BAHRIA UNIVERSITY KARACHI CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

Bahria University
Discovering Knowledge

DATA STRUCTURES AND ALGORITHM LAB

(CSL-221)

BS CS 3B

DIGITAL DICTIONARY (USING BST)

Submitted by:

MOHAMMAD SOBAN (02-134222-010)

SAKINA YAMEEN (02-134222-048)

Submitted To:

Engr. Rabia Amjad

Submitted On:

08/01/24

Table of Contents

Acknowledgement

We would like to express our gratitude to the group members for helping in the required work. Also our sincere thanks to our teachers who helped us through whole semester by clearing all our doubts and helping us ace through the difficulties.

Abstract

The Digital Dictionary project is a machine-readable version of a standard dictionary implemented using a Binary Search Tree (BST). It aims to provide users with a convenient and efficient way to access word meanings digitally. The project includes features such as word insertion, searching, updating, deleting, and displaying. It utilizes file handling to store dictionary data in a .txt file.

Introduction

In a world increasingly reliant on digital resources, the Digital Dictionary project addresses the need for a convenient and portable solution to access word meanings. By utilizing a Binary Search Tree, the project allows users to efficiently search, update, and manage dictionary entries digitally.

Problem statement

Traditional dictionaries are often bulky and inconvenient for users to carry. The Digital Dictionary project aims to solve this problem by providing a lightweight, easily accessible digital alternative. Additionally, the project addresses the need for efficient word search and management.

Existing Systems

Comparing with existing systems, our project distinguishes itself by combining the convenience of digital access with the organizational efficiency of a Binary Search Tree. This approach enables faster search operations and efficient management of dictionary entries. It also allows us to save the data and read and access from the txt file anytime.

Objectives and goals

The primary objectives of the Digital Dictionary project include:

- Providing a user-friendly interface for word search and management.
- Implementing a Binary Search Tree for efficient data organization.
- Enabling users to insert, search, update, delete, and display words and meanings.

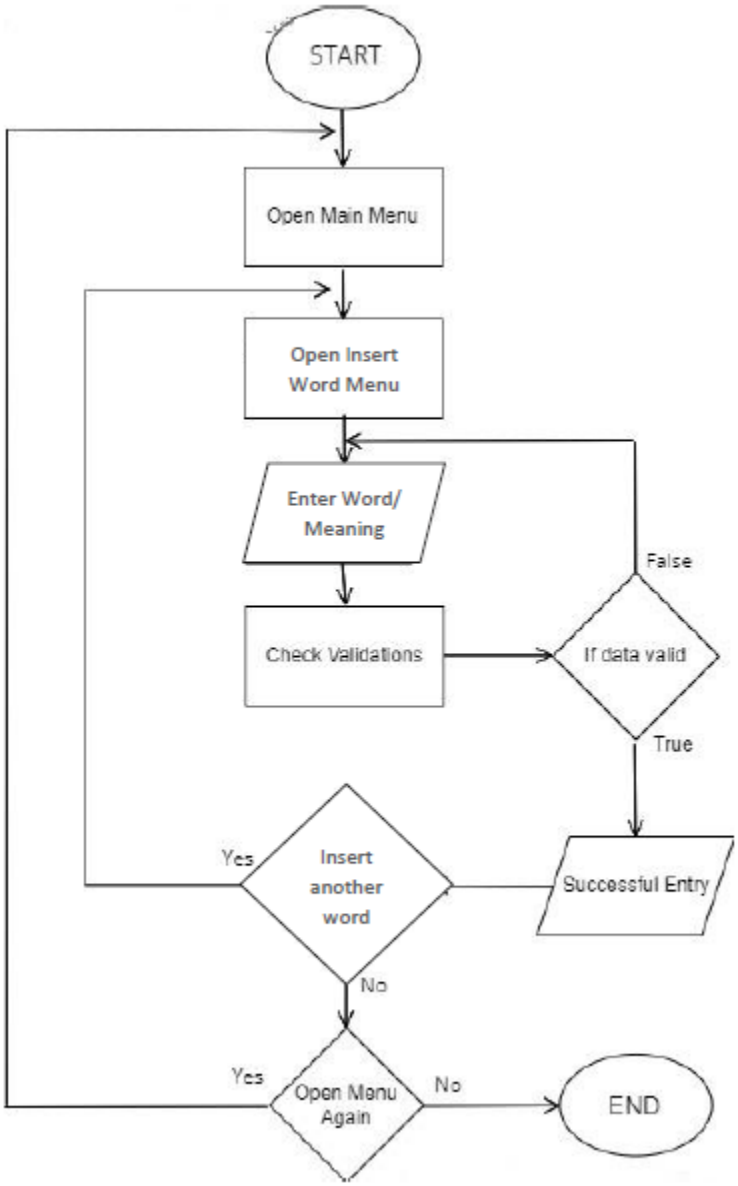
Project Scope

The project scope encompasses the development of a Digital Dictionary with essential features, focusing on user convenience and efficient data storage using a Binary Search Tree. The proposed project is a Digital Dictionary. Digital Dictionary is a machine-readable version of a standard dictionary; organized alphabetically. Digital Dictionaries bring convenience to the user as they can easily be accessed to find the words and their meanings without having need to carry hard-cover heavy dictionaries. This also makes the learning process for students easy.

