Department of Computer and Software Engineering – ITU
SE200T: Data Structures & Algorithms

Course Instructor: Usama Bin Shakeel	**Dated: 28th Nov 2024**
Teaching Assistant: Zainab, Sadia & Ryan	**Semester: Fall 2024**
Session: 2024-2028	**Batch: BSSE2023B**

Assignment 14. Implementation of AVL Tree

Name	Roll number	Obtained Marks/35

Checked on: ______________________________

Signature: _____________________________

Submission:

- Email instructor or TA if there are any questions. You cannot look at others' solutions or use others' solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.
- Submission after due time will not be accepted.

In this assignment you have to do following tasks:

Task 1: Ensure that you have installed all three softwares in your personal computer (Github, Cygwin & CLion). Now, accept the assignment posted in the classroom (e.g Google, LMS etc) and after accepting, clone the repository to your computer. Make sure you have logged into the github app with your account.

Task 2: Open Cygwin app, Move to your code directory with following command "cd <path_of_folder>", <path_of_folder> can be automatically populated by dragging the folder and dropping it to the cygwin window.
Run the code through Cygwin, use command "make run", to get the output of the code

Task 3: Solve the given problems, write code using **CLion** or any other IDE.
Task 4: Keep your code in the respective git cloned folder.
Task 5: Commit and Push the changes through the Github App
Task 5: Write the code in separate files **(as instructed**). Ensure that file names are in lowercase (e,g **main.cpp**).

Task 6: Run '**make run**' to run C++ code
Task 7: Run '**make test**' to test the C++ code
Write code in functions, after completing each part, verify through running code using **"make run"** on Cygwin. Make sure to test the code using **"make test".**

Objective

The objective of this assignment is to implement an **AVL Tree**, a self-balancing binary search tree that automatically maintains balance after each insertion, ensuring that the height difference between the left and right subtrees of any node (balance factor) does not exceed 1.

Class: Node

The `Node` class represents an individual node in the AVL Tree.

Data Members:

- **`int key`:** The value stored in the node.
- **`Node* left`:** Pointer to the left child node.
- **`Node* right`:** Pointer to the right child node.
- **`Node* parent`:** Pointer to the parent node.
- **`int height`:** The height of the node (used for balancing).

Member Functions:

1. **Node(int key)**: Initializes the node with the given key and sets the height to 1.
2. **int getKey()**: Returns the key of the node.
3. **Node* getLeft():** Returns the pointer to the left child of the node.
4. **Node* getRight():** Returns the pointer to the right child of the node.
5. **Node* getParent():** Returns the pointer to the parent node.
6. **void setLeft(Node* leftChild):** Sets the left child pointer.
7. **void setRight(Node* rightChild):** Sets the right child pointer.
8. **void setParent(Node* parentNode):** Sets the parent pointer.
9. **int getHeight()**: Returns the height of the node.
10. **void setHeight(int height)**: Sets the height of the node.

Class: AVLTree

The `AVLTree` class manages the AVL Tree operations.

Data Members:

- **`Node* root`**: Pointer to the root of the AVL Tree.

Member Functions:

1. **AVLTree()**: Initializes the tree with the root set to `nullptr`.
2. **Node* getRoot():** Returns root of the AVL tree.
3. **void insert(int key)**: Inserts a new key into the tree and balances it.
4. **void balanceInsert(Node* node):** Balances the tree after insertion.
5. **Node* search(int key)**: Searches for a node with the given key and returns a pointer to it.
6. **int getBalanceFactor(Node* node)**: Returns the balance factor of the node.
7. **void leftRotate(Node* node):** Performs a left rotation around the given node.
8. **void rightRotate(Node* node):** Performs a right rotation around the given node.
9. **void leftRightRotate(Node* node):** Performs a left rotation followed by a right rotation.
10. **void rightLeftRotate(Node* node):** Performs a right rotation followed by a left rotation.
11. **int getHeight(Node* node): R**eturns the height of the node.
12. **void updateHeight(Node* node):** Updates the height of the node based on its children.
13. **void printTree()**: Prints the AVL tree using an in-order traversal.

Please read the following instructions carefully:

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.

INFORMATION TECHNOLOGY UNIVERSITY

Assessment Rubric for Assignment

Performance metric	CLO	Able to complete the task over 80% (4-5)	Able to complete the task 50-80% (2-3)	Able to complete the task below 50% (0-1)	Marks
1. Realization of experiment	3	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed.	
2. Conducting experiment	2	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	4	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	4	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	2	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	2	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	3	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					