



OVERVIEW OF PROJECT

The Dictionary using Binary Search Tree

The most common objective of computer programs is to store and retrieve data. Much of this book is about efficient ways to organize collections of data records so that they can be stored and retrieved quickly. In this section we describe a simple interface for such a collection, called a dictionary. The dictionary ADT provides operations for storing records, finding records, and removing records from the collection. This ADT gives us a standard basis for comparing various data structures. Simply speaking, we can say that any data structure that supports insert, search, and deletion is a “Dictionary”.

Dictionaries depend on the concepts of a search key and comparable objects. To implement the dictionary’s search function, we will require that keys be totally ordered. Ordering fields that are naturally multi-dimensional, such as a point in two or three dimensions, present special opportunities if we wish to take advantage of their multidimensional nature. This problem is addressed by spatial data structures.

Discussing Complexities:

A simple implementation for the Dictionary can be based on sorted or unsorted lists. When implementing the dictionary with an unsorted list, inserting a new record into the dictionary can be performed quickly by putting it at the end of the list. However, searching an unsorted list for a particular record requires $\Theta(n)$ time in the average case. For a large database, this is probably much too slow. Alternatively, the records can be stored in a sorted list. If the list is implemented using a linked list, then no speedup to the search operation will result from storing the records in sorted order. On the other hand, if we use a sorted array-based list to implement the dictionary, then binary search can be used to find a record in only $\Theta(\log n)$ time. However, insertion will now require $\Theta(n)$ time on average because, once the proper location for the new record in the sorted list has been found, many records might be shifted to make room for the new record.



IMPLEMENTATION

Program Code :

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

struct Dictionary
{
    char word[128], meaning[256];
    struct Dictionary *left, *right;
};

struct Dictionary *root = NULL;

FILE *fp1;

void insert(char *word, char *meaning);
void deleteNode(char *str);
void findElement(char *str);
void inorderTraversal(struct Dictionary *myNode);
void SearchSimilarWord(struct Dictionary* myNode, char *word);
void writeInFile(char word[128], char meaning[256]);
void openDictionary();
```

Main Function :

```
int main(void)
{
    int ch;
    char word[128], meaning[256];
    openDictionary();
    printf("\n -----DICTIONARY----- ");
    while (1)
    {
        printf("\n1. Insertion\n2. Deletion\n");
        printf("3. Searching\n4. Traversal\n");
        printf("5. Search Similar Word\n6. Exit\nEnter ur choice:");
        scanf("%d", &ch);
        getchar();
        switch (ch)
        {
            case 1:
                printf("Word to insert:");
                fgets(word, 100, stdin);
                printf("Meaning:");
                fgets(meaning, 256, stdin);
                insert(word, meaning);
                break;
```

Main Function :

```
case 2:
    printf("Enter the word to delete:");
    fgets(word, 100, stdin);
    deleteNode(word);
    break;
case 3:
    printf("Enter the search word:");
    fgets(word, 100, stdin);
    findElement(word);
    break;
case 4:
    inorderTraversal(root);
    break;
case 5:
    printf("Enter the word to search:");
    fgets(word, 100, stdin);
    printf("\nSimilar Words are : \n");
    SearchSimilarWord(root,word);
    break;
case 6:
    exit(0);
default:
    printf("You have entered wrong option\n");
    break;
}
}
return 0;
}
```

Dictionary nodes creation :

```
struct Dictionary *createNode(char *word, char *meaning)
{
    struct Dictionary *newnode;
    newnode = (struct Dictionary *)malloc(sizeof(struct Dictionary));
    strcpy(newnode->word, word);
    strcpy(newnode->meaning, meaning);
    newnode->left = newnode->right = NULL;
    return newnode;
}
```


Inserting word and meaning:

```
void insert(char *word, char *meaning)
{
    struct Dictionary *parent = NULL, *current = NULL, *newnode = NULL;
    int res = 0;
    if (!root)
    {
        root = createNode(word, meaning);
        if(x)
            writeInFile(word, meaning);
        return;
    }
    for (current = root; current != NULL;
        current = (res > 0) ? current->right : current->left)
    {
        res = strcasecmp(word, current->word);
        if (res == 0)
        {
            printf("Duplicate entry!!\n");
            return;
        }
        parent = current;
    }
    newnode = createNode(word, meaning);
    if(x)
        writeInFile(word, meaning);

    res > 0 ? (parent->right = newnode) : (parent->left = newnode);
    return;
}
```

Data Storing in File.

