# Dictionary Class Template
## Documentation


**0. Author**

Name: Jan Radzimiński

Field of study: Computer Science, Faculty of Electronics and Information Technology

Index number: 293052

Group: 102


**1. General Information**

Dictionary project was implemented and compiled using Code Blocks and MinGW compiler, with C++11 standard.

Dictionary is a class template constructed as AVL Tree. Dictionary consists of ‘’Nodes’’ which are elements of structure declared within the class. All struct elements consist of variables of template type Key - tree is constructed with respect to it, and Info which stores data in given Dictionary element. Class supports all basic operation on the Dictionary, such as insertion and deleting of elements by key, printing Dictionary by inorder, preorder and postorder traversals, sketching tree structure, operators, data access etc. (all functions are described later in document). Whole class implementation was placed in file ,,Dictionary.h’’.

**2. Dictionary Class Members**

**Private**:

| **Member:** | **Explanation:** |
|---|---|
| struct Node:<br>Key key;<br>Info info;<br><br>Int bf;<br>Node * left;<br>Node *right; | Nodes are simply elements of the Dictionary consisting of template type ‘Key key’ by which whole tree is constructed, ‘Info info’, and pointer ‘left’ and ‘right’ – pointers to left and right childs of a node. It also has ‘int bf’ which is balance factor of given node = height of left subtree – height of right subtree. |
| Node *root; | Pointer to first (highest) element of a tree. |

**Public**:

| **Member:** | **Explanation:** |
|---|---|
| class Iterator; | Iterator class that moves (only forward) through elements in the tree. |

### 3. Dictionary Class Functions

**Public:**

| **Function:** | **Explanation:** |
|---|---|
| **Constructors**: | |
| Dictionary<Key, Info>( ): | Empty constructor of the class, which assigns root pointer to null. |
| ~Dictionary<Key, Info>( ); | Destructor of the class, which consist of Clear() method, which deletes all the nodes of the structure and assures that there are no memory leaks after destruction of it. |
| Dictionary<Key, Info>(const Dictionary<Key, Info>& source); | Copy constructor of the class, which firstly sets its pointer to null (for Clear() to work properly) and then uses assignment operator to copy source Dictionary to itself. |
| **Operators**: | |
| Dictionary<Key, Info>& **operator=**(const Dictionary<Key, Info>& source); | Assignment operator, which firstly clears (this) Dictionary to assure its empty, and then by use of Rec_Copy() function it copies structure of source to *this. |
| Dictionary<Key, Info> **operator+**(const Dictionary<Key, Info>& source); | Add operator, which adds all Nodes from source Dictionary using Rec_Add() method. |
| Dictionary<Key, Info>& **operator+=(**const Dictionary<Key, Info>& source) | *this = *this + source; |
| friend std::ostream& **operator<<(**std::ostream& os, const Dictionary<K, I>& seq); | Operator << which allows to sketch whole Dictionary structure (horisontaly) in different streams. Its basicly function Sketch() |
| | |

| | |
|---|---|
| **Tree modifiers:** | |
| bool **Add**(const Key &k, const Info &i); | Add function that properly adds new node on proper place in the tree and then, if tree is unbalanced, it calls balance function which makes tree balanced. It also updates all balance factors. If function is called with key that already exists in the tree it does nothing and returns false. Otherwise it returns true. |
| bool **Remove**(const Key &k); | Function which removes nodes with given key from the tree and balance it, using balance function. If such node does not exist it returns false. |
| void **Clear**() | Clear function which uses recursive Clear function and removes all nodes from the tree. |
| **Tree Info Functions:** | |
| bool **Is_Empty**() const; | Function that returns true if tree does not have any nodes. |
| int **Tree_Height**() const; | Function that returns tree height (from root to the lowest node). |
| int **Node_Height**() const; | Function that returns height of given node if it exists in given tree, or -1 if node doesn't exist. |
| bool **Does_Node_Exist**(const Key &k) const; | Function that checks whether node with given key exists in the tree. |
| **Data Access functions:** | |
| Info &**Get_Info**(const Key &k); | Returns reference to info of node with a given key, or throws an exception if such node does not exist. |
| const Info &**Get_Info**(const Key &k) const; | Same as previous one, but for constant element. |
| **Display:** | |
| void **Print_Inorder**(std::ostream& os=cout) const | Function that prints all tree elements by inorder traversal. |
| void **Print_Preorder**(std::ostream& os=cout) const | Function that prints all tree elements by preorder traversal. |

| | |
|---|---|
| void **Print_Postorder**(std::ostream& os=cout) const | Function that prints all tree elements by postorder traversal. |
| void **Sketch**(std::ostream &os=std::cout) const | Function that sketches (draws, prints) whole tree structure horizontally. |

