Batch: A1 Roll No.: 16010123012

Experiment No. 9

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implement a dictionary for some real world application. Use C/C++ or python.

Objective: To implement a dictionary for real world application using C.

Expected Outcome of Experiment:

CO	Outcome
3	Describe concepts of advanced data structures like set, map & dictionary.

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.geeksforgeeks.org/binary-tree-data-structure/
5. https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html

Abstract:

In this experiment, a dictionary data structure is implemented using C to create a contact book. This contact book allows users to add, view, update, and delete contact details. The dictionary data structure provides an efficient way to manage key-value pairs, enabling quick data retrieval and management. This experiment demonstrates practical applications of dictionaries in organizing, accessing, and modifying data based on keys.

Program:

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

#define MAX_CONTACTS 100

struct Contact {
    char name[50];
    char phone[15];
    char email[50];
};

struct Contact contact_book[MAX_CONTACTS];
int contact_count = 0;

void add_contact(const char *name, const char *phone, const char *email) {
    if (contact_count < MAX_CONTACTS) {
        strcpy(contact_book[contact_count].name, name);
        strcpy(contact_book[contact_count].phone, phone);
        strcpy(contact_book[contact_count].email, email);
        contact_count++;
        printf("Contact for %s added successfully.\n", name);
    } else {
        printf("Contact book is full!\n");
    }
}

void view_contact(const char *name) {
    int found = 0;
    for (int i = 0; i < contact_count; i++) {
        if (strcmp(contact_book[i].name, name) == 0) {
            printf("Name: %s\n", contact_book[i].name);
            printf("Phone: %s\n", contact_book[i].phone);
            printf("Email: %s\n", contact_book[i].email);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("No contact found for %s.\n", name);
    }
}

void update_contact(const char *name, const char *new_phone, const char
*new_email) {
```

```
    int found = 0;
    for (int i = 0; i < contact_count; i++) {
        if (strcmp(contact_book[i].name, name) == 0) {
            if (new_phone) {
                strcpy(contact_book[i].phone, new_phone);
            }
            if (new_email) {
                strcpy(contact_book[i].email, new_email);
            }
            printf("Contact for %s updated successfully.\n", name);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("No contact found for %s.\n", name);
    }
}

void delete_contact(const char *name) {
    int found = 0;
    for (int i = 0; i < contact_count; i++) {
        if (strcmp(contact_book[i].name, name) == 0) {
            for (int j = i; j < contact_count - 1; j++) {
                contact_book[j] = contact_book[j + 1];
            }
            contact_count--;
            printf("Contact for %s deleted successfully.\n", name);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("No contact found for %s.\n", name);
    }
}

int main() {
    add_contact("Aaryan", "9876543210", "aaryan@gmail.com");
    add_contact("Sharma", "9234014473", "sharma@gmail.com");

    printf("\nViewing contacts:\n");
    view_contact("Aaryan");
    view_contact("Sharma");

    printf("\nUpdating Aaryan's contact:\n");
    update_contact("Aaryan", "9421487343", NULL);
```

```
    printf("\nViewing Aaryan's updated contact:\n");
    view_contact("Aaryan");

    printf("\nDeleting Sharma's contact:\n");
    delete_contact("Sharma");

    printf("\nTrying to view Sharma's deleted contact:\n");
    view_contact("Sharma");
    return 0;
}
```

Output:

```
Contact for Aaryan added successfully.
Contact for Sharma added successfully.

Viewing contacts:
Name: Aaryan
Phone: 9876543210
Email: aaryan@gmail.com
Name: Sharma
Phone: 9234014473
Email: sharma@gmail.com

Updating Aaryan's contact:
Contact for Aaryan updated successfully.

Viewing Aaryan's updated contact:
Name: Aaryan
Phone: 9421487343
Email: aaryan@gmail.com

Deleting Sharma's contact:
Contact for Sharma deleted successfully.

Trying to view Sharma's deleted contact:
No contact found for Sharma.
```

Conclusion

In this experiment, we implemented a dictionary-based contact book in C, allowing us to add, view, update, and delete contacts. By using an array of structures, we demonstrated efficient storage and retrieval based on unique identifiers. This exercise provided insights into data organization, memory management, and the importance of dictionaries and

similar data structures for quick and organized data access.

PostLab Questions:

1) List the main functions or methods you implemented in your dictionary. What is the purpose of each?

1. add_contact(const char *name, const char *phone, const char *email)
    - Purpose: Adds a new contact to the contact book.
    - Functionality: This function accepts a contact's name, phone number, and email. It stores these details in the contact_book array if there's available space. It increments the contact_count to keep track of the number of contacts in the contact book.
2. view_contact(const char *name)
    - Purpose: Displays the details of a contact by name.
    - Functionality: This function searches for a contact by its name in the contact_book array. If found, it prints the contact's name, phone number, and email. If not, it notifies the user that no contact was found for the given name.
3. update_contact(const char *name, const char *new_phone, const char *new_email)
    - Purpose: Updates the phone number and/or email of an existing contact.
    - Functionality: This function searches for a contact by name. If found, it updates the contact's phone number and/or email with the new values provided. If the contact is not found, it notifies the user.
4. delete_contact(const char *name)
    - Purpose: Deletes a contact from the contact book.
    - Functionality: This function searches for a contact by name in the contact_book array. If found, it removes the contact by shifting all subsequent contacts one position up in the array to fill the gap. It then decrements the contact_count. If the contact is not found, it notifies the user.

These functions together enable the creation, retrieval, updating, and deletion of contacts, which are key operations for managing contact data in a simple dictionary-based application.