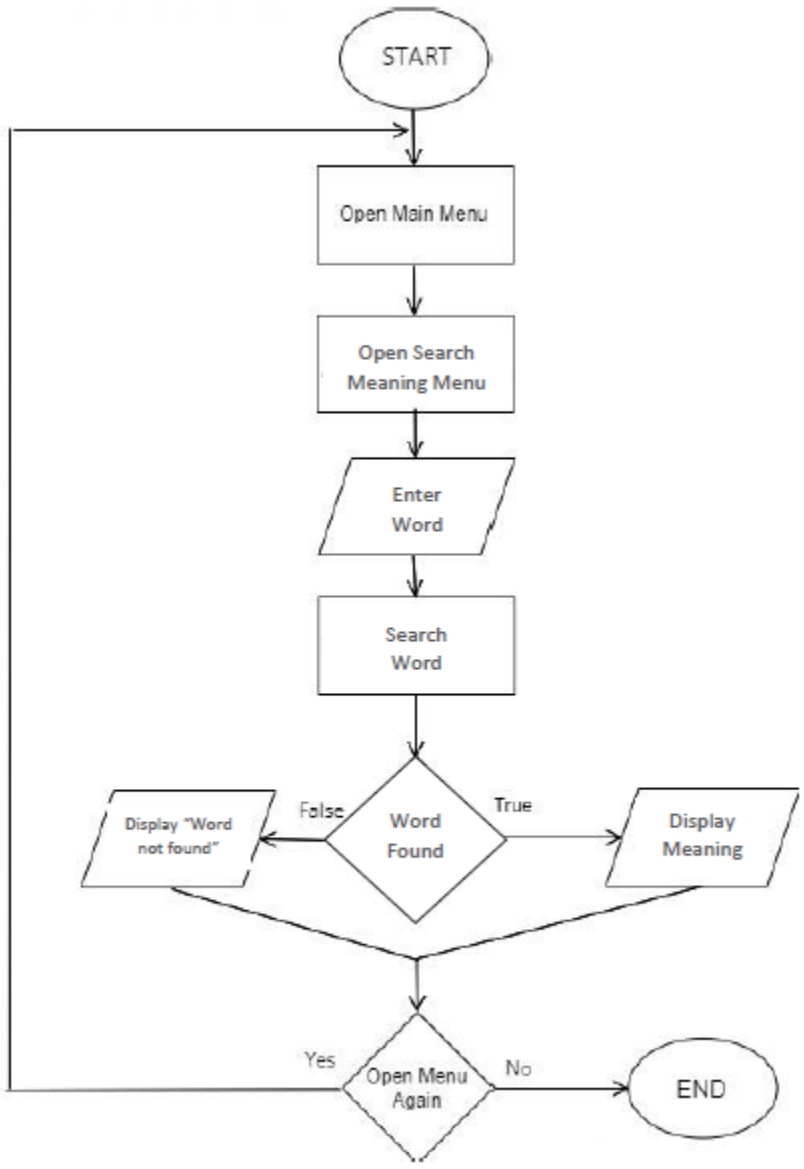
Workflow

**(Represented with Flowcharts instead of UML because OOP was not used in this project)*

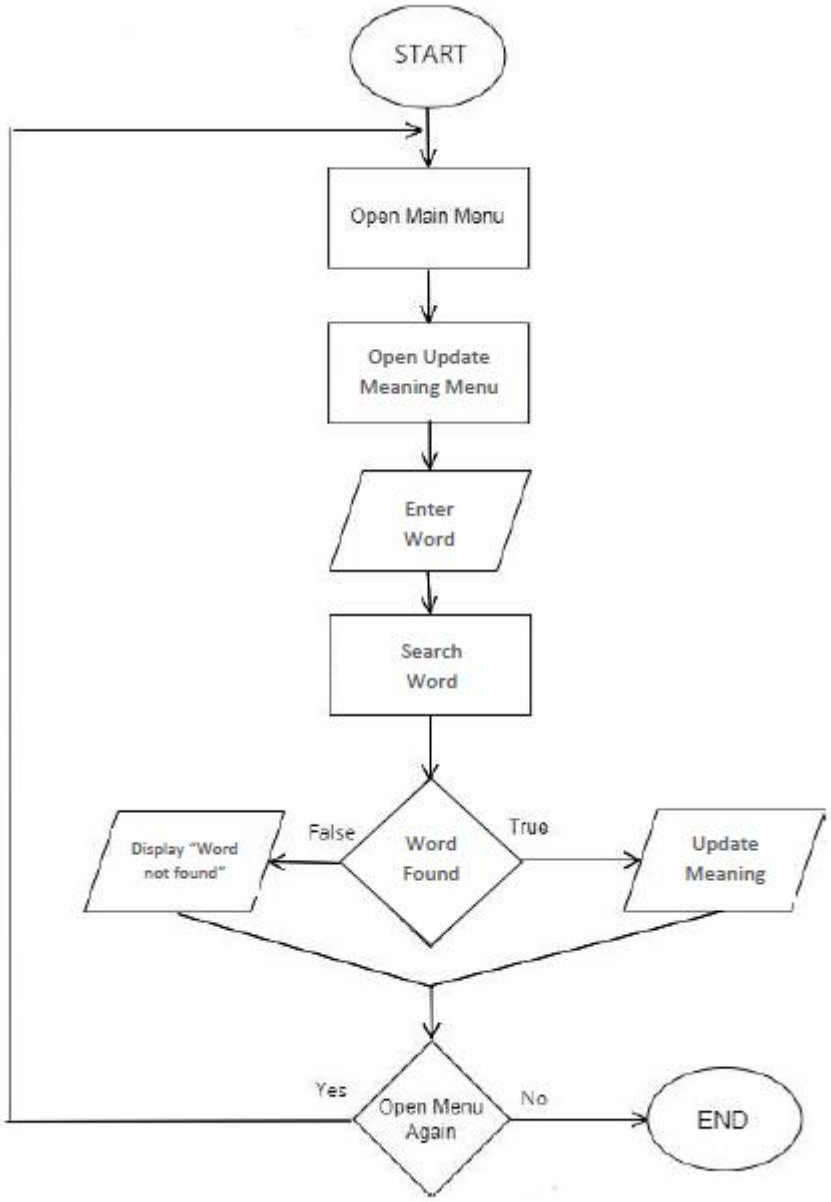
- **INSERT:**