```
void writeInFile(char word[128],char meaning[256])
{
    fp1 = fopen("E:\\Win K\\3rd SEM\\DSA\\DSA Project\\first.txt","a");
    fprintf(fp1,"\n");
    fprintf(fp1,"%s",word);
    fprintf(fp1,"%s",meaning);
    fclose(fp1);
}

int x=0;
void openDictionary(){
    fp1=fopen("E:\\Win K\\3rd SEM\\DSA\\DSA Project\\first.txt","r");
    char word[128];
    char meaning[256];
    char ch;
    while((ch = fgetc(fp1)) != EOF ){
        if((ch== '`')){
            fscanf(fp1,"%s",word);
            fscanf(fp1,"%s",meaning);
            insert(word,meaning);
        }
    }
    x=1;
}
```

Finding a word from Dictionary:

```
void findElement(char *str)
{
    struct Dictionary *temp = NULL;
    int flag = 0, res = 0;
    if (root == NULL)
    {
        printf("No words are there in Dictionary\n");
        return;
    }
    temp = root;
    while (temp)
    {
        if ((res = strcasecmp(temp->word, str)) == 0)
        {
            printf("Word   : %s", str);
            printf("Meaning: %s", temp->meaning);
            flag = 1;
            break;
        }
        temp = (res > 0) ? temp->left : temp->right;
    }
    if (!flag)
        printf("Word doesnot exist in Dictionary\n");
    return;
}
```

Search for similar word.

```
void SearchSimilarWord(struct Dictionary* myNode, char word[100]){
    if(myNode)
    {
        SearchSimilarWord(myNode->left, word);
        char s1[128];
        char s2[128];
        strcpy(s1, myNode->word);
        strcpy(s2, word);
        int j=0, x=0;
        for(int i=0; i<strlen(s1); i++){
            if(s2[j]!='\0' && s1[i]==s2[j]){
                j++;
                if(j==strlen(s2)-1){
                    x=1; break;
                }
            }
            else if(j!=0){
                j=0;
            }
        }
        if(x) printf(" %s", myNode->word);
        x=0;
        SearchSimilarWord(myNode->right, word);
    }
}
```

Printing words in Lexicographic order:

```
void inorderTraversal(struct Dictionary *myNode)
{
    if (myNode)
    {
        inorderTraversal(myNode->left);
        printf("Word      : %s", myNode->word);
        printf("\n");
        printf("Meaning : %s", myNode->meaning);
        printf("\n");
        inorderTraversal(myNode->right);
    }
    return;
}
```

Deletion from dictionary:

```
void deleteNode(char *str)
{
    struct Dictionary *parent = NULL, *current = NULL, *temp = NULL;
    int flag = 0, res = 0;
    if (!root)
    {
        printf("No Words present in the Dictionary!!\n");
        return;
    }
    current = root;
    while (1)
    {
        res = strcasecmp(current->word, str);
        if (res == 0)
            break;
        flag = res;
        parent = current;
        current = (res > 0) ? current->left : current->right;
        if (current == NULL)
            return;
    }
}
```

Deletion of Leaf Node.

