|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| ZAT Dictionary  DSA Project |
| |  |  |  | | --- | --- | --- | | Zainab Eman 22f-3738 | Abdullah 22f-3722 | M.Taqi 22f-3708 | |

Contents

[ZAT Dictionary 1](#_Toc152800238)

[Introduction 1](#_Toc152800239)

[Overview: 1](#_Toc152800240)

[Project Structure 1](#_Toc152800241)

[1. Node Class 1](#_Toc152800242)

[2. Trie Class 1](#_Toc152800243)

[3. Vector Class 1](#_Toc152800244)

[Initialization and Data Loading 2](#_Toc152800245)

[Main Interface 2](#_Toc152800246)

[Main Menu 2](#_Toc152800247)

[Main Features 2](#_Toc152800248)

[1. GUI 3](#_Toc152800249)

[2. Exception Handling 3](#_Toc152800250)

[Working 3](#_Toc152800251)

[Search function and suggestions: 3](#_Toc152800252)

[Update function: 4](#_Toc152800253)

[Delete function: 6](#_Toc152800254)

[Add Word function: 7](#_Toc152800255)

[Exit function: 9](#_Toc152800256)

[Conclusion 10](#_Toc152800257)

# **ZAT Dictionary**

# Introduction

In the realm of computer science, dictionaries play a crucial role in storing and managing data, offering fast and reliable access to information. Our C++ dictionary employs the trie data structure, a tree-like data structure, to organize and store words or keys in a manner that allows for efficient searching, insertion, and deletion and updating operations.

# **Overview:**

## Project Structure

Our C++ dictionary is mainly structured around three key classes: node, Trie, and Vector. These classes work collaboratively to create a feature-rich dictionary that efficiently stores and retrieves key-value pairs.

### **1. Node Class**

The Node class serves as the building block for creating linked lists within the dictionary. Each node is designed to hold a character value and a pointer to the next node, facilitating the creation of dynamic data structures.

### **2. Trie Class**

The heart of our dictionary lies in the Trie class, which implements a trie data structure. In this trie, each node can have up to 26 children, and each child can further have 26 children. This hierarchical structure is particularly advantageous for storing and searching for words efficiently. The trie enables a quick and organized representation of the entire dictionary, allowing for fast retrieval of word meanings.

### **3. Vector Class**

The Vector class is used to handle dynamic arrays, providing a flexible mechanism for managing the size and storage of data within the dictionary. It contributes to the adaptability of our implementation.

## Initialization and Data Loading

At the beginning of the program, the dictionary is initialized by reading data from an external text file. This file contains a pre-existing set of words and their corresponding meanings, populating the dictionary with an initial dataset.