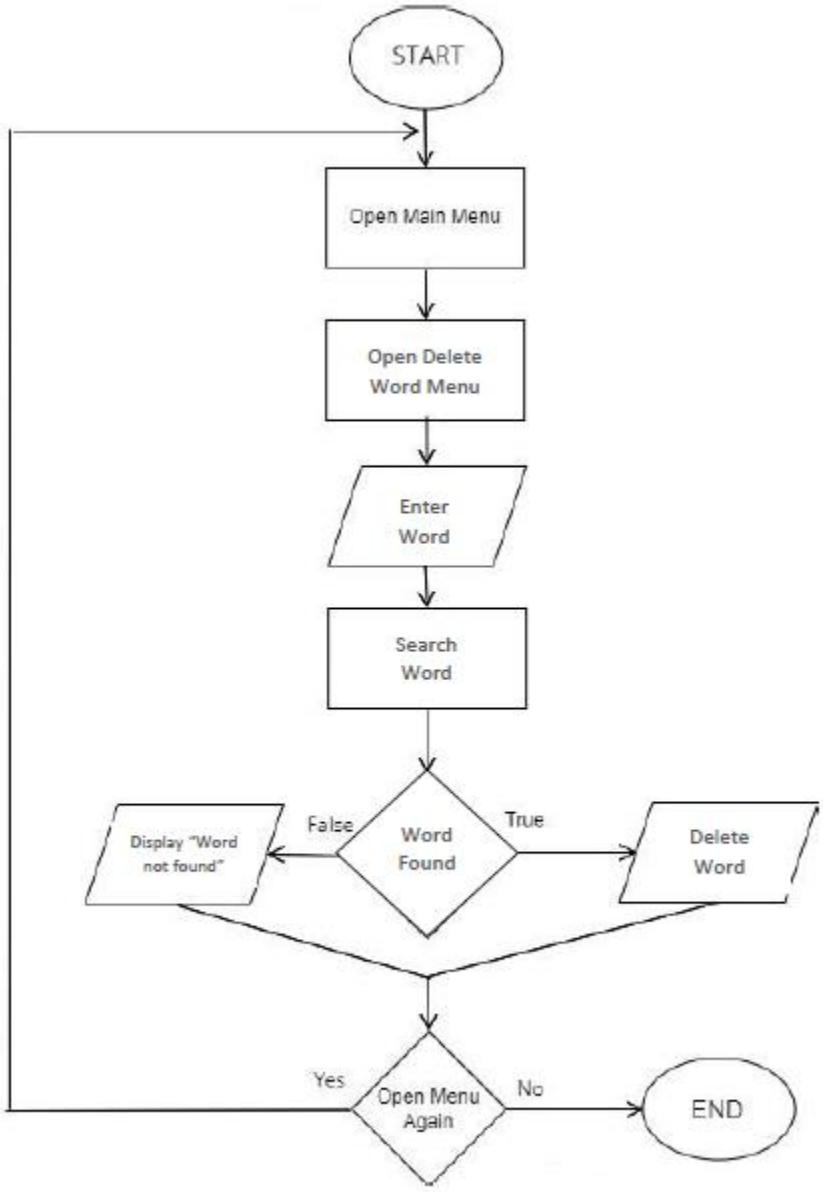
- **SEARCH:**


START
Open Main Menu
Open Search Meaning Menu
Enter Word
Search Word
Word Found
False
True
Display "Word not found"
Display Meaning
Open Menu Again
Yes
No
END