```
/* deleting leaf node */
if (current->right == NULL)
{
    if (current == root && current->left == NULL)
    {
        free(current);
        root = NULL;
        return;
    }
    else if (current == root)
    {
        root = current->left;
        free(current);
        return;
    }

    flag > 0 ? (parent->left = current->left) : (parent->right = current->left);
}
else
```

Deletion node with single child :

```
{
    /* delete node with single child */
    temp = current->right;
    if (!temp->left)
    {
        temp->left = current->left;
        if (current == root)
        {
            root = temp;
            free(current);
            return;
        }
        flag > 0 ? (parent->left = temp) : (parent->right = temp);
    }
    else
```


Deletion node with 2 child :

```
{
    /* delete node with two children */
    struct Dictionary *successor = NULL;
    while (1)
    {
        successor = temp->left;
        if (!successor->left)
            break;
        temp = successor;
    }
    temp->left = successor->right;
    successor->left = current->left;
    successor->right = current->right;
    if (current == root)
    {
        root = successor;
        free(current);
        return;
    }
    (flag > 0) ? (parent->left = successor) : (parent->right = successor);
}
}
free(current);
return;
}
```

OUTPUT

Insertion of Data :

```
-----DICTIONARY-----
```

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit

```
Enter ur choice:1  
Word to insert:Chillax  
Meaning:calm down
```

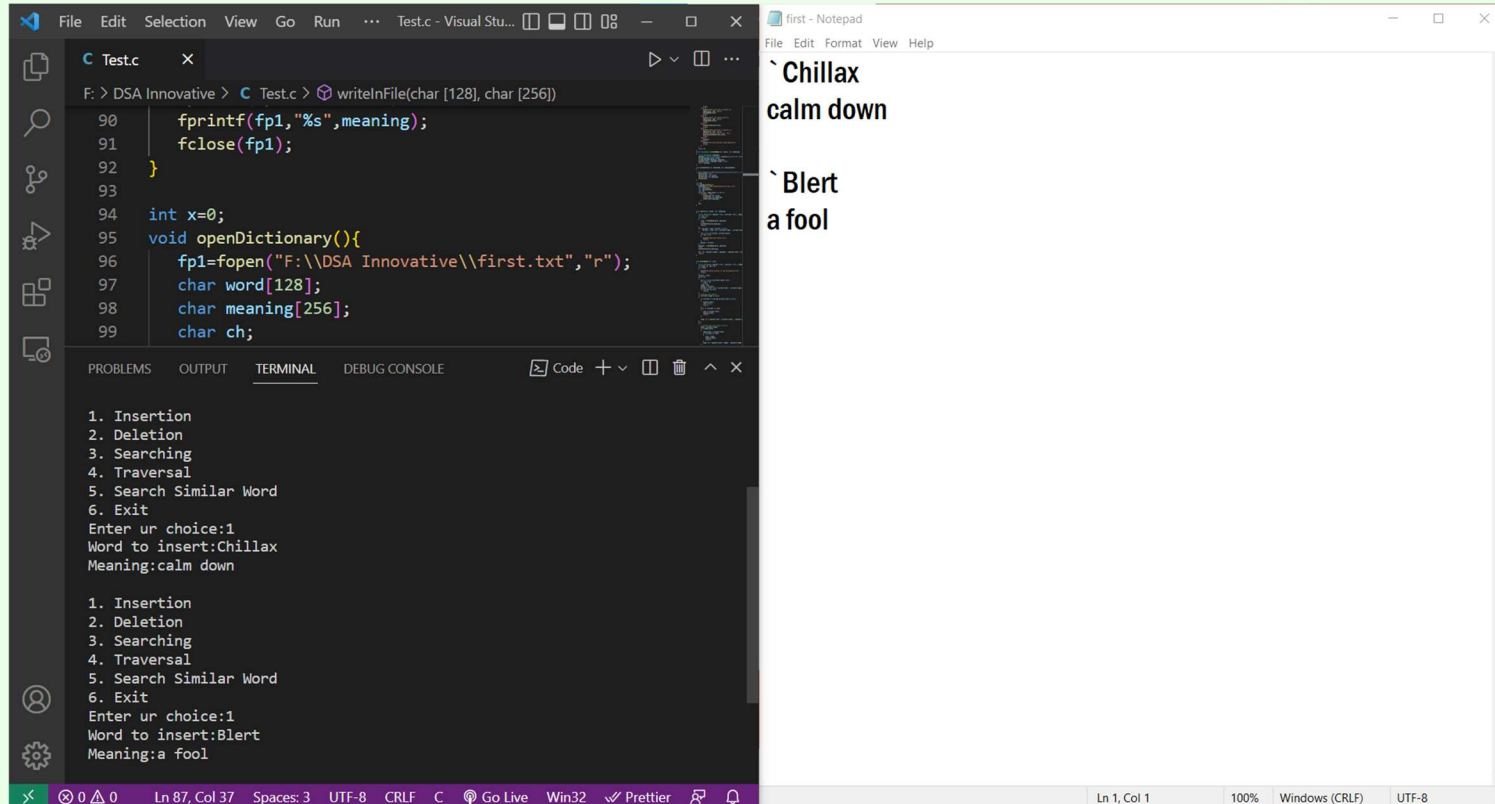
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit

```
Enter ur choice:1  
Word to insert:Blert  
Meaning:a fool
```

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit

```
Enter ur choice:1  
Word to insert:Comp  
Meaning:to give goods and services
```

Side by Side saving in File :



The image shows a side-by-side comparison of a file's content. On the left, the Visual Studio Code editor displays a C program in a file named 'Test.c'. The program includes a menu with options like Insertion, Deletion, Searching, Traversal, and Search Similar Word. It prompts the user to enter a choice, a word to insert, and its meaning. The code uses `fopen`, `fprintf`, and `fclose` to write to a file. The terminal window shows the program's execution, confirming the insertion of 'Chillax' with the meaning 'calm down' and 'Blert' with the meaning 'a fool'. On the right, a Notepad window shows the content of the file 'first.txt' after it has been saved. The text in Notepad matches the output shown in the VS Code terminal, demonstrating that the file was saved correctly with the latest changes.

```
F: > DSA Innovative > C Test.c > writeInFile(char [128], char [256])

90     fprintf(fp1,"%s",meaning);
91     fclose(fp1);
92 }
93
94 int x=0;
95 void openDictionary(){
96     fp1=fopen("F:\\DSA Innovative\\first.txt","r");
97     char word[128];
98     char meaning[256];
99     char ch;
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:1
Word to insert:Chillax
Meaning:calm down

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:1
Word to insert:Blert
Meaning:a fool
```

first - Notepad

```
` Chillax
calm down

` Blert
a fool
```

Previous saved words can be retrieved from Dictionary and can perform operations on it :

```
PS C:\Users\NAITIK> cd "f:\DSA Innovative\" ; if ($?) { gcc Test.c -o Test } ; if ($?) { .\Test }  
  
-----DICTIONARY-----  
1. Insertion  
2. Deletion  
3. Searching  
4. Traversal  
5. Search Similar Word  
6. Exit  
Enter ur choice:4  
Word   : Angered  
Meaning : fill  
Word   : Blert  
Meaning : a  
Word   : Chillax  
Meaning : calm  
Word   : Comp  
Meaning : to
```

Searching of Word :

```
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:1
Word to insert:Angered
Meaning:fill someone with anger

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:3
Enter the search word:Comp
Word   : Comp
Meaning: to give goods and services
```

Traversing whole Dictionary :

```
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:4
Word    : Angered

Meaning : fill someone with anger

Word    : Blert

Meaning : a fool

Word    : Chillax

Meaning : calm down

Word    : Comp

Meaning : to give goods and services
```

You can search for similar words too 😊

```
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:5
Enter the word to search:C

Similar Words are :
Chillax
Comp
```


Deletion from Dictionary :

```
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:2
Enter the word to delete:Chillax

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:4
Word    : Angered

Meaning : fill someone with anger

Word    : Blert

Meaning : a fool

Word    : Comp

Meaning : to give goods and services
```

**Exit the program but your inputs
are saved so, no worries :**

```
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Search Similar Word
6. Exit
Enter ur choice:6
PS F:\DSA Innovative> 
```