Dictionary and Spell Checker

Terence Munro

Contents

1. Problem Statement

The goal of this project is to create a dictionary from an input file and using a height balanced Binary Search Tree (AVL tree) sort it alphabetically. The dictionary file must be saved to disk and also added to a HashTable for use in a spelling checker.

2. User Requirements

The following outlines the user requirements for the program:

The user should be able to:

- Read in a text file to create dictionary
- Print out the dictionary in alphabetical order
- Save the dictionary to file
- Check another text file’s spelling using the dictionary created

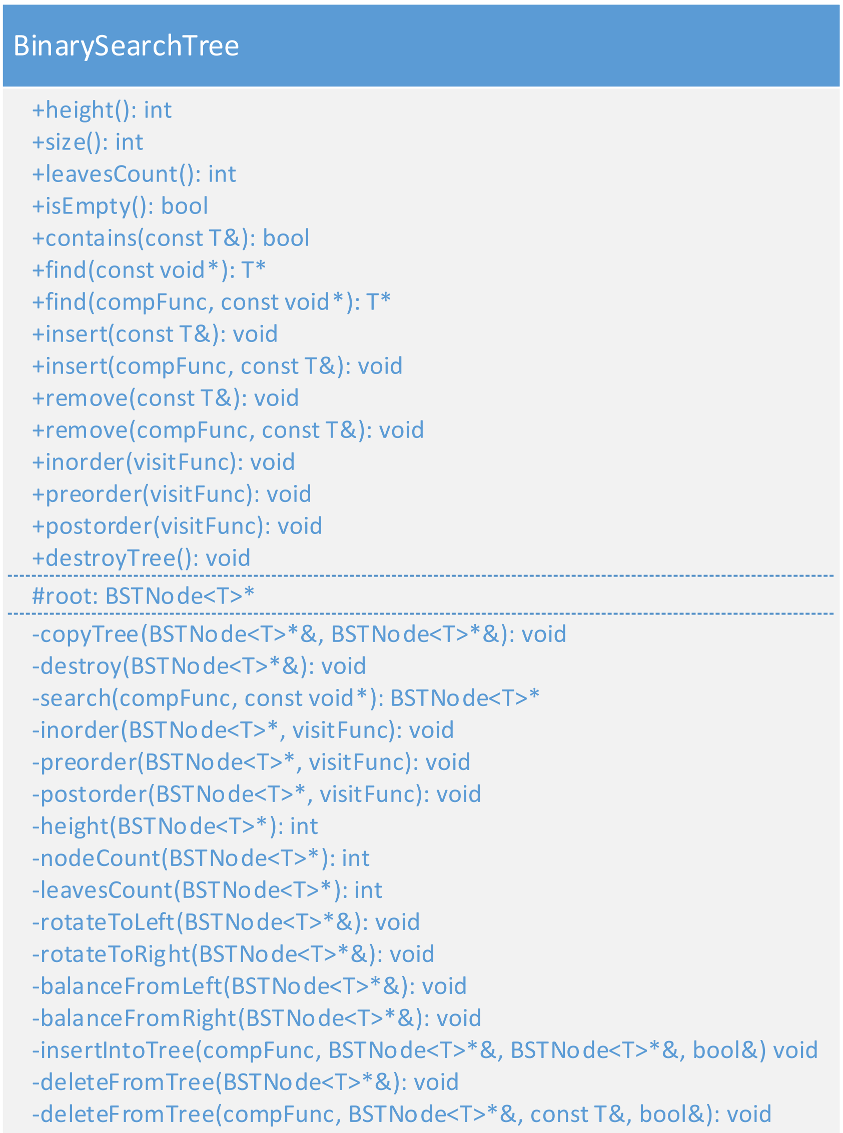
3. Software Requirements

The software requirements are:

- File must be read in using C library functions and FILE pointers
- C++ strings must be used when reading in the file and creating the AVL tree
- The string must be split into individual words
- Insertion into the tree must keep it reasonally well balanced
- All words must be converted to lower case
- Words are only allowed to contain letters, hyphens and apostrophies. No numbers.
- Inorder traversal of the AVL should produce an alphabetical listing of all the words.
- The alphabetical listing must then be saved to file using C library functions and separated by newlines
- The file must be read again using C++ stream functions and C strings and added to a hash table
- A list of similar words according to a hash function must be retrievable