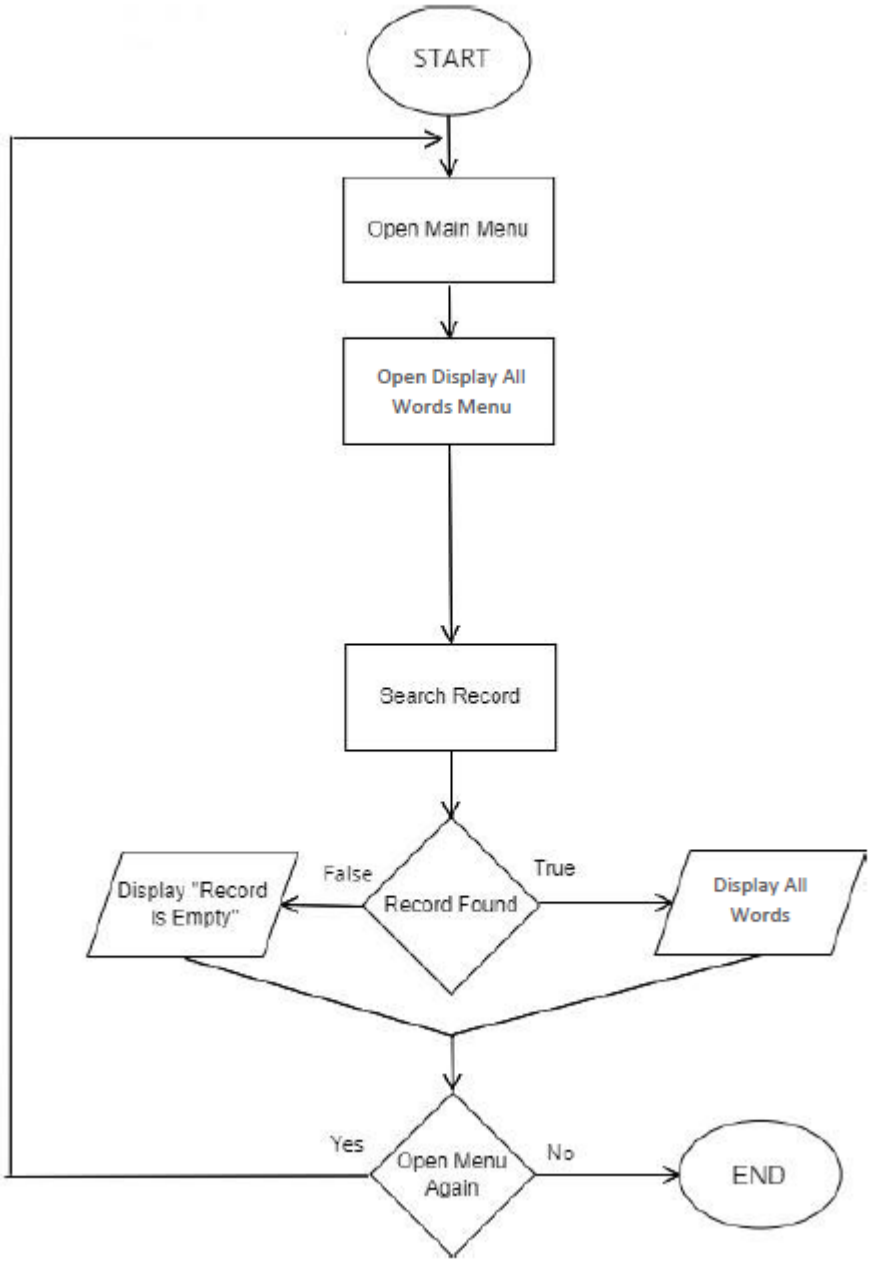
- **UPDATE:**


START
Open Main Menu
Open Update Meaning Menu
Enter Word
Search Word
Word Found
False
True
Display "Word not found"
Update Meaning
Open Menu Again
Yes
No
END

- **DELETE:**


START
Open Main Menu
Open Delete Word Menu
Enter Word
Search Word
Word Found
False
True
Display "Word not found"
Delete Word
Yes
Open Menu Again
No
END

- **DISPLAY:**


START
Open Main Menu
Open Display All Words Menu
Search Record
Record Found
False
True
Display "Record Is Empty"
Display All Words
Open Menu Again
Yes
No
END

Overview of project

The main process in this digital dictionary is the search process where user inputs the word and gets the meaning of it. User can also add a new word with its meaning if it doesn't already exist in the dictionary. Furthermore, there is an option of deleting a word and meaning from the dictionary. Another functionality this project provides is that user can update the meaning of a word if required. The display all operation will allow the user to see all the words and the meanings altogether at once.

Tools and Technologies

The project is implemented using C++ and utilizes multiple libraries such as for strings and file handling and even exception handling. File handling is employed for data storage, and the user interface is designed for ease of interaction.

Project features

Functional Requirements:

1. Insert Word/Meaning
2. Search Word/Meaning
3. Update Meaning
4. Delete Word/Meaning
5. Display All Words/Meanings
6. Filing

Non-Functional Requirements:

1. User-friendly Interface
2. Efficient Search Algorithm (BST)
3. Data Security and Integrity

DSA implemented concepts

The Digital Dictionary project implements concepts such as Binary Search Tree (BST) for efficient word storage and retrieval. File handling is employed for persistent data storage.

- **CODE:**

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<cstdio>
#include<cstring>
#include<windows.h>
#include<string.h>
#include<fstream>

using namespace std;

struct Node {
    char word[50];
    char meaning[100];
    char synonym[100];
    char partOfSpeech[50];
    Node* left;
    Node* right;
};

Node* root = nullptr;

Node* CreateNode(const char* word, const char* meaning, const char* synonym,
const char* partOfSpeech);

Node* InsertNode(Node* root, const char* word, const char* meaning, const char*
synonym, const char* partOfSpeech);

void InOrderTraversal(Node* root);

bool SearchWord(Node* root, const char* word);

void UpdateDetails(Node*& root, const char* word, const char* newMeaning, const
char* newSynonym, const char* newPartOfSpeech);
```

```
Node* DeleteWord(Node* root, const char* word);

Node* DeleteWordInternal(Node* root, const char* word);

Node* FindMinValueNode(Node* root);

void InOrderTraversalToFile(Node* root, ofstream& outFile);

void WriteToFile(const char* word, const char* meaning, const char* synonym,
const char* partOfSpeech);

void ReadFromFile();

void MenuHeader();

int main() {
    system("color 0B");

