REPORT

01

Word Dictionary Management System in C

02

Modules based on Stacks

03

Modules based on Binary Search Tree (BST)

04

Modules based on Recursion

01

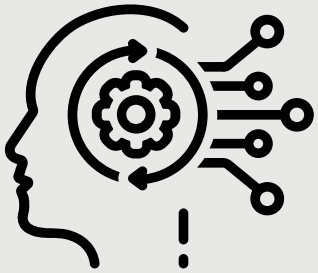
Word Dictionary Management System in C

1. Overview

This C program manages a dictionary of words and their synonyms and antonyms. It includes functionality to load word data from files, compute statistics (characters and vowels), add or delete words, sort the list alphabetically, detect palindromes, and filter words based on partial matches.

2. Data Structure

```c
typedef struct node {
    char word[50];
    char synonym[50];
    char antonym[50];
    int numChars[3];
    int numVowels[3];
    struct node* next;
} node;
```



countChars / countVowels

```
countChars("apple") → 5
countVowels("apple") → 2
```

createNode

```
TList n = createNode("apple", "fruit", "vegetable");
```

getSynWords / getAntoWords

```
FILE* f = fopen("synonyms.txt", "r");
TList list = getSynWords(f);
```

getInfWord

```
The number of char in this word is: 5
The number of vowels in this word is: 2
The synonym of this word is: fruit
The antonym of this word is: vegetable
```

deleteWord

```
deleteWord(f, "synonyms.txt", &syn, &ant, "apple");
```

palindromWord

```
madam
level
```

reverseString

```
char str[] = "apple";
reverseString(str); // Output: elppa
```

addWord

```
addWord(f, &synList, &antList, "kind", "nice", "mean");
```

sortWord Alphabetically

```
synList = sortWord(synList);
```

output

```
Words listed in ascending alphabetical order.
```

sortWord2 : List sorted in ascending order by number of characters in each word.

```
sortWord2(&synonyms);
```

sortWord3 : List sorted in descending order by number of vowels in each word

```
sortWord3(&synonyms);
```

getInfWord2 (Reverse lookup from synonym or antonym)

```
bright is synonym of the word: shiny
Characters: 5
Vowels: 1
```

syllable

```
Queue built where words are sorted by number
```

prounounciation

```
queues[0] contains short vowel words
queues[1] contains long vowel words
queues[2] contains diphthong words
```

toQueue

```
Converted doubly linked list to queue.
```

Modules based on Stacks

addWordStack

```
addWordStack(stack, "kind", "nice", "mean");
```

```
Word 'kind' added to stack maintaining order.
```

getInfWordStack

```
Word: kind
Synonym: nice
Antonym: mean
Number of Characters: 4 4 4
Number of Vowels: 1 2 2
```

deleteWordStack

```
Word 'kind' removed from stack if it exists.
```

updateWordStack

```
updateWordStack(stack, "kind", "gentle", NULL);
```

syllableStack

```
Stack sorted based on syllable count.
```

pronunciationStack

```
Words grouped into short, long vowel, and diphthong stacks.
```

StacktoList

```
Sorted stack converted to bidirectional list.
```

Modules based on Binary Search Tree (BST)

2. Data Structure

```
typedef struct TTree {
    char word[50];
    char synonym[50];
    char antonym[50];
    int numChars[3];
    int numVowels[3];
    struct TTree* left;
    struct TTree* right;
} TTree;
```

AddWordBSTT

```
TTree *root = NULL;
root = AddWordBSTT(root, "apple", "fruit", "vegetable");
```

deleteWordBST

```
Word 'apple' deleted from BST.
```

UpdateWordBST

```
Synonym and antonym of 'apple' updated.
```

TraversalBSTinOrder / preOrder / postOrder

```
Words printed in ascending order.
```

Height and Size

```
int h = Height(root);
int s = size(root);
```

```
Tree size: 5, Height: 3
```

BSTMirror

```
Left and right children of all nodes swapped.
```

LowestCommonAncestor

```
Returns pointer to the lowest common ancestor of "bat" and "cat"
```

CountNodesRanges

```
Number of words with length between 3 and 5: 4
```

inOrderSuccesor

```
Pointer to next word in BST order.
```

isBalancedBST

```
Returns true if BST is height-balanced.
```

BSTMerge

```
Merged tree built from two sorted BSTs.
```

Modules based on Recursion

countWordOccurence

```
FILE *f = fopen("text.txt", "r");
int count = countWordOccurence(f, "apple");
```

```
Number of times 'apple' appears: 3
```

removeWordOccurence

```
FILE *f = fopen("text.txt", "r");
FILE *updated = removeWordOccurence(f, "apple");
```

```
Returns new FILE* excluding all 'apple' words.
```

replaceWordOccurence

```
Replaces 'apple' with 'orange' throughout the file.
```

wordPermutation

```
wordPermutation("cat");
```

```
cat
cta
act
atc
tac
tca
```

subseqWord

```
subseqWord("abc");
```

```
abc,ab,ac,a,bc,b,c
```

longestSubseq

```
int lcs = longestSubseq("abcde", "ace", 5, 3);
printf("%d", lcs);
```

```
3 (the longest common subsequence is "ace")
```

isPalindromWord

```
bool res = isPalindromWord("radar");
```

```
true ("radar" is a palindrome)
```