4. Software Design

UML Diagrams

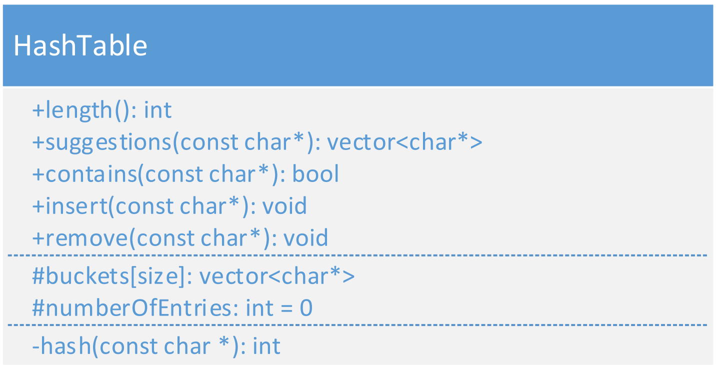


BinarySearchTree

+height(): int
+size(): int
+leavesCount(): int
+isEmpty(): bool
+contains(const T&): bool
+find(const void*): T*
+find(compFunc, const void*): T*
+insert(const T&): void
+insert(compFunc, const T&): void
+remove(const T&): void
+remove(compFunc, const T&): void
+inorder(visitFunc): void
+preorder(visitFunc): void
+postorder(visitFunc): void
+destroyTree(): void

#root: BSTNode<T>*

-copyTree(BSTNode<T>*&, BSTNode<T>*&): void
-destroy(BSTNode<T>*&): void
-search(compFunc, const void*): BSTNode<T>*
-inorder(BSTNode<T>*, visitFunc): void
-preorder(BSTNode<T>*, visitFunc): void
-postorder(BSTNode<T>*, visitFunc): void
-height(BSTNode<T>*): int
-nodeCount(BSTNode<T>*): int
-leavesCount(BSTNode<T>*): int
-rotateToLeft(BSTNode<T>*&): void
-rotateToRight(BSTNode<T>*&): void
-balanceFromLeft(BSTNode<T>*&): void
-balanceFromRight(BSTNode<T>*&): void
-insertIntoTree(compFunc, BSTNode<T>*&, BSTNode<T>*&, bool&) void
-deleteFromTree(BSTNode<T>*&): void
-deleteFromTree(compFunc, BSTNode<T>*&, const T&, bool&): void



HashTable

+length(): int
+suggestions(const char*): vector<char*>
+contains(const char*): bool
+insert(const char*): void
+remove(const char*): void

#buckets[size]: vector<char*>
#numberOfEntries: int = 0

-hash(const char *): int

Data structures in the software

BinarySearchTree

- **Name:** bst
- **Type:** Graph
- **Purpose:** To store the words of the dictionary allowing an inorder traversal to produce an alphabetically sorted list of words

Array of Vectors

- **Name:** buckets
- **Type:** Array of Lists
- **Purpose:** Underlying datastructure of the hash table, the array represents the buckets and the vector is for storing collisions

5. Requirement Acceptance Tests

Software Requirement No	Test	Implemented (Full/Partial/None)	Test Results (Pass/Fail)	Comments (for partial implementation or failed test results)
1	Reads in file ‘Ass2 Dictionary.txt’ using C library functions fopen, fread, fclose and FILE pointers	Full	Pass	
2	Correctly processed words are in C++ string format and added to Binary Search Tree	Full	Pass	
3	Statistics of the Binary Search Tree are provided	Full	Pass	
4	Alphabetical listing can be produced	Full	Pass	
5	Dictionary is saved to file ‘dict.txt’ using C library functions	Full	Pass	
6	Dictionary is read in using C++ streams	Full	Pass	
7	Words are split into C strings and added to Hash Table	Full	Pass	
8	Dictionary is used to check another text file for spelling errors and some suggestions are offered	Full	Pass	

6.