    Beep(2020, 1100);
    char l = 219;

    cout << endl << endl << endl;
    cout << "Loading.";
    cout << endl;
    for (int i = 0; i < 5; i++) {
        Beep(2000, 500);
        cout << l;
    }

    try {
        int choice1;
        char choice2[10], choice3[10];
        bool terminate = false;
        char word[50], meaning[100], synonym[100], partOfSpeech[50];
        cout << endl;

        do {
            system("cls");
            MenuHeader();


            cout << "Enter serial no. from the menu to proceed: ";
            cin >> choice1;

            switch (choice1) {
```

```
            case 1:
                do {
                    cout << "\nInput Word: ";
                    cin >> word;
                    cout << "Input Meaning: ";
                    cin.ignore();
                    cin.getline(meaning, sizeof(meaning));
                    cout << "Input Synonym: ";
                    cin.getline(synonym, sizeof(synonym));
                    cout << "Input Part of Speech: ";
                    cin.getline(partOfSpeech, sizeof(partOfSpeech));
                    root = InsertNode(root, word, meaning, synonym,
partOfSpeech);
                    WriteToFile(word, meaning, synonym, partOfSpeech);
                    cout << "Enter another word? (yes/no): ";
                    cin >> choice3;
                } while (strcmp(choice3, "yes") == 0 || strcmp(choice3, "Yes") ==
0);
                cout << "Words added successfully!" << endl;
                break;
            case 2:
                cout << "\nInput Word: ";
                cin >> word;
                SearchWord(root, word);
                break;
            case 3:
                cout << "\nInput Word: ";
                cin >> word;
                UpdateDetails(root, word, meaning, synonym, partOfSpeech);
                break;
            case 4:
                cout << "\nInput Word: ";
                cin >> word;
                root = DeleteWord(root, word);
                break;
            case 5:
                cout << endl << "Fetching Data." << endl;
                for (int i = 0; i < 5; i++) {
                    Beep(2000, 500);
                    cout << l;
                }
                system("cls");
                MenuHeader();
                cout << "                _____________________________________" << endl;
                cout << "               |  _____________________________  |" << endl;
```

```
                cout << "         | |                           | |" << endl;
                cout << "         | |    COMPLETE DICTIONARY    | |" << endl;
                cout << "         | |___________________________| |" << endl;
                cout << "         |_____________________________|" << endl << endl;
                ReadFromFile();
                InOrderTraversal(root);
                break;
            case 6:
                terminate = true;
                strcpy_s(choice2, "no");
                break;
            default:
                cout << endl << "Invalid serial no. entered!" << endl << endl;
            }

            if (!terminate) {
                cout << "Would you like to open the menu again? (yes/no): ";
                cin >> choice2;
                cout << endl;
            }
        } while (strcmp(choice2, "yes") == 0 || strcmp(choice2, "Yes") == 0);
    }
    catch (const exception& e) {
        cerr << "Exception handled: " << e.what() << endl;
    }

    return 0;
}

Node* CreateNode(const char* word, const char* meaning, const char* synonym,
const char* partOfSpeech) {
    Node* newNode = new Node();
    strcpy_s(newNode->word, word);
    strcpy_s(newNode->meaning, meaning);
    strcpy_s(newNode->synonym, synonym);
    strcpy_s(newNode->partOfSpeech, partOfSpeech);
    newNode->left = nullptr;
    newNode->right = nullptr;
    return newNode;
}

Node* InsertNode(Node* root, const char* word, const char* meaning, const char*
synonym, const char* partOfSpeech) {
    if (root == nullptr) {
```

```
        root = CreateNode(word, meaning, synonym, partOfSpeech);
    }
    else if (strcmp(word, root->word) < 0) {
        root->left = InsertNode(root->left, word, meaning, synonym,
partOfSpeech);
    }
    else if (strcmp(word, root->word) > 0) {
        root->right = InsertNode(root->right, word, meaning, synonym,
partOfSpeech);
    }

    return root;
}

void InOrderTraversal(Node* root) {
    if (root != nullptr) {
        InOrderTraversal(root->left);
        cout << "Word: " << root->word << endl;
        cout << "Meaning: " << root->meaning << endl;
        cout << "Synonym: " << root->synonym << endl;
        cout << "Part of Speech: " << root->partOfSpeech << endl;
        InOrderTraversal(root->right);
    }
}

bool SearchWord(Node* root, const char* word) {
    if (root == nullptr) {
        cout << "The word is not in the Dictionary!" << endl;
        return false;
    }
    else if (strcmp(word, root->word) == 0) {
        cout << "_____" << endl
<< endl;
        cout << "Word: " << root->word << endl;
        cout << "Meaning: " << root->meaning << endl;
        cout << "Synonym: " << root->synonym << endl;
        cout << "Part of Speech: " << root->partOfSpeech << endl;
        cout << "_____" <<
endl;
        return true;
    }
    else if (strcmp(word, root->word) < 0) {
        return SearchWord(root->left, word);
    }
    else if (strcmp(word, root->word) > 0) {
```

```
        return SearchWord(root->right, word);
    }
    return false;
}

void UpdateDetails(Node*& root, const char* word, const char* newMeaning, const
char* newSynonym, const char* newPartOfSpeech) {
    if (root == nullptr) {
        cout << "The word is not in the Dictionary!" << endl;
    }
    else if (strcmp(word, root->word) == 0) {
        cout << "_____" << endl
<< endl;
        cout << "Existing Meaning: " << root->meaning << endl;
        cout << "Enter New Meaning: ";
        cin.ignore();
        cin.getline(root->meaning, sizeof(root->meaning));

        cout << "Existing Synonym: " << root->synonym << endl;
        cout << "Enter New Synonym: ";
        cin.getline(root->synonym, sizeof(root->synonym));

        cout << "Existing Part of Speech: " << root->partOfSpeech << endl;
        cout << "Enter New Part of Speech: ";
        cin.getline(root->partOfSpeech, sizeof(root->partOfSpeech));

        WriteToFile(root->word, root->meaning, root->synonym, root-
>partOfSpeech);

        cout << "Details updated successfully!" << endl;
        cout << "_____" << endl
<< endl;
    }
    else if (strcmp(word, root->word) < 0) {
        UpdateDetails(root->left, word, newMeaning, newSynonym, newPartOfSpeech);
    }
    else if (strcmp(word, root->word) > 0) {
        UpdateDetails(root->right, word, newMeaning, newSynonym,
newPartOfSpeech);
    }
}

