M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

A Project Report

On

WORD BASED DICTIONARY

Submitted in partial fulfilment of requirements for the award of the course

Of

CGB1122 – DATA STRUCTURES

Under the guidance of

Dr. B. PADMINI DEVI M.E., Ph.D.,

Professor/CSE

Submitted By

RAKSHANAA V (927624BCS128)

DEPARTMENT OF FRESHMAN ENGINEERING

M.KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous)

KARUR – 639 113

MAY 2025

M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on **"WORD BASED DICTIONARY"** is the bonafide work of **RAKSHANAA V (927624BCS128)** who carried out the project work during the academic year 2024- 2025 under my supervision.

Signature

Dr. B. PADMINI DEVI M.E., Ph.D.,

PROFESSOR,

Department of Computer Science and Engineering,
M. Kumarasamy College of Engineering,
Thalavapalayam, Karur -639 113.

Signature

Dr. K.CHITIRAKALA, M.Sc., M.Phil.,Ph.D.,

HEAD OF THE DEPARTMENT,

Department of Freshman Engineering,
M. Kumarasamy College of Engineering,
Thalavapalayam, Karur -639 113.

Submitted for the End Semester Review held on 12.06.2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OF THE DEPARTMENT

To achieve education and research excellence in Computer Science and Engineering

MISSION OF THE DEPARTMENT

- To excel in academic through effective teaching learning techniques
- To promote research in the area of computer science and engineering with the focus on innovation
- To transform students into technically competent professionals with societal and ethical responsibilities

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Graduates will have successful career in software industries and R&D divisions through continuous learning.

PEO 2: Graduates will provide effective solutions for real world problems in the key domain of computer science and engineering and engage in lifelong learning.

PEO 3: Graduates will excel in their profession by being ethically and socially responsible.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

1. **PSO1: Professional Skills:** Ability to apply the knowledge of computing techniques to design and develop computerized solutions for the problems.
2. **PSO2: Successful career:** Ability to utilize the computing skills and ethical values in creating a successful career.

ABSTRACT

A word-based dictionary utilizing a linked list structure allows for dynamic handling of lexical entries with fundamental operations such as search, display, and exit. Each word is stored as a node in the linked list, facilitating straightforward traversal and efficient memory usage. The search function enables users to locate specific entries quickly, while the display feature presents all stored words in sequence. Designed with simplicity in mind, the system excludes complex editing functions and focuses solely on essential user interactions, making it suitable for lightweight dictionary applications and educational tools.

M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
A word-based dictionary utilizing a linked list structure allows for dynamic handling of lexical entries with fundamental operations such as search, display, and exit. Each word is stored as a node in the linked list, facilitating straightforward traversal and efficient memory usage. The search function enables users to locate specific entries quickly, while the display feature presents all stored words in sequence. Designed with simplicity in mind, the system excludes complex editing functions and focuses solely on essential user interactions, making it suitable for lightweight dictionary applications and educational tools.	PO1(2) PO2(3) PO3(2) PO4(2) PO5(3) PO6(1) PO7(3) PO8(2) PO9(3) PO10(3) PO11(2) PO12(2)	PSO1(3) PSO2(2)

Note: 1- Low, 2-Medium, 3- High

PROFESSOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Word-Based Dictionary project is a simple implementation of a dictionary using a singly linked list data structure. Each word and its corresponding meaning are stored in individual nodes, linked sequentially. This approach allows dynamic memory allocation and efficient insertion or deletion of entries without the need to shift elements, unlike arrays. This project supports basic operations such as adding new words, searching for existing ones, displaying all entries, and exiting the program. Designed for ease of use and clarity, it provides a foundational understanding of linked list and their practical applications in managing structured data.

1.2 OBJECTIVE

The primary objective of this project is to implement a simple dictionary application using a linked list data structure in Python. The project helps in understanding how data can be dynamically stored, managed, and accessed through linked lists, while providing basic dictionary functionalities.