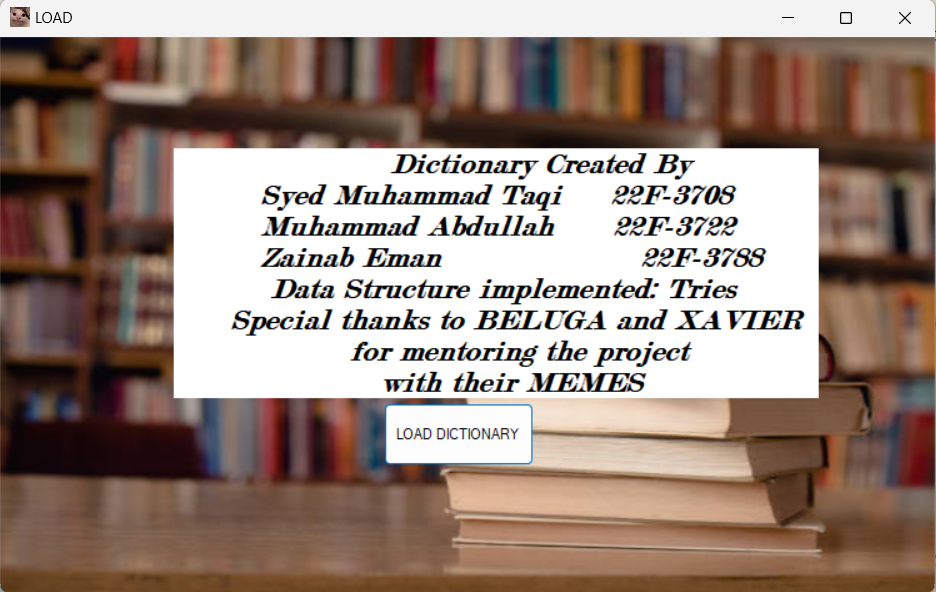
## Main Interface

After the initialization, a load dictionary page appears before user that leads to the main interface. This interface serves as the gateway for users to interact with the dictionary, enabling operations such as:

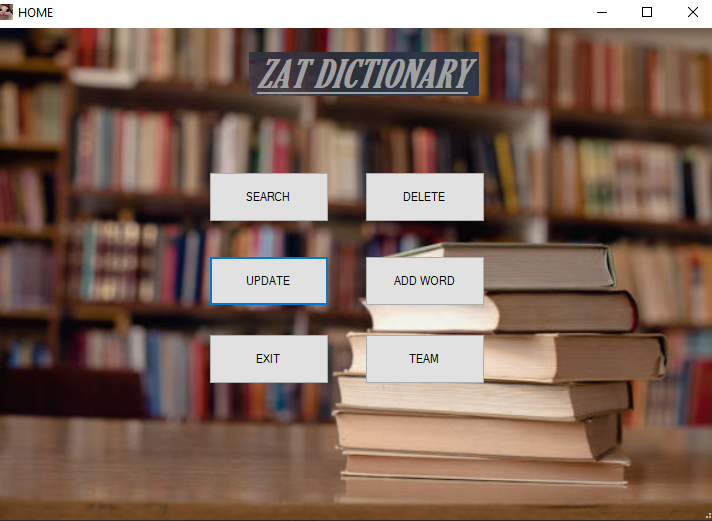
* Search
* Update
* Delete
* Add Word
* Team

The trie structure ensures that these operations are performed with optimal efficiency.

Loading Screen:



Main Menu:



## Main Features

## GUI

* **User-Friendly Interface:** The GUI is designed to be user-friendly, providing a visually intuitive experience for users of all levels.
* **Interactive Operations:** Users can easily perform operations such as searching for word meanings, adding new words, and deleting entries through a visually appealing interface.

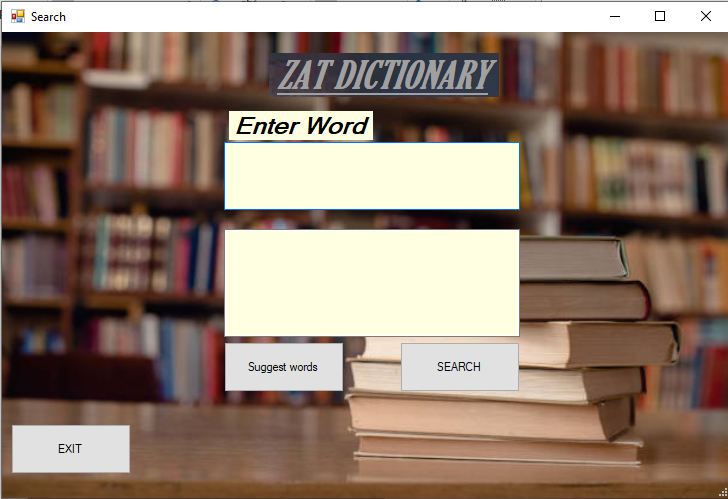
## Exception Handling

* **Precise Error Messages**: Exceptions are carefully crafted to provide users with informative and precise error messages, aiding in quick issue resolution.
* **Improved Debugging:** With explicit exception handling, debugging becomes more straightforward, allowing developers to identify and address issues efficiently.