Node* DeleteWord(Node* root, const char* word) {
    bool wordExists = SearchWord(root, word);
```

```
    if (!wordExists) {
        return root;
    }

    root = DeleteWordInternal(root, word);

    ofstream outFile("Dictionary.txt");
    if (!outFile.is_open()) {
        cerr << "Error opening file for writing." << endl;
        exit(1);
    }

    InOrderTraversalToFile(root, outFile);

    cout << "The word is deleted successfully!" << endl;

    outFile.close();

    return root;
}

Node* DeleteWordInternal(Node* root, const char* word) {
    if (root == nullptr) {
        return root;
    }

    if (strcmp(word, root->word) < 0) {
        root->left = DeleteWordInternal(root->left, word);
    }
    else if (strcmp(word, root->word) > 0) {
        root->right = DeleteWordInternal(root->right, word);
    }
    else {
        if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        }
        else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        Node* temp = FindMinValueNode(root->right);
```

```
        strcpy_s(root->word, temp->word);
        strcpy_s(root->meaning, temp->meaning);
        strcpy_s(root->synonym, temp->synonym);
        strcpy_s(root->partOfSpeech, temp->partOfSpeech);

        root->right = DeleteWordInternal(root->right, temp->word);
    }

    return root;
}

Node* FindMinValueNode(Node* root) {
    Node* curr = root;
    while (curr->left != nullptr) {
        curr = curr->left;
    }
    return curr;
}

void InOrderTraversalToFile(Node* root, ofstream& outFile) {
    if (root != nullptr) {
        InOrderTraversalToFile(root->left, outFile);
        outFile << root->word << endl;
        outFile << root->meaning << endl;
        outFile << root->synonym << endl;
        outFile << root->partOfSpeech << endl;
        InOrderTraversalToFile(root->right, outFile);
    }
}

void WriteToFile(const char* word, const char* meaning, const char* synonym,
const char* partOfSpeech) {
    ofstream myfile("Dictionary.txt", ios::app);
    if (!myfile.is_open()) {
        cerr << "Error opening file for writing." << endl;
        exit(1);
    }

    myfile << word << endl;
    myfile << meaning << endl;
    myfile << synonym << endl;
    myfile << partOfSpeech << endl;

    if (myfile.fail()) {
        cerr << "Error writing to file." << endl;
```

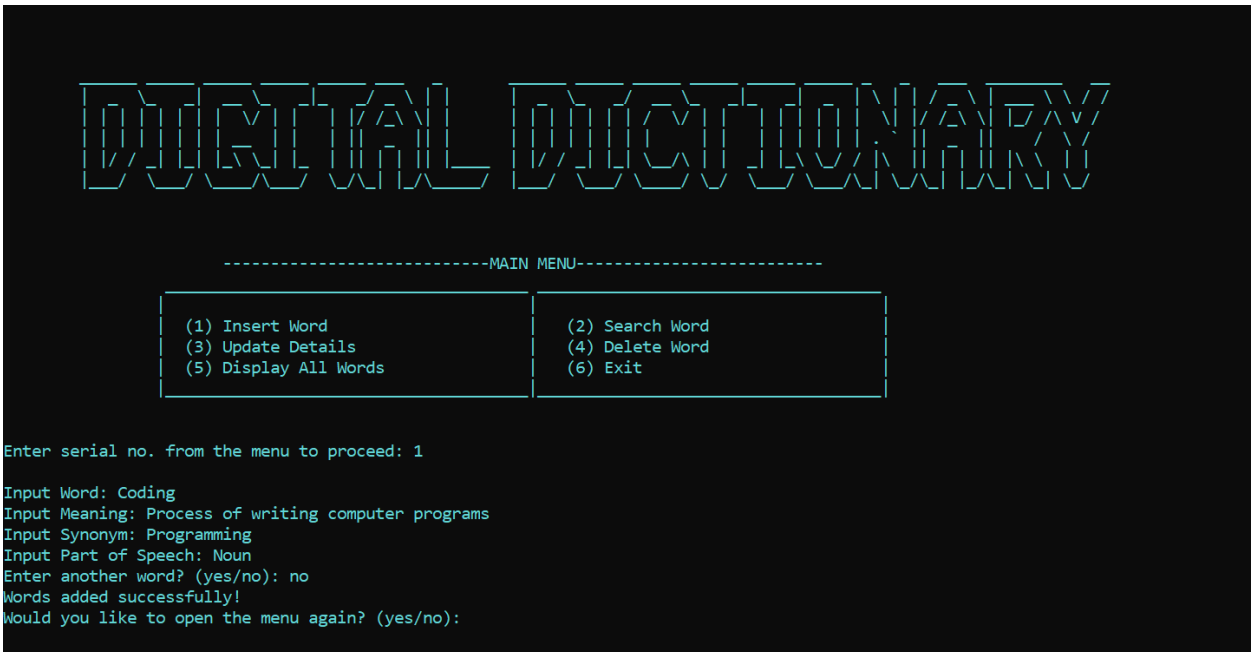
```
        exit(1);
    }

    myfile.close();
}

void ReadFromFile() {
    ifstream inputFile("Dictionary.txt");
    if (!inputFile.is_open()) {
        cerr << "Error opening file for reading." << endl;
        exit(1);
    }

    char word[50], meaning[100], synonym[100], partOfSpeech[50];

    while (inputFile.getline(word, sizeof(word))) {
        inputFile.getline(meaning, sizeof(meaning));
        inputFile.getline(synonym, sizeof(synonym));
        inputFile.getline(partOfSpeech, sizeof(partOfSpeech));

