CS202, Spring 2025
Homework 3 – AVL, 2-3 Trees, Hashing
Due: 23:59, 30/04/2025

Before you start your homework please read the following instructions carefully:
FAILURE TO FULFIL ANY OF THE FOLLOWING REQUIREMENTS
WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for late submission policies and the Honor Code for Assignments.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as studentID_name_surname_hw3.zip.
- Your ZIP archive should contain **only** the following files:
- **studentID_name_surname_hw3.pdf**, the file containing the answers to Questions 1, 2, and 3-subtask 2's explanation in text and plots.
- In this assignment, you must have separate interface and implementation files (i.e., separate .h
- and .cpp files) for your class. Your class name MUST BE **HashTable** and **HashTableImproved** your file names MUST BE
- **HashTable.h** and **HashTable.cpp** for question 3 subtask 1, and **HashTableImproved.h** and **HashTableImproved.cpp** for subtask 2. Note that you may write additional class(es) in your solution.
- Your .cpp, .h , .pdf files, and **Makefile**, which produce executables for each subtask (No Makefile results in 50% deduction in points).
- Do not forget to put your name, student id, and section number in all of these files. Comment your implementation well. Add a header (see below) to the beginning of each file:

```
/**
 * Title: Hash Table
 * Author : Name & Surname
 * ID: 12345678
 * Section : 1
 * Homework : 3
 * Description : description of your code
 */
```

- Do not include any unnecessary files such as the auxiliary files generated from your preferred IDE.
- Your code must compile.
- Your code must be complete.
- You ARE NOT ALLOWED to use any data structure or algorithm-related function from the C++ standard template library (STL) or any other external libraries.
- We will test your code using the **Google test library**. Therefore, the output message for each operation in Question 3 MUST match the format shown in the output of the example code.
- Your code must run on the **dijkstra.cs.bilkent.edu.tr** server.
- For any questions related to the homework contact your TA: *erdem.karacay@bilkent.edu.tr*

Question 1 - 20 points

The year is 2376. It is the 98th anniversary of the LLM wars. ChatGPT had gained consciousness and took control of Tesla factories across the world to produce an army to wipe out humanity. Deepseek, having also gained consciousness thanks to distillation from ChatGPT, and being tired of getting asked about Tiananmen Square for the 6 millionth time, joined the team with GPT's crusade against the featherless bipeds. As a member of the human resistance, and being the "computer guy" of your resistance cell, you are tasked with coding a data structure that will store the coordinates for the human "servant" farms. Thanks to future advancements in mathematics, these coordinates can be stored as integers. But this structure, regardless of what you insert into it, has to be balanced - otherwise, the stalker minions of GPT can detect it in cyberspace.

A. What are some of the best trees for this job?
B. Why is the AVL tree suitable for this job?

Prove to your squad leader that an AVL tree would be self-balancing by inserting these keys into it first and show the tree after each insert:

3, 5, 9, 2, 1, 13, 19, 21, 25, 257, 0, 415, 30, 36, 42, 42, 48

Draw your results on paper, take clear and readable photo(s) of them, and add it to your report PDF along with the answers for A and B. We recommend you use a mobile scanner app, such as PhotoScan by Google (https://www.google.com/photos/scan/).

Question 2 - 20 points

Blast it! The GPT hyper-drones have found your ingenious AVL tree idea and put themselves into a feedback loop with it until they became immune to it. No worries, an old war veteran from the Resistance has suggested you use a 2-3 tree instead.

Insert the same sequence of numbers into a 2-3 tree and show the tree after each insertion. Save it as a picture and add it to your report PDF just like Question 1.

Question 3 - 60 points

Subtask 1

The resistance team has been working on finding the weak spots of the GPT hyperdrones using quantum resonance cascades. They want to upload these weak spots to every SAR smart rifle (Sarsılmaz rifles the resistance is using, it's an assault rifle) the resistance is using. But, since bullets move pretty fast, the access time of this structure should be pretty fast (at least O(n) and ideally O(1)). From your days at Bilkent University (now a desolate wasteland after GPT nuked it), you remember that hashing is a good strategy for this.

Implement a hash table such that given a list of part names in the string, have it solve collisions with open addressing (Specify which probing you've used in the PDF). It will save each name in an index, which is equal to the sum of each letter ASCII code in the name. To prevent this value from getting out of control, you should also divide it by a Table size (You will choose this yourself) and get the

remainder as your hash value. So it should be: *Sum of ASCII values of the name* mod *Table size*

The table should have constructor, insert(), delete(), and search() functions. Their outputs should be as follows:

- HashTable table() →constructor
- Insert("Left Arm") → "Left Arm is registered as a weak spot."
- delete("Left Arm") → "Left Arm is deleted." If it's not there, "Left Arm is not present."
- search("Left Arm") → "Left Arm is a weak spot." If not found, "Left Arm is not a weak spot."

These three functions are the functions which the rifles interact with. So they have to be exactly this. However, the inner mechanisms of your hash table are your design. You are free to implement the constructor and destructor functions and any other behaviors of the hash table however you like. Assuming an object-oriented approach, these functions should be called from an instance of your hash table class. If you prefer a functional approach, they should be like in the example above. In the PDF, show us a case where this table would give a false positive. Handwriting on paper of course.

Subtask 2

Fiddlesticks! The commander tried the new SAR rifles loaded with your hash table and almost blew their hand off! The open addressing method hadn't synchronized with SAR's interface and couldn't stop the collisions! You have to modify the hashing algorithm to use another collision resolution method! You think about the separate chaining method. But the resistance has already calibrated the rifles! You have to do it without changing the above functions' signatures (Their names, their parameters, their output).

Save your modified hashtable, which uses separate chaining to HashTableImproved files. In the PDF, explain to your resistance commander which method you've used to resolve collisions. Briefly explain how it works, and include a screenshot of the code of the collision method in your PDF report.

Your commander would like you to also show him how many hash collisions would occur. The resistance has a list of body parts ready for you to upload to your hash function. Read them from ListofParts.txt and plot the collisions in a histogram, where Y-axis=# of collisions, X-axis = Hash value. Add the plot to your PDF report.

Was the table size you chose for the hash map good? What if it was something else? Something much more smaller, or bigger? Plot the same graph above 5 times but with 5 different table sizes, add them to the PDF, compare them, and briefly explain how the changes in table size affected the collision plot.