Task:

This assignment has two parts:

1. Implementing a binary search of an array
2. The construction of a self-balancing binary search tree and using it to spell check a standard document.

Part 1: Binary Searching

Create a console application to conduct searching activities on a simple array, using the binary search algorithm.

Requirements:

* The program must create a static sorted array containing the numbers 1 to 10000 (10000 entries)
* The version of binary search used must also be non-recursive.
* Capture and display the start and end times for searches of each array. Search for each number (10000 searches).

Part 2: Auto-balancing Binary Search Trees

You are tasked to build a console-based spell checker that will be used to show the spelling mistakes in a standard text document.

Requirements:

* The application will start by reading all the words in a dictionary file that has been provided for you. The file is not a complete dictionary. It simply contains all the correct spelling for specific words. The dictionary words will be the basis for a custom binary search tree in your application. The dictionary is NOT sorted, and will require that your code fully support all kinds of insertions.
* By programmatic means you must balance the binary search tree of words and **display** it for review. It will be up to you to choose and implement how the tree will be balanced. You will need to completely explain your balancing algorithm for full marks. Hint: AVL trees are easier to code.
  + Once the binary search tree has been filled and balanced, you will read in a second file that is a sample document needing spell checking. All the words in the document needing checking are and now in the binary search tree. You will then compare each word in the document against the binary search tree. If the search fails, you will display the misspelled word to the console.

**Hint**: You cannot modify either the provided dictionary file or document file. However, you may find it easier to test your application with a smaller dictionary and/or document before testing against the given files. The following resource is provided as additional reading material and is used under the fair usage clause for copyrighted materials: “RL Kruse – Data Structures Text – Chapter 10.pdf”. It is available in the Resources folder for the course.

**Notes**:

Remember, you may **not** use the STL for the tree or related algorithms. You may use it for file I/O, screen I/O, etc.

Evaluation:

This assignment is worth 44 marks. Please see the marking rubric below.

Assignment Notes:

The assignment must be demonstrated to the instructor on or before the due date during class.

If your assignment is late please send an e-mail to the instructor, hal.o’connell@nscc.ca, to confirm submission. This e-mail will constitute the timestamp for evaluating any late penalty the assignment may incur.

See the **Marking Rubric** below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Criteria | Developing | Acceptable | Good | Exceptional | Marks |
| 0 | 1 | 2 | 3 |
| Binary Search | * No Binary Search attempted | * Binary search algorithm does not find each key value searched for | * Binary Search algorithm has minor errors or times not captured and displayed | * Binary Search Algorithm finds all keys, * Timings captured and displayed * No errors | **\_\_\_\_x2** |
| Auto Balancing Binary Search Tree | * BST isn’t used to store dictionary - no attempt to balance tree was made - too many errors exist | * Dictionary was correctly read into BST - BST can be viewed on the console * An attempt to balance tree was made and can be explained * Some errors exist | * Dictionary BST was read and displayed correctly - BST has almost all nodes balanced * Few errors exist | * BST contains all dictionary words and is perfectly balanced * Balance algorithm can be explained - can easily view tree on the console | **\_\_\_\_x5\_** |
| Spell Checker Output | * The program does not output misspelled words from the test document | * Some misspelled words are correctly identified and listed * Could not handle punctuation, capitalization or special characters * Some errors exist | * Most misspelled words are correctly identified and listed * Could not handle rare cases * Few errors exist | * All misspelled words in the text document are correctly identified and listed * No errors exist | **\_\_\_\_** |
|  |  |  |  | **Sub Total** | **\_24\_\_** |

| Aesthetics | * incorrect or non-existent use of whitespace in output * output is confusing and hard to follow | * fair use of   whitespace   * most output is clear, but poorly presented | * excellent use of whitespace * output is clear and attractively presented | |  | | \_\_\_\_\_ |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Readability | * source code is poorly organized and very difficult to read | * source code is fairly easy to read, but is hard to follow in some areas | * source code is exceptionally well organized and easy to follow | |  | | \_\_\_\_\_ |
| Reusability | * source code cannot be reused * no functions or classes used | * portions of code could be reused with modifications | * source code could be easily reused with few modifications | |  | | \_\_\_\_\_ |
| Efficiency | * contains large portions that could have been easily reduced using a different method * too much code is replicated, copy /pasted | * tried some methods to improve efficiency * can explain what they attempted | * very clean and efficient code * can propose new ideas for improvement | |  | | \_\_\_\_\_ |
| Comments | * little to no comments used | * comments are used, some are meaningful and easily understood * some files and functions have headers | * not over/under commented * comments are meaningful and easily understood * files and functions have headers * Code is self-documenting | |  | | \_\_\_\_\_ |
| **Naming** Convention | * no standard naming convention followed | * a standard naming convention was used for part of the program, but deviated often | * industry standard naming convention used throughout the program | |  | | \_\_\_\_\_ |
| Consistency | * no consistency in formatting or layout of source code | * source code formatting was present but inconsistent with whitespace, brackets, etc | * source code formatting never deviated from the programmer’s layout | |  | | \_\_\_\_\_ |
| SubTotal | | **14** | |
| Assignment Total | | **38** | |

0 - Assignment not submitted or work not original.