Submitted By,

Anwar M.R-ASI23CA015

Arjun S Nair-ASI23CA016

Arjun Suresh-ASI23CA017

Arpitha Bhandary-ASI23CA018

Ashna Thajudheen-ASI23CA019

Ashwin C.J-ASI23CA020

Data structures course project

Project Update

Topic :Mobile phone Book

**Project Overview:**

The \*\*Mobile Phone Book\*\* is an efficient system for managing contact information such as names and phone numbers. It allows users to add, update, search, delete, and display contacts. The system leverages dynamic memory allocation to ensure that the phone book can grow as needed. User interactions are managed through a simple menu interface, providing an easy and intuitive experience.

**Key Features:**

1. \*\*Contact Registration\*\*: Users can add new contacts with Name and Phone Number.

2. \*\*Sorting and Search\*\*: Contacts can be searched by name, with plans to include case-insensitive search functionality for more user-friendly results.

3. \*\*Contact Deletion\*\*: Users can delete contacts based on their names.

4. \*\*Dynamic Memory Management\*\*: The system dynamically reallocates memory when new contacts exceed the initial capacity.

5. \*\*User Interface\*\*: A menu-driven system allows users to choose operations such as adding, displaying, searching, and deleting contacts.

**Data Structures Used:**

- \*\*Dynamic Array\*\*: Contacts are stored in a dynamically allocated array, which expands as new contacts are added. The array of `struct Contact` ensures efficient storage and retrieval.

**Algorithms Used:**

1. \*\*Linear Search\*\*: This is used for searching contacts by name in the phone book.

2. \*\*Dynamic Memory Allocation\*\*: The `realloc` function is used to expand memory when the maximum number of contacts is reached.

**System Design:**

1. **Menu:**

- The system provides a menu for the user to select operations:

- Add a new contact.

- Display all contacts.

 - Search for a contact by name.

- Delete a contact by name.

- Exit the program.

1. **Contact Registration:**

1. The system checks if the current contact count has reached the maximum capacity.

2. If needed, it dynamically reallocates memory to add more contacts.

3. The user is prompted to enter the contact's name and phone number.

1. **Contact Search:**

- The system searches through the phone book for a contact by name using a case-sensitive comparison. Future updates will introduce case-insensitive searching for ease of use.

1. **Contact Deletion:**

- A contact is deleted by name. The system shifts all subsequent contacts in the array to maintain proper order.

1. **Dynamic Memory Management:**

- Memory is dynamically allocated and reallocated as new contacts are added. Initially, memory is allocated for a fixed number of contacts. If more space is needed, the `realloc` function increases the allocated memory in chunks.

* **Enhanced Features:**
  + 1. **Dynamic Memory Reallocation:**

- The system allows the phone book to grow without a predefined size limit by dynamically reallocating memory as contacts are added.

* + 1. **Input Validation:**

- Input validation is included for the phone number to ensure it is in the correct format, with only digits allowed. This ensures data integrity within the system.

* + 1. **Case-Insensitive Search (Upcoming):**

- Future updates will include a case-insensitive search feature, enhancing the user experience by allowing searches regardless of case differences in the input.

**Code Example:**

void addContact() {

// Check if more memory needs to be allocated for new contacts

if (contactCount == maxContacts) {

maxContacts += 5; // Increase memory by 5 contacts each time

phoneBook = realloc(phoneBook, maxContacts \* sizeof(struct Contact));

if (phoneBook == NULL) {

printf("Memory reallocation failed!\n");

return;

}

}

struct Contact newContact;

printf("Enter name: ");

scanf("%s", newContact.name);

// Input validation for phone number

printf("Enter phone number: ");

scanf("%s", newContact.phone);

// Ensure phone number contains only digits

int isValid = 1;

for (int i = 0; i < strlen(newContact.phone); i++) {

if (!isdigit(newContact.phone[i])) {

isValid = 0;

break;

}

}

if (!isValid) {

printf("Invalid phone number! Please enter only digits.\n");

return;

}

phoneBook[contactCount++] = newContact;

printf("Contact added successfully!\n");

}

This project update highlights the \*\*Mobile Phone Book's\*\* progress, focusing on dynamic memory management, efficient user interactions, and upcoming improvements like case-insensitive search and input validation. The system is evolving to be more robust and user-friendly, aiming to meet the users' contact management needs efficiently.

**1. Add Contact:**

* **Best Case**: O(1)
  + If there's space in the array (phone book) and no additional operations are needed (e.g., just adding the contact).
* **Worst Case**: O(1)
  + It only involves placing a new contact at the next available position, so no traversal of the list is required.
* **Average Case**: O(1)
  + The time complexity remains constant, as adding a contact does not depend on the number of existing contacts.

**2. Search Contact (Linear Search):**

* **Best Case**: O(1)
  + The contact being searched is found in the first position.
* **Worst Case**: O(n)
  + The contact is not found or is located at the last position, requiring a traversal of the entire list.
* **Average Case**: O(n)
  + On average, you’ll need to check half of the contacts before finding the target.

**3. Delete Contact (Linear Search and Shifting):**

* **Best Case**: O(1)
  + The contact to delete is the first element in the list, so only one element needs to be removed.
* **Worst Case**: O(n)
  + The contact is at the end of the list, requiring a linear search and shifting all elements after the contact one position forward.
* **Average Case**: O(n)
  + On average, you'd need to search half the list before finding the contact, and then shift half of the remaining elements.

**4. Display Contacts:**

* **Best Case**: O(n)
  + To display all contacts, you need to iterate through the list once.
* **Worst Case**: O(n)
  + The number of contacts does not affect the worst case; it's always linear.
* **Average Case**: O(n)
  + As all contacts need to be displayed, the time complexity remains O(n) regardless of the case.

| **Operation** | **Best Case** | **Worst Case** | **Average Case** |
| --- | --- | --- | --- |
| **Add Contact** | O(1) | O(1) | O(1) |
| **Search Contact** | O(1) | O(n) | O(n) |
| **Delete Contact** | O(1) | O(n) | O(n) |
| **Display Contacts** | O(n) | O(n) | O(n) |

