Assumes an uninitialized BSTNode object

Postcondition

> An initialized BSTNode object

Algorithm

> Intializers used to set data members to the values passed in as parameters

Exceptions

None	

Parameters

in	*nodeData*	A const reference parameter of type DataType which the node will hold (Data-Type)
in	*leftPtr*	A pointer of type BSTNode to the left child node (BSTNode∗)
in	*rightPtr*	A pointer of type BSTNode to the right child node (BSTNode∗)

Returns

None

Note

Initializers used

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

- BSTClass.h
- BSTClass.cpp

3.3 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**

3.3.1 Constructor & Destructor Documentation

3.3.1.1 SimpleTimer::SimpleTimer ( )

Default constructor.

Constructs Timer class

Parameters

None	

Note

> set running flag to false

3.3.1.2 SimpleTimer::∼SimpleTimer ( )

Default constructor.

Destructs Timer class

Parameters

None	

Note

> No data to clear

3.3.2 Member Function Documentation

3.3.2.1 void SimpleTimer::start ( )

Start control.

Takes initial time data

Parameters

None	

Note

> None

3.3.2.2 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

3.4 StudentType Class Reference

Public Member Functions

- StudentType ()

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

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

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

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

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

    Data serialization.

Static Public Attributes

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

Private Member Functions

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

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

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

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

    Letter to lower case facility.

Private Attributes

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

3.4.1 Constructor & Destructor Documentation

3.4.1.1 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

3.4.1.2 StudentType::StudentType (char ∗ *initStudentName,* int *initUnivIDNum,* char *initGender*)

Initialization constructor.

Constructs StudentType with provided data

Precondition

> assumes uninitialized StudentType object, assumes string max length < STD_STR_LEN

Postcondition

> Initializes all data quantities

Algorithm

> Initializes class by assigning name, Id number, and gender

Exceptions

None	

Parameters

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

Returns

> None

Note

> None

3.4.2 Member Function Documentation

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

String comparison facility.

Compares two strings ignoring case

Precondition

assumes standard string conditions, including NULL_CHAR end

Postcondition

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

Algorithm

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

Exceptions

None	

Parameters

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

Returns

Difference between two strings (int)

Note

None

3.4.2.2 int StudentType::compareTo (const **StudentType** & *otherStudent*) const

Data comparison utility.

Provides public comparison operation for use in other classes

Precondition

Makes no assumption about StudentType data

Postcondition

Provides integer result of comparison such that:

- result < 0 indicates this $<$ other
- result $== 0$ indicates this $==$ other
- result > 0 indicates this $>$ other

Algorithm

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

Exceptions

None	

Parameters

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

Returns

Integer result of comparison process

Note

None

3.4.2.3 void StudentType::copyString (char ∗ *destination,* const char ∗ *source*) const `[private]`

String copy utility.

Copies source string into destination string

Precondition

assumes standard string conditions, including NULL_CHAR end

Postcondition

desination string holds copy of source string

Algorithm

Copies string character by character until end of string character is found, assumes string max length $<$ STD-_STR_LEN

Exceptions

None	

Parameters

out	*Destination*	string
in	*Source*	string

Returns

None

Note

None

3.4.2.4 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

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

Name parsing utility.

Takes full name and breaks into first and last names

Precondition

> assumes standard string conditions, including NULL_CHAR end

Postcondition

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

Algorithm

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

Exceptions

None	

Parameters

out	*lastName*	String containing last name of student
out	*firstName*	String containing first name of student

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

Returns

> None

Note

> None

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

Data setting utility.

Allows resetting data in StudentType

Precondition

> Makes no assumption about StudentType data

Postcondition

> Data values are correctly assigned in StudentType

Algorithm

> Assigns data values to class members

Exceptions

None	

Parameters

in	*studentName*	String name of student
in	*studentID*	Integer value of student ID
in	*gender*	Character identifier for gender

Returns

> Integer result of comparison process

Note

> None

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

Letter to lower case facility.

None

Precondition

> No assumptions are made related to the input data

Postcondition

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

Algorithm

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

Exceptions

None	

Parameters

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

Returns

None

Note

None

3.4.2.8   void StudentType::toString (char ∗ *outString*) const

Data serialization.

Converts data set to string for output by other data types

Precondition

Assumes data is initialized

Postcondition

Provides all data as string containing the name as initials

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

4 File Documentation

4.1 BSTClass.cpp File Reference

Implementation file for BSTClass.

```
#include <iostream>
#include <cstdlib>
#include "BSTClass.h"
```

4.1.1 Detailed Description

Implementation file for BSTClass.

Author

Bryan Kline

Implements all member methods of the BSTNode and BSTClass classes

Version

1.10 Bryan Kline (17 March 2016) Updated for new assignment

1.00 Bryan Kline (10 March 2016) Original implementation

None

4.2 BSTClass.h File Reference

Definition file for BSTClass.

```
#include <iostream>
```

Classes

- class BSTNode< DataType >
- class BSTClass< DataType >

4.2.1 Detailed Description

Definition file for BSTClass. Specifies all member methods of the BSTClass

Version

1.20 Bryan Kline (17 March 2016) Updated for use with new assignment

1.10 Michael Leverington (27 February 2016) Updated for use with new assignment

1.00 Michael Leverington (30 August 2015) Original code

None

4.3 PA07.cpp File Reference

Driver program to exercise the BST class, with account information.

```
#include <iostream>
#include <cstring>
#include "StudentType.h"
#include "BSTClass.cpp"
```

Enumerations

- enum **cCodes** {
  DUMMY, **BC_1_IOT**, **BC_1_ASSGND**, **BC_1_PSTOT**,
  BC_1_PREOT, **BC_1_RI**, **BC_1_FI**, **BC_1_CLRD**,
  BC_1_CPYD, **BC_1_SHTR** }

Functions

- int **getControlCode** (istream &consoleIn)
- bool **getALine** (istream &consoleIn, StudentType &inputData)
- void **displayCodeChoices** ()
- int **main** ()

Variables

- const char **ENDLINE_CHAR** = '\n'
- const char **CARRIAGE_RETURN_CHAR** = '\r'
- const char **NULL_CHAR** = '\0'
- const char **SEMI_COLON** = ';'
- const int **MAX_NAME_LEN** = 50
- const int **MAX_DATA_LEN** = 100
- const bool **INPUT_TEST** = false
- const bool **MANUAL_ENTRY** = false

4.3.1 Detailed Description

Driver program to exercise the BST class, with account information. Allows for testing the BST class, along with a timer class that will be used for evaluation

Version

> 1.10 Michael Leverington (28 February 2016) Updated for new assignment

1.00 Michael Leverington (09 October 2015) Original code

Requires iostream.h, cstring, StudentType.h, BSTClass.cpp

4.4 SimpleTimer.cpp File Reference

Implementation file for SimpleTimer class.

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

4.4.1 Detailed Description

Implementation file for SimpleTimer class.

Author

Michael Leverington

Implements member methods for timing

Version

1.00 (11 September 2015)

Requires SimpleTimer.h.

4.5 SimpleTimer.h File Reference

Definition file for simple timer class.

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

Classes

- class SimpleTimer

4.5.1 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

4.6 StudentType.cpp File Reference

Implementation file for StudentType class.

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

4.6.1 Detailed Description

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

Version

> 1.20 Bryan Kline (17 March 2016) Update with change to parsNames method to input only first and last initials

1.1 0 Michael Leverington (10 February 2016) Update for use with SorterClass

1.00 Michael Leverington (30 January 2016) Initial development

Requires StudentType.h

Index