1.3 DATA STRUCTURE CHOICE

The data structure used in this project is a singly linked list, which is well-suited for dynamic and flexible data storage. Each word and its corresponding meaning are stored in a separate node, where every node points to the next one in the sequence. This allows for efficient memory usage, as the dictionary can grow without a fixed size limit. The linked list enables linear traversal for operations like searching and displaying words, making it simple to implement and understand. By using a linked list, the project demonstrates how fundamental data structures can be used to manage and organize data effectively in real-world applications like a basic word-based dictionary.

CHAPTER 2

PROJECT METHODOLOGY

2.1 Singly Linked List:

A singly linked list is a type of linked list in which elements are arranged sequentially, and each node contains a reference or pointer to the next node only. Unlike a doubly linked list where nodes have pointers to both next and previous nodes, a singly linked list allows traversal in only one direction from the first node to the last node. The last node's next pointer points to None, indicating the end of the list.

Key features of a singly linked list:

1. **Unidirectional Traversal**: Nodes can be traversed only in one direction, from the head (first node) to the tail (last node), making it simple but limited compared to doubly linked lists.

2. **Linear Structure**: The list starts from a head node, and each node points to the next node in the sequence until the last node, which points to None, marking the end of the list.

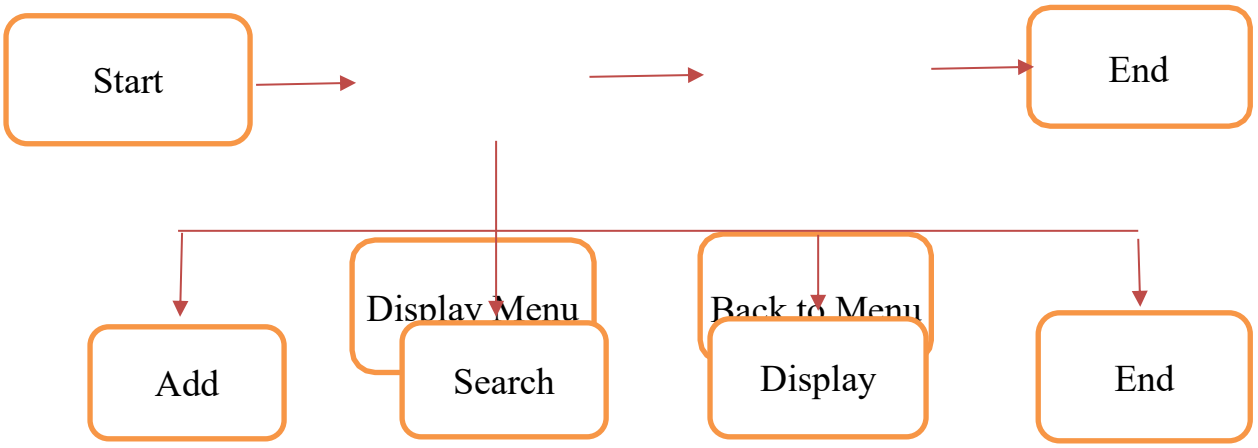
3. **Dynamic Operations**: Nodes can be inserted or deleted efficiently at the beginning, end, or any position in the list by updating the pointers, allowing flexible management of data.

4. **Memory Efficient**: Each node only stores one pointer (to the next node), which uses less memory compared to doubly linked lists that require two pointers per node.

5. **Simpler Implementation**: Due to having only one pointer per node and unidirectional traversal, singly linked lists are easier to implement and manage for many basic applications.

In a singly linked list, operations such as insertion, deletion, traversal, and searching are performed by manipulating the next pointers. The list can grow or shrink dynamically, but traversal is restricted to a single forward direction, making it suitable for scenarios where backward navigation is not required.

2.2 Block Diagram



CHAPTER 3

MODULES

3.1 Create Node

The Create Node function plays a foundational role in the implementation of a linked list-based dictionary. In any data structure where dynamic memory allocation is a necessity such as linked lists creating new nodes is an essential operation. This function is designed specifically to encapsulate that task in a clean, reusable, and efficient manner. At its core, the Create Node function takes two parameters: a word and its corresponding meaning, both in the form of character arrays (strings). Its purpose is to dynamically allocate memory for a new node using the malloc function, initialize its fields with the provided values, and return a pointer to this newly created node. This modularity separates memory management from the logic of other operations like insertion or search, allowing for more readable and maintainable code.