        root = InsertNode(root, word, meaning, synonym, partOfSpeech);
    }

    if (inputFile.fail() && !inputFile.eof()) {
        cerr << "Error reading from file." << endl;
        exit(1);
    }

    inputFile.close();
}

void MenuHeader() {
    cout << endl << endl << endl;

    cout << "\t_____ _____ _____ _____ _____ ___   _       _____ _____ _____
_____ _____ _____ _   _   ___  ________   __" << endl;
    cout << "\t|  _  \\_   _|  __ \\_   _|_   _/ _ \\ | |     |  _  \\_   _/  __
\\_   _|_   _|  _  | \\ | | / _ \\ | ___ \\ \\ / /" << endl;
    cout << "\t| | | | | | | |  \\/ | |   | |/ /_\\ \\| |     | | | | | | |
/  \\/ | |   | | | | | |  \\| |/ /_\\ \\| |_/ /\\ V / " << endl;
    cout << "\t| | | | | | | | __  | |   | ||  _  || |     | | | | | | | |     |
|   | | | | | | . ` ||  _  ||    /  \\ /  " << endl;
    cout << "\t| |/ / _| |_| |_\\ \\_| |_  | || | | || |____ | |/ / _| |_|
\\__/\\ | |  _| |_\\ \\_/ / |\\  || | | || |\\ \\  | |  " << endl;
```

```
    cout << "\t|___/  \\___/ \\____/\\___/  \\_/\\_| |_/\\_____/  |___/  \\___/
\\____/ \\_/  \\___/ \\___/\\_| \\_/\\_| |_/\\_| \\_| \\_/  " << endl << endl;

    cout << endl << endl;
    cout << "\t\t       ----------------------------MAIN MENU----------------------
------" << endl;