7. Detailed Software Testing

Unit tests are only available in the 2013 project as the 2008 edition of Visual Studio uses a different framework and it would have been to much extra time to convert the syntax

No	Test	Expected Results	Actual Results
1.0	**User testing**		**Figures 1-4**
1.1	- Read in text file and add to binary tree	~500 nodes	453 words (Figure 1)
1.2	- Produce alphabetical listing by traversing tree inorder	Alphabetical listing of words	As expected (Figure 2, 3)
1.3	- Save dictionary to file	File is created with dictionary of words in alphabetical order separated by new lines	As expected
1.4	- Read back in and added to hash table	Hash table contains dictionary and does not have too many collisions	Collisions are more frequent than hoped
1.5	- Reasonable spelling suggestions given	Inputted incorrect word returns similar words	Not very close (Figure 3, 4)
2.0	**Unit Tests (37)**	**Expected results in unit tests**	**All tests passed (Figure 5)**

Example unit test:

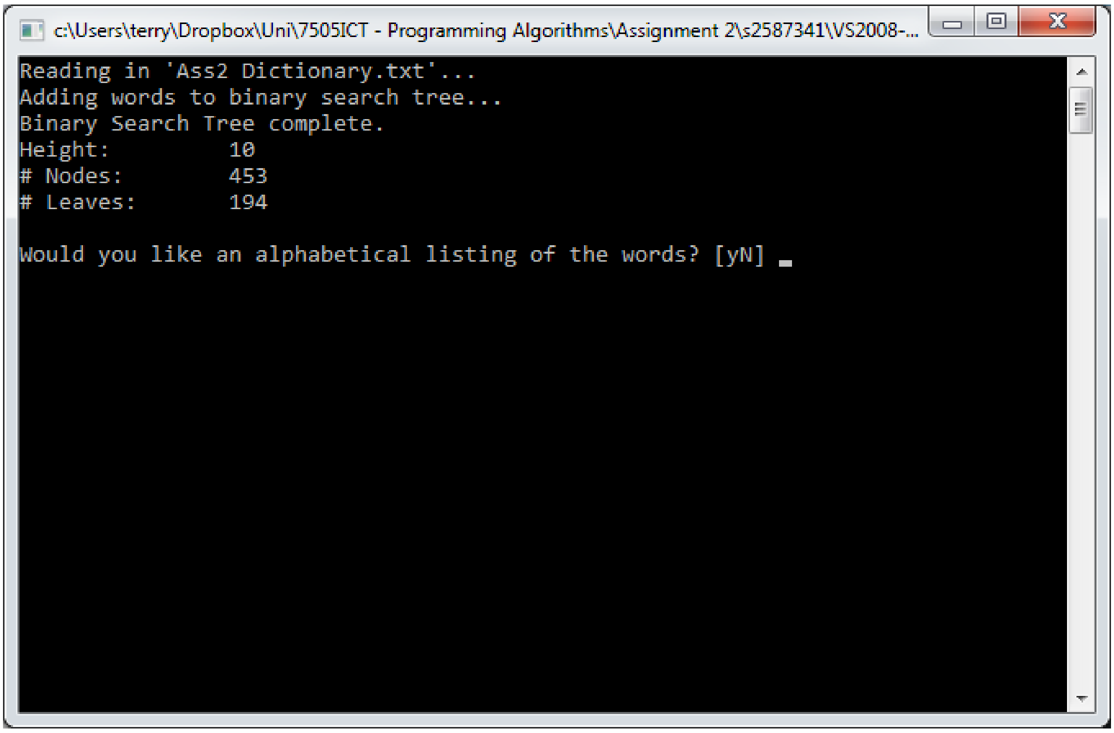
```
TEST_METHOD(Queue_PushBackAndPopFront)
{
      Queue<int> q;

      q.push_back(7);
      q.push_back(9);
      q.push_back(11);

      Assert::AreEqual<int>(3, q.length());

      int ret = q.pop_front();

      Assert::AreEqual<int>(7, ret);
      Assert::AreEqual<int>(2, q.length());
}
```

The entire suite of unit tests can be found in VS2013-UnitTests/7501ICT_s2587341_Assignment2/UnitTests.