# **Working**

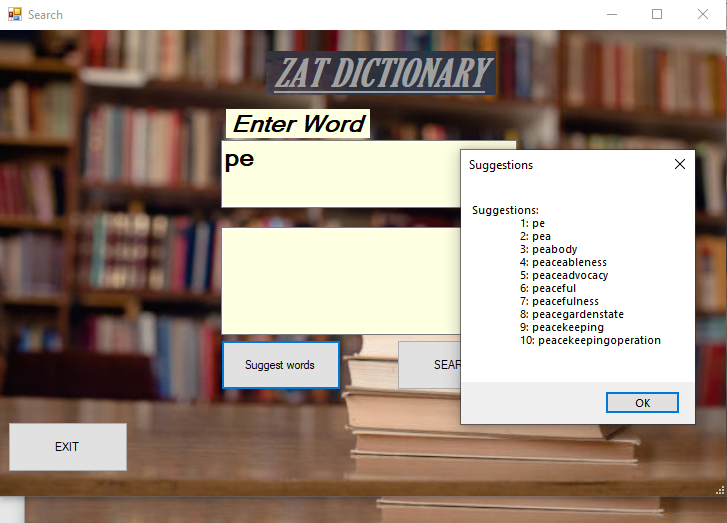
## Search function and suggestions:

**Main screen :**

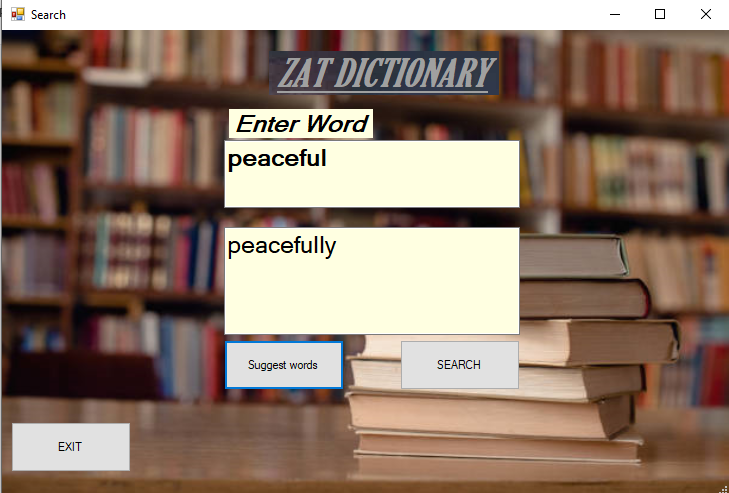


**Adding word:**

Will show top ten suggestion as shown in below picture:

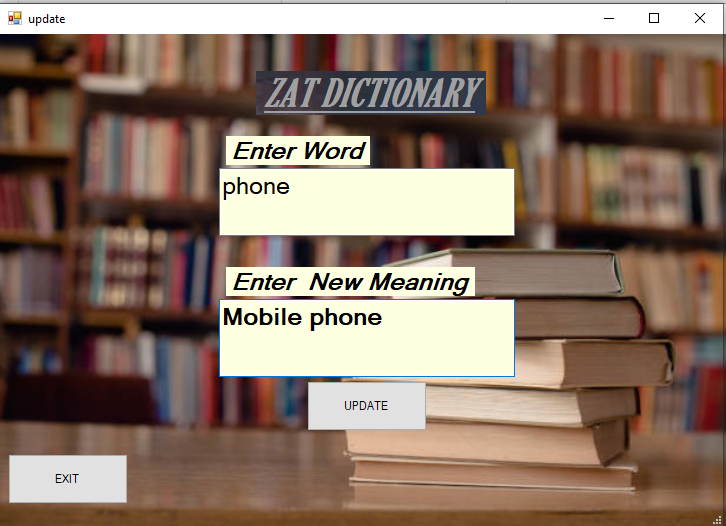


**Searching :**

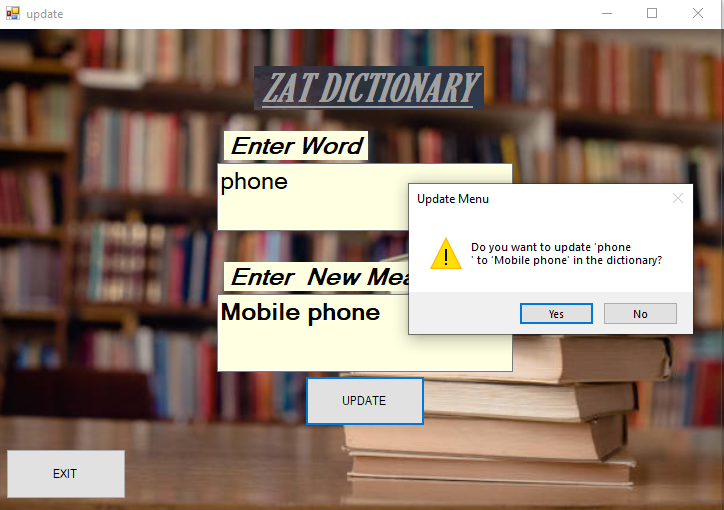
****

## Update function:

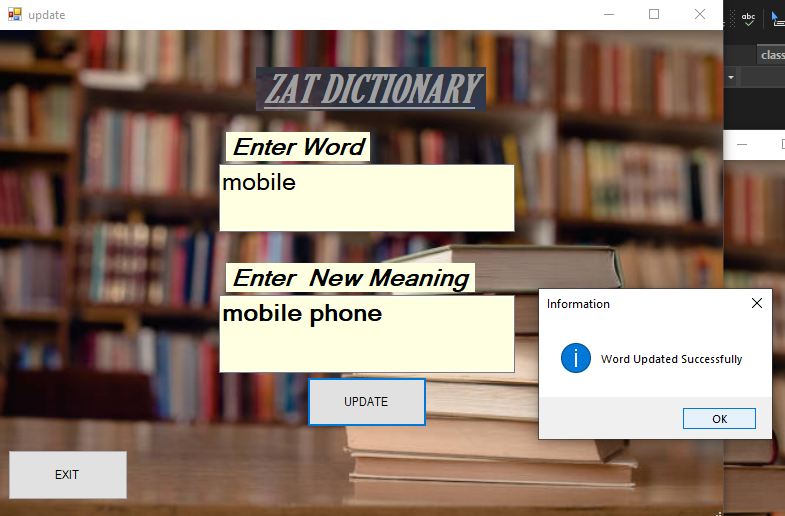
Main screen after writing a word and its meaning :



By entering update button a confirmation screen pop up appear:

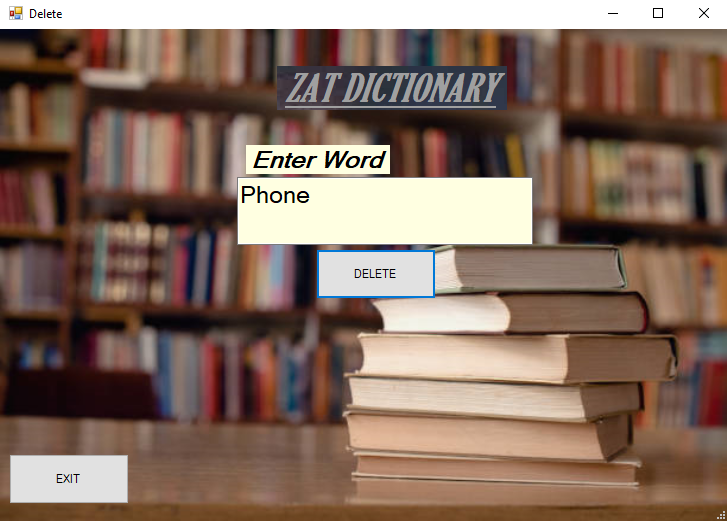


Word updated successfully:

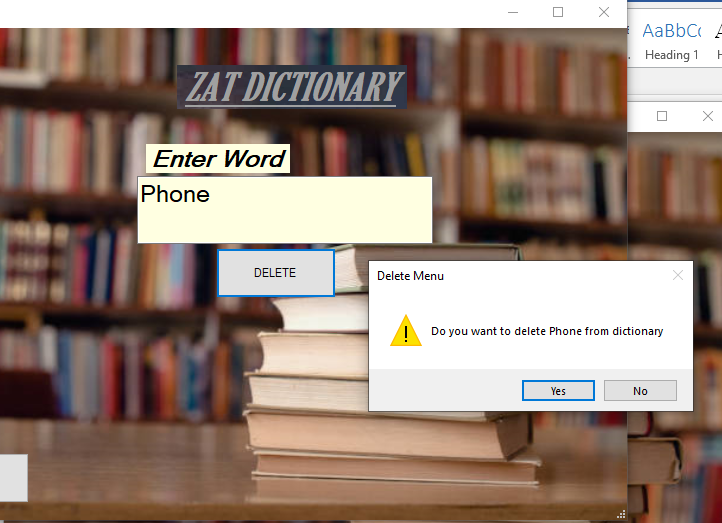


## Delete function:

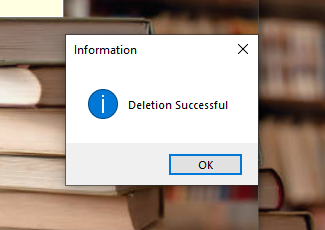
Main screen after writing a word :



By entering delete button a confirmation screen pop up appear:

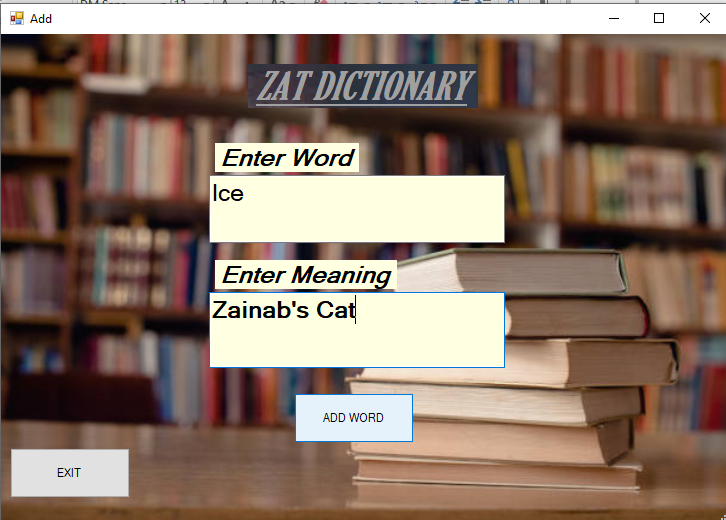


Word deleted successfully:

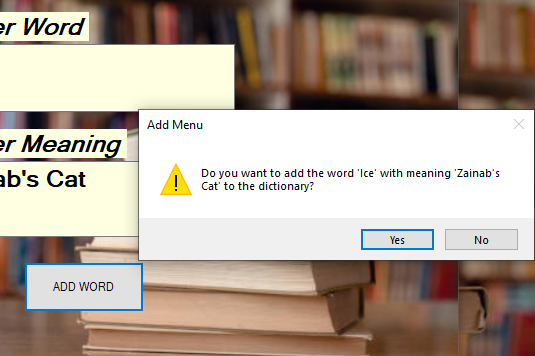


## Add Word function:

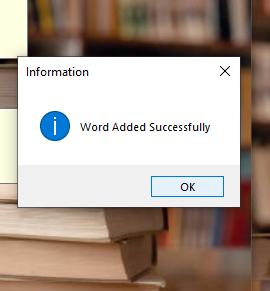
Main screen after writing a word and its meaning:



By entering delete button a confirmation screen pop up appear:



Word added successfully:



## Exit function:

On main menu, after clicking team the main menu closes and a pop screen appear as shown in the picture below:



# Conclusion

Our C++ dictionary combines the power of linked lists, tries, and dynamic arrays to create a robust and adaptable solution for managing and exploring word meanings. The trie structure, in particular, offers a powerful and efficient mechanism for organizing vast amounts of word data, making our dictionary a valuable tool for users and developers alike.