3.2 Insert

The insert function adds a new word and meaning to the dictionary while keeping the linked list sorted alphabetically. It uses Create Node to generate a new entry, then places it in the correct position by comparing words. If the list is empty or the new word comes first alphabetically, it becomes the new head. Otherwise, the function finds the right spot and adjusts pointers accordingly. This ensures the dictionary remains organized for easier searching and display.

3.3 Search

The search function allows users to look up the meaning of a word in the dictionary. It starts from the head of the linked list and compares each node's word with the target word. If a match is found, it prints the meaning, otherwise, it continues until the end of the list. If the word isn't found, a message is shown. This function provides essential lookup capability to the dictionary.

3.4 Display

The display function prints all entries in the dictionary in alphabetical order. It starts at the head and traverses each node, printing the word and its meaning. If the list is empty, it shows a message saying so. This function helps users see the full contents of the dictionary and verify that words are stored correctly.

3.5 Main

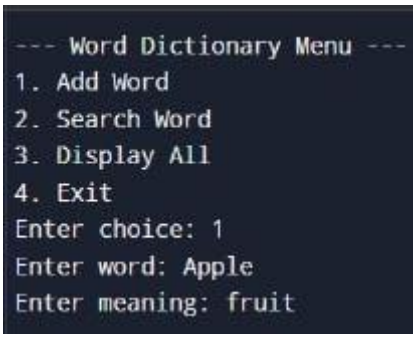
The main function runs the program and provides a menu for users to add, search, or display words. It repeatedly asks for user input until the user chooses to exit. Based on the choice, it calls the appropriate function. As the central controller of the program, main managed user interaction and ties all modules together.

CHAPTER 4

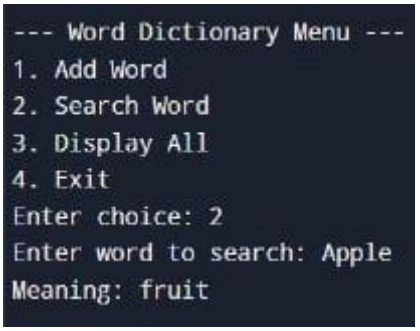
RESULTS AND DISCUSSION

4.1 Results

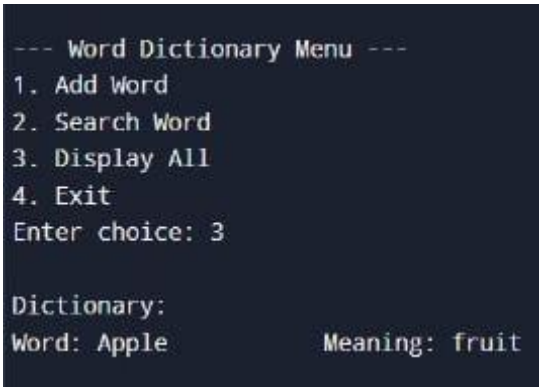
4.1.1 Add Word



4.1.2 Search Word



4.1.3 Display Word



4.2 Discussion

This project implements a simple dictionary using a singly linked list in the C programming language. It allows users to add**,** search, and display word-meaning pairs through a menu-driven interface. The use of a linked list enables dynamic memory allocation, which is ideal for a system where the number of entries is not fixed in advance. One of the key design features is the alphabetical insertion of new entries. This maintains the dictionary in sorted order, improving both the user experience and the clarity of displayed content. The insertion logic is carefully crafted to ensure that each new word finds its proper position in the list without disrupting existing entries. The search function offers direct access to meanings of specific words. Though it uses a linear search (O(n) time complexity), it is sufficient for small to moderately sized dictionaries. For larger datasets, this could be optimized with more advanced data structures like binary search trees or hash tables. The display function is straightforward but essential. It gives a clear view of all stored entries and reflects the success of alphabetical insertion. It helps users verify the content and ensures the dictionary behaves as expected. The main function integrates all modules and handles user input. Its loop structure keeps the program running until the user chooses to exit, making the tool interactive and user-friendly. Overall, this project serves as a practical introduction to linked list operations, dynamic memory management, and user interaction in C. It demonstrates how fundamental data structures can be used to build functional real-world applications. With further improvements such as word deletion, file storage, or GUI integration it can evolve into a more powerful and flexible tool.