8. User Instructions

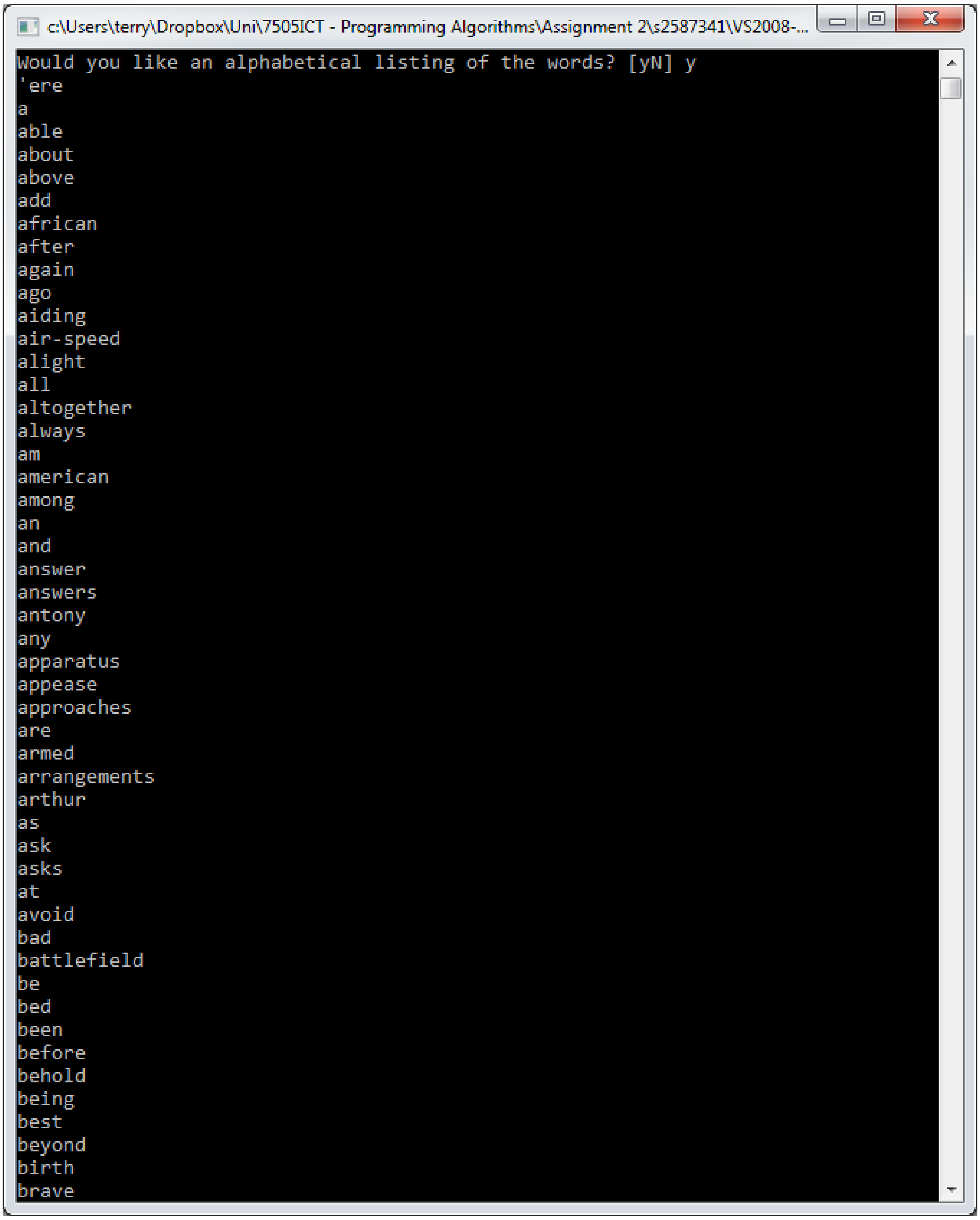
- User can load up the project using the Visual Studio 2008 solution file in the same directory as this Documentation.
- Statically compiled executable is available in the Release directory
- If the user wanted run user tests themselves they need to use Visual Studio 2013 and load up the VS2013 solution in a directory external to the question 1 and 2 directory labelled VS2013-UnitTests

9. Appendix


c:\Users\terry\Dropbox\Uni\7505ICT - Programming Algorithms\Assignment 2\s2587341\VS2008-...