**Private:**

//Remark: these are the functions which take or show Node pointers as arguments, used by public methods of class but not visible to user of the class.

| **Function:** | **Explanation:** |
|---|---|
| Node* **R_Rotation**(Node *rot_node); | Function that perform Single right rotation on rot node, and returns pointer to the node that is supposed to be on the place of original one. Throws exception in case rot node is null, or does nothing and returns rotnode if it doesn't have any childs. |
| Node* **L_Rotation**(Node *rot_node); | Single left rotation on rot node. |
| Node* **LR_Rotation**(Node * rot_node); | Double rotation, firstly left rotation on rot_node left child, and then right rotation on rot_node, returns pointer in the same way as single rotation functions. |
| Node* **RL_Rotation**(Node * rot_node); | Double rotation, firstly right rotation on rot_node right child, and then left rotation on rot node. |
| Node ***Find** (const Key &k, Node *root) const; | Returns pointer to node with given key, starting from root node (or nullptr if such node does not exist). |
| Node ***Find** (const Key &k) | Same as previous, but starting from root. |
| int **Node_Height**(Node *temp) const; | Returns height from temp to lowest node. |
| int **Node_Left_Height**(Node *temp); | Returns height of left subtree of a node. |
| int **Node_Right_Height**(Node *temp); | Returns height of right subtree of a node. |

| | |
|---|---|
| void **Inorder**(std::ostream& os, Node *temp) const; | Function that perform inorder traversal starting from temp node. |
| void **Preorder**(std::ostream& os, Node *temp) const; | Function that perform preorder traversal starting from temp node. |
| void **Postorder**(std::ostream& os, Node *temp) const; | Function that perform postorder traversal starting from temp node. |
| void **BF_Update**(Node *temp); | Function that updates balance factors of nodes. |
| void **Balance_Tree**(Node *temp); | Function that balance whole tree, starting from lowest ones, by performing proper rotations of nodes. It stops when tree is balanced – all balance factors are not smaller than -1 and not bigger than 1. It is called in add and remove functions, so usually it performs only one type of rotation. |
| void **Sketch_Node**(Node *root, int space, std::ostream &os) const; | Helper function for sketch() – it sketches tree from given node. |
| void **Rec_Add**(Node *temp); | Helper function for plus operator – adds nodes starting from temp. |
| Node ***Rec_Copy**(Node *temp); | Helper function for assignment operator – adds nodes in the same order as temp to this. |
| void **Clear**(Node *temp); | Helper function for clear() – it deletes all child of temp, and then temp. |

## 4. Iterator Class Members & Functions

**Members (private):**

| **Member:** | **Explanation:** |
|---|---|
| Node *current; | 'Hidden' pointer to element of the Dictionary. |

**Functions:**

| **Function:** | **Explanation:** |
|---|---|
| Iterator(); | Empty constructor, sets current to null. |
| Iterator(Node *ptr): | **Private** constructor which sets current to ptr. (available for Dictionary class because its friend of iterator class). |
| ~Iterator() | Destructor of class. |
| Iterator(const Iterator& other) | Copy constructor sets this->currant as same as other.current . |
| Iterator& **operator**=(const Iterator& other) | Assignment operator sets this->currant as same as other.current . |
| bool **operator**==(const Iterator& source) const | Returns true if this and source current pointers are the same. |
| bool Go_Left(); | Navigator of iterator, moves to next element in the Dictionary going left and returning true(if its possible – otherwise it stays on null and returns false). |
| bool Go_Right(); | Navigator of iterator, moves to next element in the Dictionary going right and returning true (if its possible – otherwise it stays on null and returns false). |
| Key &Show_Key() const | Returns key of node that current points to or throw exception if current=null; |
| Info &Show_Info() const | Returns Info of node that current points to or throw exception if current=null; |