Project 3 Michalski Kacper 293464

Templated class Dictionary with typenames <Key, Info>. It is implemented as a binary tree of Node structures pointing at 2 other Node structures, until the final one points at 2 NULLs. Node structures each have 4 variables: key of type Key, info of type Info, left of type Node* and right of type Node*. They also have a constructor for creating nodes. Dictionary points at the first Node with a private Node* root. Dictionary has 3 constructors – blank constructor, copy constructor and constructor from arrays, and a destructor.

Method:	**Arguments:**	**Description:**	**Returns:**
height	void	Returns number of levels, calls private height method	Number of levels
printree	void	"draws" tree to cout	void
inorder/preorder /postorder	void	Prints the tree in-, pre-, or postorder to cout	void
remove	Key k	Remove element with Key k, calls private remove method	True if removed, false if not
clear	void	Clears the tree, calls private clear method	void
insert	Key k, Info i	Inserts a node with Key k and Info I, calls private insert method	True if inserted, else false
balance, rightrot, leftrot	Node* head	Private balancing functions	void
search	Info i	Searches for elements with Info I, calls private method search	Number of elements
seek	Key k	Looks for element with Key k	True if found, else false
furthest/ avgfurthest	void	Looks for info at most distant leaves, returns info of leftmost most distant leaf, or the average	Info at furthest leftmost leaf, or average of it
max/min/rightmost /leftmost	void	Return maximum info in the tree, minimum info, maximum key or minimum key	Maximum/minimum values of Info and Key
equals	Node* head, Node*other	Private helper function of ===	True if trees are equal, else false
insertion	Node* other	Private helper function of = and copy constructor, inserts all elements of "other" tree into this tree	void
isempty	void	-	True if tree is empty, else false
rootkey, rootinf	void	Return values at root	Key or Info at root

Operator:	Description:	Returns:
=	Assignment operator, copies the “right” list to the “left” one (deep copy).	The “right” list
==	Comparison operator.	True if trees are equal, else false.
[]	Takes Key as argument	Returns info at key

Iterator:

Operator:	Description:	Returns:
++/--	Right/ left element	*this
***/!**	Access element	Info/Key at iterator position
==	Equals	True if equal, else false
=	Assignment operator	*this
reset	Reset iterator to where it started	Void
finished	-	True if iterator points to NULL, else false
Begin/end	Take iterator to lowest/highest element	*this

Iterator implements 3 constructors: blank, copy and from a Dictionary object.