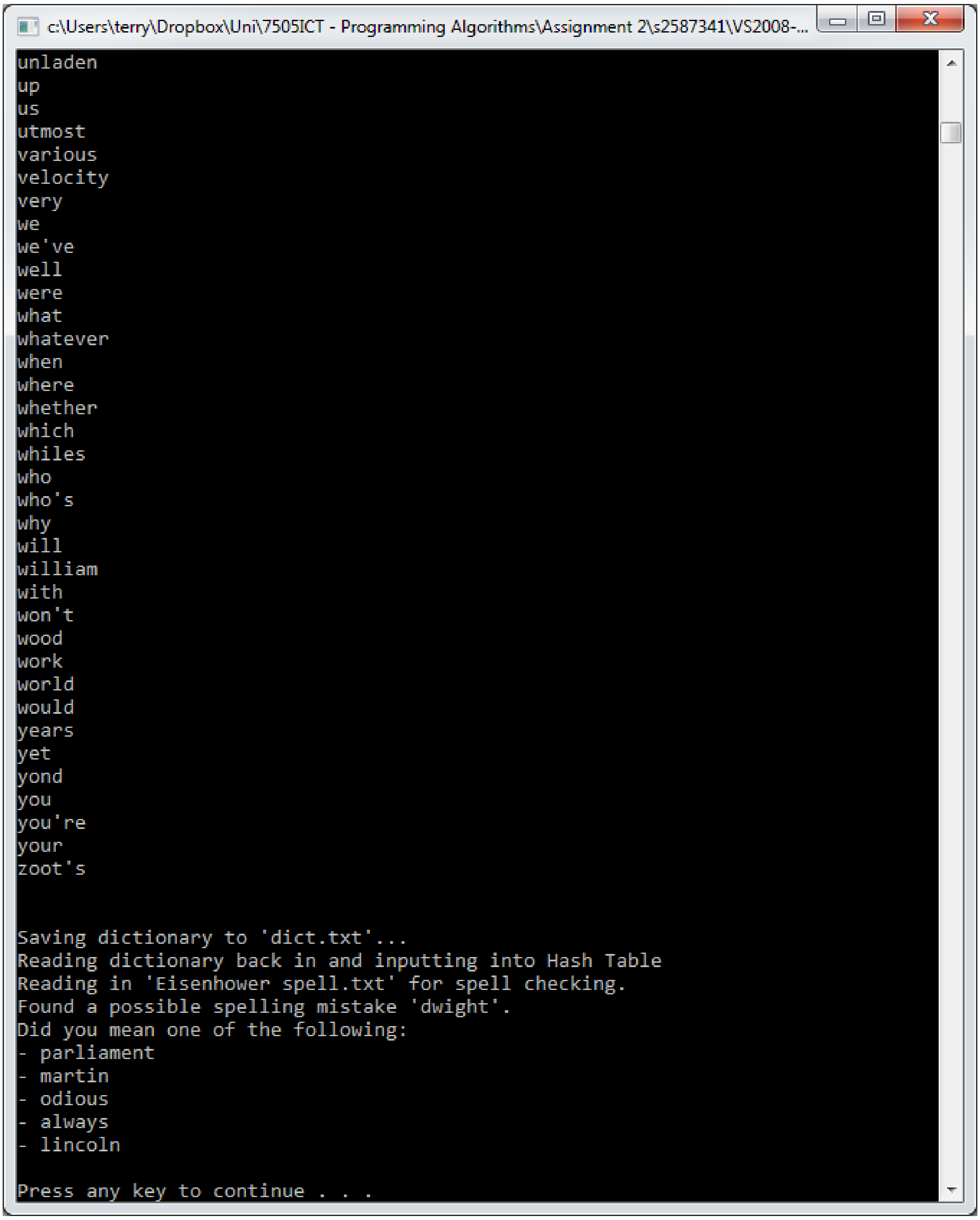
```
Reading in 'Ass2 Dictionary.txt'...
Adding words to binary search tree...
Binary Search Tree complete.
Height:         10
# Nodes:        453
# Leaves:       194

Would you like an alphabetical listing of the words? [yN] _
```