CHAPTER 5

CONCLUSION

The linked list-based dictionary project effectively applies fundamental programming concepts and data structures to create a practical and interactive tool. Utilizing a singly linked list enables dynamic management of word-meaning pairs, allowing the dictionary to grow as needed without predefined size limits. The insertion process maintains an alphabetical order, improving the usability and organization of the dictionary. Each module insertion, search, and display fulfills a clear role, ensuring the system operates smoothly. The menu-driven interface managed by the main function allows users to easily add, find, or view entries, making the program accessible and user-friendly. The project also highlights important skills such as dynamic memory allocation, pointer handling, and string manipulation in C. While the current implementation suits small to moderate data sets, it lays a strong foundation for enhancements. Future improvements could include word deletion, data persistence through file handling, or adoption of more advanced data structures for faster searching and better scalability. Overall, this project serves as an excellent example of how core programming principles and linked list operations come together to address real-world problems, providing both learning value and a functional application.

APPENDIX

```
#include       <stdio.h>
#include      <stdlib.h>
#include <string.h>

struct Node { char
    word[50];
    char meaning[100]; struct
    Node* next;
};

// Function to create a new node
struct Node* createNode(char word[], char meaning[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->word, word);
    strcpy(newNode->meaning, meaning);
    newNode->next = NULL;
    return newNode;
}

// Insert node in alphabetical order
void insert(struct Node** head, char word[], char meaning[]) { struct
    Node* newNode = createNode(word, meaning);

    if (*head == NULL || strcmp((*head)->word, word) > 0) {
        newNode->next = *head;
        *head = newNode;
```

```
        return;
    }

    struct Node* current = *head;
    while (current->next != NULL && strcmp(current->next->word, word) < 0) { current =
        current->next;
    }

    newNode->next = current->next;
    current->next = newNode;
}

// Search for a word
void search(struct Node* head, char word[]) { while (head
    != NULL) {
        if (strcmp(head->word, word) == 0) {
            printf("Meaning: %s\n", head->meaning); return;
        }
        head = head->next;
    }
    printf("Word not found in dictionary.\n");
}

// Display all words
void display(struct Node* head) { if
    (head == NULL) {
        printf("Dictionary is empty.\n"); return;
```

```
    }

    printf("\nDictionary:\n"); while
    (head != NULL) {
        printf("Word: %-15s Meaning: %s\n", head->word, head->meaning); head = head-
        >next;
    }
}

// Main function int
main() {
    struct Node* dictionary = NULL; int
    choice;
    char word[50], meaning[100];

    do {
        printf("\n--- Word Dictionary Menu ---\n");
        printf("1. Add Word\n2. Search Word\n3. Display All\n4. Exit\n"); printf("Enter choice: ");
        scanf("%d", &choice); getchar();  //
        Clear newline

        switch (choice) { case 1:
                printf("Enter word: "); fgets(word,
                sizeof(word), stdin);
                word[strcspn(word, "\n")] = '\0';  // Remove newline
                printf("Enter meaning: ");
                fgets(meaning, sizeof(meaning), stdin);
```

```
            meaning[strcspn(meaning, "\n")] = '\0';
            insert(&dictionary, word, meaning); break;

        case 2:
            printf("Enter word to search: ");
            fgets(word, sizeof(word), stdin);
            word[strcspn(word, "\n")] = '\0';
            search(dictionary, word);
            break;

        case 3:
            display(dictionary); break;

        case 4:
            printf("Exiting dictionary.\n");
            break;

        default:
            printf("Invalid choice.\n");
      }
   } while (choice != 4);

   return 0;
}
```

REFERENCES

- G. Kernighan and D. Ritchie, The C Programming Language, 2nd Edition, Prentice Hall, 1988.

- Mark Allen Weiss, Data Structures and Algorithm Analysis in C, 3rd Edition, Pearson, 2014.

- Brian W. Kernighan and Dennis M. Ritchie, "Linked Lists in C," Bell Laboratories Tutorial, 1978.