 **Weight: 19%**

Data Structures

Project 3 – Word Find

Contents

[Objective 1](#_Toc286820541)

[Requirements 2](#_Toc286820542)

[Milestones 2](#_Toc286820543)

[Grading Breakdown 2](#_Toc286820544)

[Standard Deductions 3](#_Toc286820545)

[Submission 3](#_Toc286820546)

# Objective

In this game, the player is given a 4x4 field of random characters and 60 seconds to make as many real words as they can from adjacent letters in the playfield. In order to get a score for a word, it must be 3 letters long. The same letter position in the field cannot be used twice in the same word, and the same word cannot be scored twice regardless of how many times it actually appears in the playfield.

Let’s take a quick look at a sample playfield and some words that are found through connecting adjacent characters:

air- slow-

a r l w a r l w

i t o s i t o s

e n s t e n s t

w p k j w p k j

went- knew-

a r l w a r l w

i t o s i t o s

e n s t e n s t

w p k j w p k j

know- stink-

a r l w a r l w

i t o s i t o s

e n s t e n s t

w p k j w p k j

# Requirements

* You are required to use your hash table class from Lab 5 along with a hash function of your own design.
  + The hash table must have at least 500 buckets.
  + No more than 5% of the buckets should be empty.
  + The words should be as evenly distributed as possible.
* You are required to use the Binary Search Tree class from Lab 6 to store the player’s correct guesses.
  + You are required to display the player’s correct guesses to the screen and update them in real time
* You are required to use the graph class from Lab 7 to store the grid of characters and to search for words based on their adjacencies in the graph.
* You must present the player with an option to randomize a grid or to use the hardcoded test grid shown above. This option should be presented to the player at the beginning of every round.
* You must have at least 4 random vowels in the grid, and they must be in random locations.
* The score for a correct word should be the factorial of the length of the word.
* The user has 60 seconds to guess as many words as possible.
* You need a high score table saved to a file in the format of your choosing.
  + The high score table should contain the score and (at least) a 3 letter string to identify the player who achieved the high score.
  + The high score list should be sorted and displayed with the highest score at th top and the lowest at the bottom
* When the time is up, the user should be presented with a replay option.
  + You should clear out the BST and reset the score for the next game.
* You are required to display a timer on screen that counts down in real time.
* You are required to get the player’s input in real time
  + Pressing alphabetical keys (a-z) should add the corresponding character to the string.
  + Pressing the backspace key should remove the last character from the string.
  + Pressing the enter key should check the string to see if it scores. This should also clear the string.
  + Pressing the escape key should end the round.
  + Any other key should do NOTHING.
* The same letter position in the field cannot be used twice in the same word.
* The same word cannot be scored twice regardless of how many times it actually appears in the playfield.

# Loading the Dictionary

Add the following function to your hash table class. it will help you gather some data about your distribution rate.

\*you may need to change the names of some of my variables to match your class member's names.

// the following variable names are used below, match them to your variable names

// theTable - this is the array of lists

// buck - the number of buckets

template <typename Type>

void HTable<Type>::printSomeStuff(const char\* filePath = "hashdata.txt")

{

ofstream outFile(filePath);

if(outFile.is\_open())

{

unsigned int empty = 0;

unsigned int totalCount = 0;

unsigned int loIndex = 0;

unsigned int hiIndex = 0;

for(unsigned int i = 0; i < buck; ++i)

{

totalCount += theTable[i].size();

outFile << i << " : " << theTable[i].size() << '\n';

if(theTable[i].size() == 0)

++empty;

if(theTable[i].size() < theTable[loIndex].size())

loIndex = i;

else if(theTable[i].size() > theTable[hiIndex].size())

hiIndex = i;

}

outFile << '\n' << totalCount << " Total items stored in " << buck << " buckets\n";

outFile << '\n' <<empty << " Buckets are empty\n\n";

unsigned int Low = theTable[loIndex].size();

unsigned int range = theTable[hiIndex].size() - Low + 1;

outFile << "each bucket contains between " << Low << " and " << Low+range-1 << " items.\n\n";

unsigned int\* arr = new unsigned int[range];

for(unsigned int j = 0; j < range; ++j)

arr[j] = 0;

for(unsigned int k = 0; k < buck; ++k)

++arr[theTable[k].size() - Low];

for(unsigned int p = 0; p < range; ++p)

outFile << arr[p] << " buckets have " << p+Low << " items\n";

delete[] arr;

}

}

# Milestones

The following is a series of milestones and the day on which they should be completed:

Day 6: Load dictionary from engldict.txt into a Hash Table

Day 7: Display timer on screen in real-time.

Accept and display player’s input in real-time.

Store user’s correct guesses in Binary Search Tree (for now, just check if the word is in the

dictionary) and display them to the screen in alphabetical order.

Store and display the player’s score.

Day 8: Display option to choose whether to randomize or hard code a grid of characters.

Store the grid of characters in a Graph and display it to the screen.

Day 9: Check player’s input to ensure it is made from adjacencies in the graph.

Day 10: Spit & Polish

# Grading Breakdown

Requirement Point value

|  |  |
| --- | --- |
| Hash table and function | 25 |
| Score | 10 |
| Player’s correct guesses displayed to screen | 10 |
| Hardcoded and randomized grids | 10 |
| Graph adjacency check | 30 |
| High Score table saved to file | 15 |

# Standard Deductions

|  |  |
| --- | --- |
| Compiler errors | 100 |
| Crash | 100 |
| Memory Leak | 50 each |
| Header protection | 20 each |
| Compiler warnings | 10 each |

# Submission

Due Date : 23:59:59 PM on Day 11

To submit the Project 3 assignment:

1. Clean and build the project in Visual Studio. ensure there are no errors or warnings.

2. Run the project in debug mode with leak detection turned on to ensure that all behavior is correct and there are no memory leaks.

3. Close visual studio.

4. Navigate to the Project 3 folder (The folder that contains your Project 3 Visual Studio solution and project).

5. Delete all files and folders - except for the following:

* Your .sln file (this is the Visual Studio solution file)
* Your .vcxproj file (this is the Visual Studio project file)
* Your .cpp and .h files (these contain the code that you wrote)
* The .engldicy.txt file that you were given for test purposes.

**\*\* Any unnecessary files or folders you submit will cost you 10 points per file or folder\*\*  
\*\*Make sure you have your Windows environment set to show you hidden files and folders, or you may miss some of these\*\***

6. On your desktop, create a new folder with your name in the following format: "Last Name, First Name" - nothing else.

     Your Last Name - a comma - a single space - your First Name. Appropriate capitalization for proper names should be used.

     Examples : "Pollack, Joey"; "De La Paz, Christhian"; "Tjarks, Matthew".

7. Copy your Project 3 folder into the folder that you just created.

8. Compress the folder by right clicking on the folder with your mouse and selecting 'Send To->Compressed (Zipped) Folder'.

9. Submit the Compressed Folder.