Figure 1


c:\Users\terry\Dropbox\Uni\7505ICT - Programming Algorithms\Assignment 2\s2587341\VS2008-...

```
Would you like an alphabetical listing of the words? [yN] y
'ere
a
able
about
above
add
african
after
again
ago
aiding
air-speed
alight
all
altogether
always
am
american
among
an
and
answer
answers
antony
any
apparatus
appease
approaches
are
armed
arrangements
arthur
as
ask
asks
at
avoid
bad
battlefield
be
bed
been
before
behold
being
best
beyond
birth
brave
```

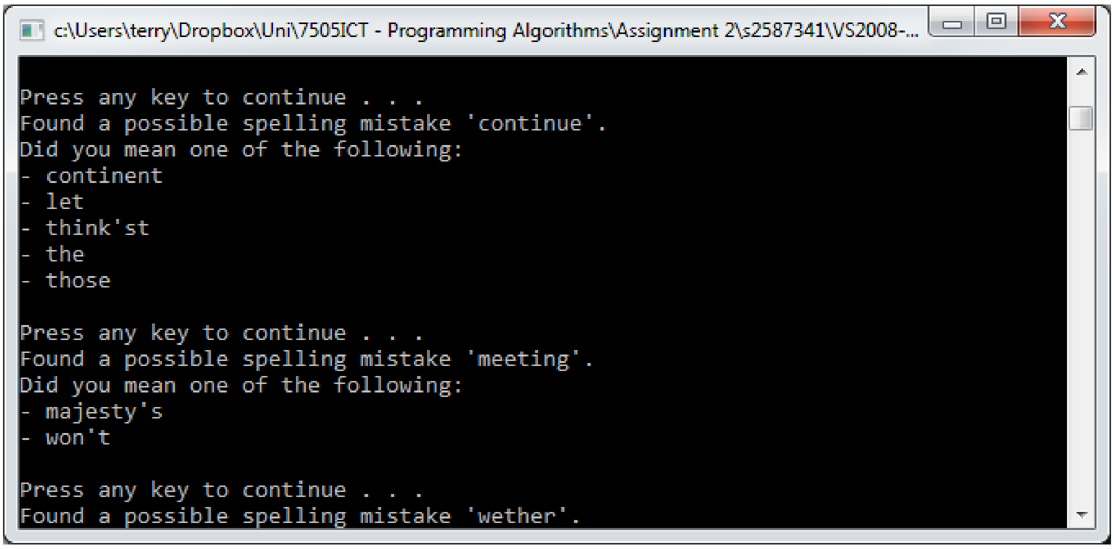
Figure 2



c:\Users\terry\Dropbox\Uni\7505ICT - Programming Algorithms\Assignment 2\s2587341\VS2008-...

```
unladen
up
us
utmost
various
velocity
very
we
we've
well
were
what
whatever
when
where
whether
which
whiles
who
who's
why
will
william
with
won't
wood
work
world
would
years
yet
yond
you
you're
your
zoot's


Saving dictionary to 'dict.txt'...
Reading dictionary back in and inputting into Hash Table
Reading in 'Eisenhower spell.txt' for spell checking.
Found a possible spelling mistake 'dwight'.
Did you mean one of the following:
- parliament
- martin
- odious
- always
- lincoln

Press any key to continue . . .
```

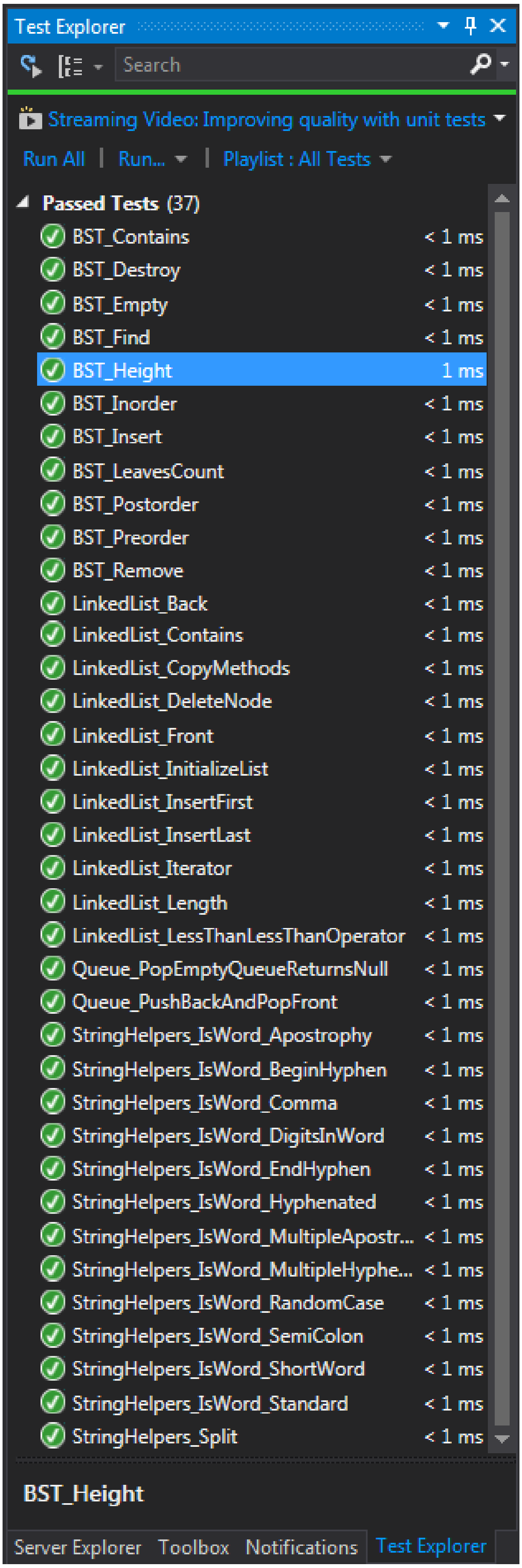
Figure 3



c:\Users\terry\Dropbox\Uni\7505ICT - Programming Algorithms\Assignment 2\s2587341\VS2008-...

```
Press any key to continue . . .
Found a possible spelling mistake 'continue'.
Did you mean one of the following:
- continent
- let
- think'st
- the
- those

Press any key to continue . . .
Found a possible spelling mistake 'meeting'.
Did you mean one of the following:
- majesty's
- won't

Press any key to continue . . .
Found a possible spelling mistake 'wether'.
```

Figure 4



```
Passed Tests (37)
  BST_Contains                                 < 1 ms
  BST_Destroy                                  < 1 ms
  BST_Empty                                    < 1 ms
  BST_Find                                     < 1 ms
  BST_Height                                     1 ms
  BST_Inorder                                  < 1 ms
  BST_Insert                                   < 1 ms
  BST_LeavesCount                              < 1 ms
  BST_Postorder                                < 1 ms
  BST_Preorder                                 < 1 ms
  BST_Remove                                   < 1 ms
  LinkedList_Back                              < 1 ms
  LinkedList_Contains                          < 1 ms
  LinkedList_CopyMethods                       < 1 ms
  LinkedList_DeleteNode                        < 1 ms
  LinkedList_Front                             < 1 ms
  LinkedList_InitializeList                    < 1 ms
  LinkedList_InsertFirst                       < 1 ms
  LinkedList_InsertLast                        < 1 ms
  LinkedList_Iterator                          < 1 ms
  LinkedList_Length                            < 1 ms
  LinkedList_LessThanLessThanOperator          < 1 ms
  Queue_PopEmptyQueueReturnsNull               < 1 ms
  Queue_PushBackAndPopFront                    < 1 ms
  StringHelpers_IsWord_Apostrophy              < 1 ms
  StringHelpers_IsWord_BeginHyphen             < 1 ms
  StringHelpers_IsWord_Comma                   < 1 ms
  StringHelpers_IsWord_DigitsInWord            < 1 ms
  StringHelpers_IsWord_EndHyphen               < 1 ms
  StringHelpers_IsWord_Hyphenated              < 1 ms
  StringHelpers_IsWord_MultipleApostr...       < 1 ms
  StringHelpers_IsWord_MultipleHyphe...        < 1 ms
  StringHelpers_IsWord_RandomCase              < 1 ms
  StringHelpers_IsWord_SemiColon               < 1 ms
  StringHelpers_IsWord_ShortWord               < 1 ms
  StringHelpers_IsWord_Standard                < 1 ms
  StringHelpers_Split                          < 1 ms

BST_Height
```

Figure 5