    cout << "\t\t _____________________________________
_____________________________________" << endl;
    cout <<
"\t\t|                                     |
|" << endl;
    cout << "\t\t|  (1) Insert Word                    |   (2) Search
Word                   |" << endl;
    cout << "\t\t|  (3) Update Details                 |   (4) Delete
Word                   |" << endl;
    cout << "\t\t|  (5) Display All Words              |   (6)
Exit                         |" << endl;
    cout <<
"\t\t|_____________________________________|_____________________________________
|" << endl;
    cout << endl << endl;
}
```

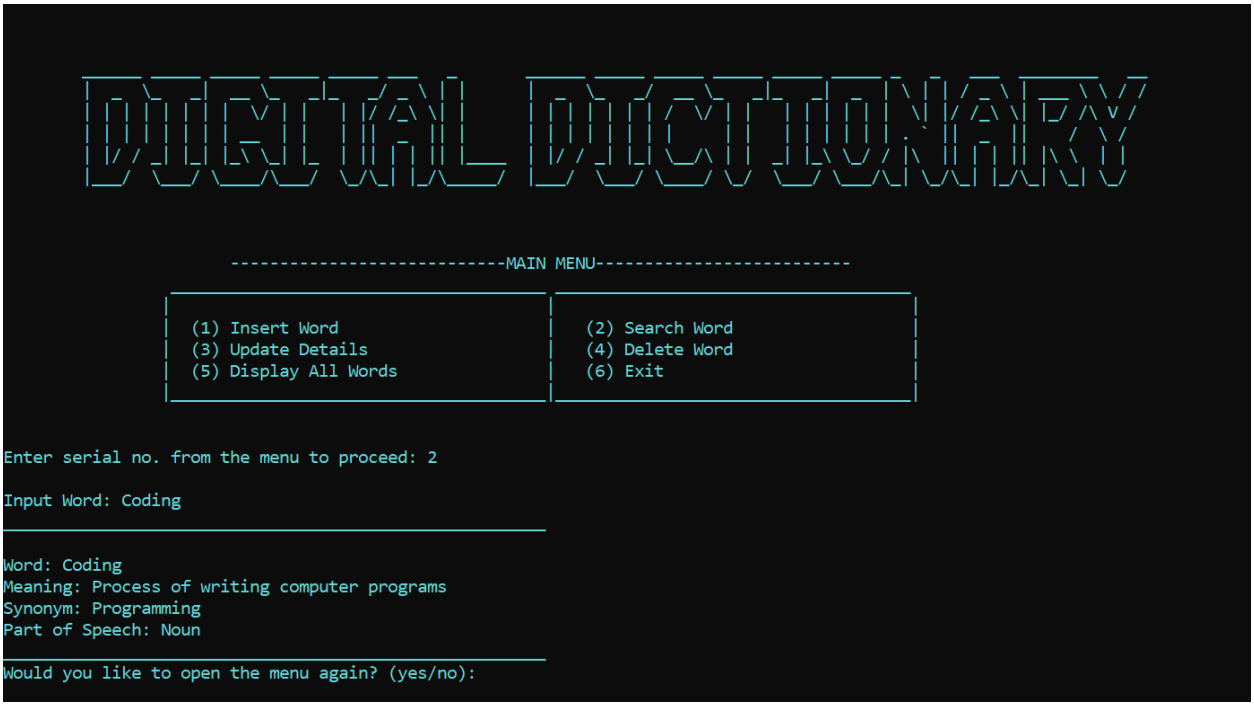
Output

- **MAIN MENU:**

- **INSERTION:**

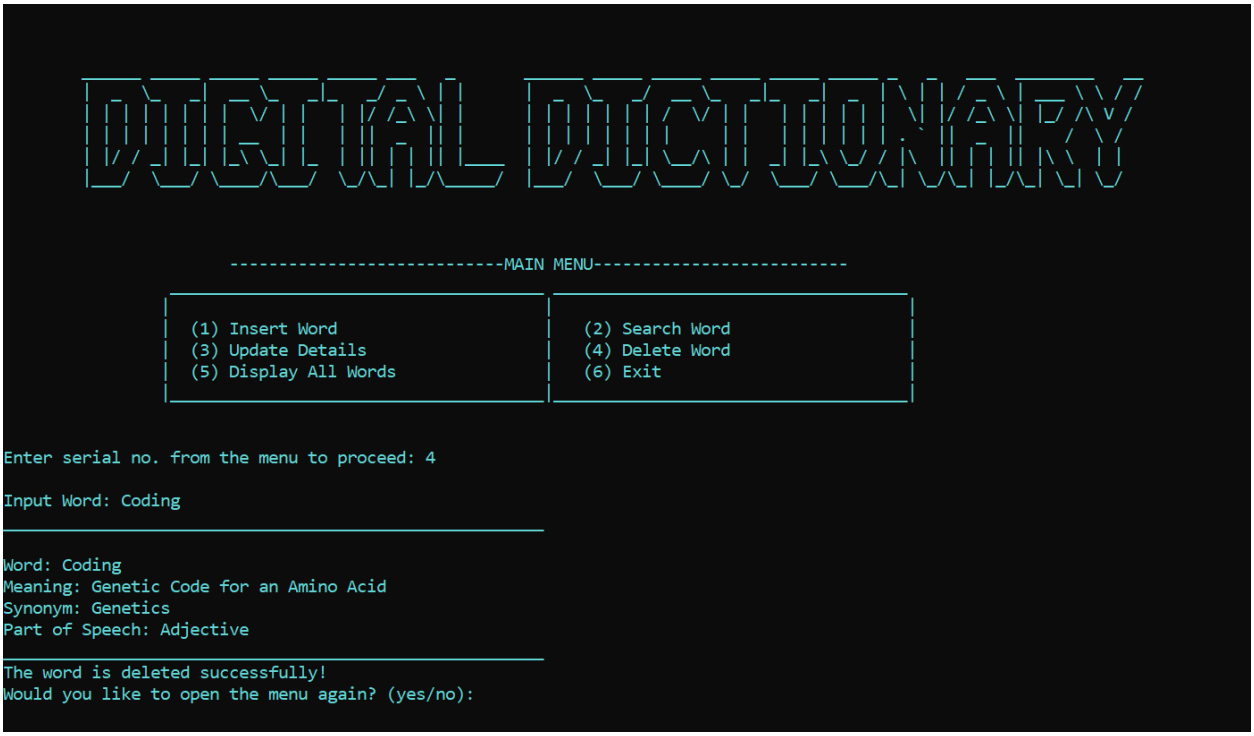


- **SEARCHING:**



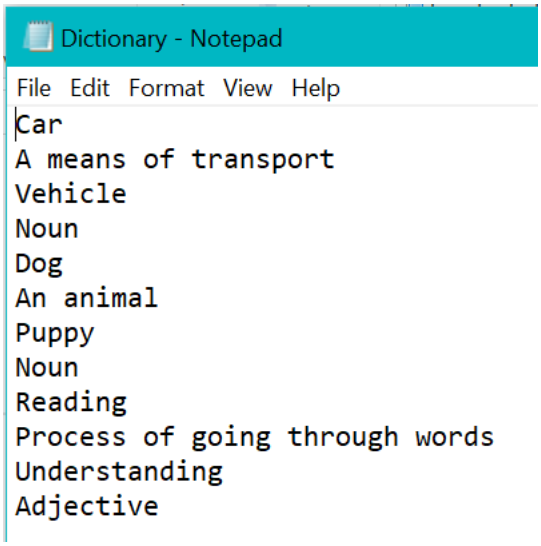
- **UPDATING:**

- **DELETION:**


DIGITAL DICTIONARY
MAIN MENU
(1) Insert Word
(2) Search Word
(3) Update Details
(4) Delete Word
(5) Display All Words
(6) Exit
Enter serial no. from the menu to proceed: 4
Input Word: Coding
Word: Coding
Meaning: Genetic Code for an Amino Acid
Synonym: Genetics
Part of Speech: Adjective
The word is deleted successfully!
Would you like to open the menu again? (yes/no):

- **DISPLAY:**

- **TEXT FILE:**



Future work

Future enhancements may include:

1. Implementation of more advanced search algorithms.
2. Integration with online dictionaries for real-time updates.
3. Multi-language support.
4. Saving data in databases.
5. More optimized menu and multiple data structures.

Conclusion

It can be observed that dictionaries are very important in every field of human endeavor. This project provides a computerized version of dictionary which will benefit the targeted audience across many sectors. The digital version of the project lessens the use of paper like in traditional way of keeping the record. The project also has the ability to update the meanings without needing to reprint the whole edition. This development will also reduce the effort as it reduces the time used for searching particular meaning. The use of Binary Search Tree makes the project efficient and fast. All the advantages of BST are encompassed in the project and major operations of BST are applied.

References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
2. Deitel, P., & Deitel, H. (2017). C++ How to Program (10th ed.). Pearson.
3. Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley.
4. Oracle. (n.d.). Java Platform, Standard Edition Documentation. Retrieved from https://docs.oracle.com/en/java/

5. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

6. IEEE Citation Guide. (n.d.). IEEE. Retrieved from https://www.ieee.org/

7. Smith, J. (2022). Digital Dictionary: A Binary Search Tree Implementation. Unpublished manuscript.

8. National Institute of Standards and Technology. (1997). Data Encryption Standard (FIPS PUB 46-3). Retrieved from https://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf