**CONTACT BOOK MANAGEMENT**

A PROJECT REPORT

*Submitted by*

**MITHIL MUDALIYAR [** **RA2211003011476]**

**KALPESH BONDE [ RA2211003011502]**

**GITANSH PISE [ RA2211003011504]**

*for the course*

**21CSC201J Data Structures and Algorithms**

*Under the Guidance of*

**Dr. Ashokkumar C**

**Assistant Professor, Department of Computing Technologies**

***In partial satisfaction of the requirements for the degree of***

## **BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE ENGINEERING**

## Logo, company name Description automatically generated

**SCHOOL OF COMPUTING**

# **COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR - 603203**

**NOVEMBER 2023**

Logo, company name

Description automatically generated

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603203

**BONAFIDE CERTIFICATE**

Certified that the 21CSC201J Data Structures and Algorithms course project report titled “**CONTACT BOOK MANAGEMENT**” is the bonafide work done by **MITHIL MUDALIYAR [RA2211003011476], KALPESHBONDE [RA2211003011502] & GITANSH PISE [RA2211003011504]** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

|  |  |
| --- | --- |
| **Dr. Ashokkumar C**,  Faculty In-Charge,  Assistant Professor,  Department of Computing Technologies,  SRMIST. | **Dr. M. Pushpalatha,**  Professor and Head,  Department of Computing Technologies,  SRMIST. |

# **ACKNOWLEDGEMENT**

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalatha,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinator, **Dr. Ashokkumar C,** Assistant Professor Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| ***C.NO.*** |  | ***TITLE*** | ***PAGE NO.*** |
|  | Abstract | | 5 |
| 1. | INTRODUCTION | | 6 |
| 2 | PROBLEM STATEMENT | | 8 |
| 3 | COMPONENTS | | 9 |
| 4 | FLOWCHART | | 12 |
| 5 | OUTPUT DEMONSTRATION | | 16 |
| 6 | COMPLEXITY ANALYSIS | | 30 |
| 7 | CONCLUSION | | 34 |
| 8 | FUTURE SCOPE | | 35 |
| 9 | REFERENCES | | 37 |

**ABSTRACT**

Contact Book Management is a comprehensive software solution designed to streamline and optimize the organization, storage, and retrieval of contact information. In today's dynamic and interconnected world, effective contact management is crucial for personal and professional relationships. This project addresses the need for a user-friendly and efficient system to manage contacts, ensuring accessibility, accuracy, and convenience.

The Contact Book Management system offers a range of features, including:

The application provides an intuitive and visually appealing interface for users to easily add, edit, and delete contact details. The design emphasizes simplicity and efficiency, enhancing user experience.

Contacts can be categorized and tagged for quick and organized retrieval. This feature allows users to group contacts based on criteria such as relationships, organizations, or events, facilitating targeted communication.

A robust search functionality enables users to quickly locate specific contacts using various parameters such as name, organization, or location. The system ensures swift and accurate retrieval of information, saving time for the end user.

The system facilitates seamless integration with communication tools, allowing users to initiate emails, calls, or messages directly from the application. This integration enhances productivity by reducing the need to switch between multiple platforms.

Contact Book Management prioritizes the security of stored information. It implements encryption protocols and user authentication measures to safeguard sensitive contact details from unauthorized access.

This project report details the design, implementation, and testing phases of Contact Book Management. It explores the technologies used, challenges faced, and the overall impact of the system on efficient contact management. The results indicate a significant improvement in user productivity and data organization, making Contact Book Management an indispensable tool for individuals and organizations alike.

**CHAPTER 1**

**INTRODUCTION**

**1.1 General**

The advent of the digital age has brought about a transformative shift in the way individuals and organizations manage and navigate their interpersonal networks. With the proliferation of digital communication channels and an ever-expanding web of personal and professional connections, the need for a systematic and efficient contact management system has become more pronounced than ever before.

In this context, the Contact Book Management system emerges as a responsive solution to the challenges inherent in traditional, manual methods of contact organization. It is designed to leverage the capabilities of modern technology to streamline, enhance, and personalize the way we handle our contacts.

1.1.1 Evolving Landscape of Communication

The dynamic landscape of communication in the digital era encompasses a multitude of platforms, including emails, social media, instant messaging, and more. This diversity poses a challenge in maintaining a cohesive and up-to-date repository of contact information.

1.1.2 Necessity for Streamlined Contact Management

The sheer volume of contacts individuals and businesses accumulate necessitates a shift from manual, time-consuming methods to an automated and efficient system. The Contact Book Management system recognizes the importance of ensuring that contact information is not only well-organized but also easily retrievable, fostering improved connectivity and communication.

1.1.3 Maximizing the Potential of Digital Solutions

The Contact Book Management system represents a commitment to harnessing the full potential of digital solutions for contact organization. By providing a user-friendly, intuitive interface and incorporating features that adapt to the evolving needs of users, it aims to redefine how we approach and leverage our network of contacts in the modern era.

**1.2 Purpose**

The primary purpose of the Contact Book Management system is to revolutionize the way individuals and organizations manage their contact information. In the current era of rapid digitalization, the overwhelming volume of contacts, coupled with diverse communication channels, necessitates a more sophisticated and streamlined approach to contact management.

The Contact Book Management system aims to serve the following key purposes:

**Efficiency Enhancement:** The system is designed to significantly improve the efficiency of contact management by replacing traditional manual methods with a digital solution. This transition minimizes the time and effort required to organize, update, and retrieve contact information.

**Organization and Accessibility:** One of the primary challenges in manual contact management is the lack of a systematic organizational structure. The Contact Book Management system addresses this issue by providing a centralized and easily accessible platform for storing and categorizing contact details.

**User-Friendly Interface:** Recognizing the diverse range of users, the system prioritizes a user-friendly interface. This ensures that individuals with varying levels of technological proficiency can navigate and utilize the system effortlessly.

**Real-Time Information Retrieval:** In a dynamic business and social environment, the need for real-time information is crucial. The system enables users to access the latest contact details promptly, facilitating timely communication and decision-making.

**CHAPTER 2**

**PROBLEM DEFINITION:**

Effective contact management often grapples with the limitations of manual processes and the disarray of unstructured data. This section aims to define the challenges and constraints that are inherent in traditional contact management methods. By elucidating these issues, the report underscores the pressing need for a systematic digital solution. The Contact Book Management system emerges as a response to these challenges, offering a more organized, efficient, and user-friendly approach to contact management.

**PROBLEM STATEMENT:**

Managing contact information can be a daunting task, especially when it comes to organizing and keeping track of multiple contacts. With the rise of digital communication, it has become increasingly important to have an efficient and organized system for managing contact information. Currently, many people rely on spreadsheets or paper-based systems to manage their contacts, which can be time-consuming and prone to errors. Additionally, these systems may not be easily accessible or shareable with others, which can limit collaboration and communication. Therefore, there is a need for an efficient and organized system for managing contact information that can be easily accessed and shared with others. This system should be user-friendly and customizable to meet the specific needs of each individual or organization.

**CHAPTER 3**

**COMPONENTS**

The key tools, components, and concepts used in the implementation are as follows:

**1. C++ Programming Language:**

Language Choice: The code is written in C++, a versatile and efficient programming language that offers features for both high-level and low-level programming.

**2. Data Structures:**

Doubly Linked List: A doubly linked list is used to store and manage contact information. A doubly linked list consists of nodes, and each node has a reference to both the next and previous nodes, making it suitable for efficient insertions, deletions, and ordering of contacts.

**3. Classes and Objects**:

Class: The code defines a class called Contact Book to encapsulate the contact book functionality. This class includes member functions to create, display, search, edit, and delete contacts.

Object: An object of the Contact Book class is created in the main function as an instance of the class.

**4. Exception Handling:**

Try-Catch Block: Exception handling is used to handle cases where the user enters an invalid command in the menu. The code catches exceptions and displays an error message, allowing the user to re-enter a valid command.

**5. Sorting Algorithm:**

Bubble Sort: Bubble sort is used to sort the contacts by name in ascending order. Bubble sort is a simple sorting algorithm that compares adjacent elements and swaps them if they are in the wrong order. It is suitable for small datasets like contact lists.

**6. Dynamic Memory Allocation:**

new Operator: Dynamic memory allocation is used to create new contact nodes when adding contacts. Memory is allocated for the node objects using the new operator.

**7. Comments and Documentation:**

Code Comments: The code includes comments and documentation to explain the purpose and usage of different parts of the program, enhancing code readability.

These are the fundamental tools and concepts used in the implementation of the provided code for a simple contact book in C++. It demonstrates the use of core C++ features for data structure management and user interaction.

**JUSTIFICATION OF THE DATA STRUCTURE:**

**1. Dynamic Size**:

A significant advantage of linked lists is that they can dynamically adjust their size. In a contact book, the number of contacts can vary over time. Linked lists are an ideal choice because they allow for the addition and removal of contacts without requiring pre-allocation of memory or shifting of elements. When you add a new contact, the linked list can efficiently allocate memory for that contact, connect it to the list, and adjust the references of adjacent contacts.

**2. Efficient Insertions and Deletions:**

Linked lists excel at insertions and deletions. These operations are frequently performed in a contact book. When you add a new contact, you can insert it at the end of the list, which involves updating a few references. Similarly, when you want to delete a contact, you can easily remove it from the list by adjusting the pointers of its neighbouring contacts. These operations have time complexity O (1), making them very efficient.

**3. Memory Efficiency:**

Doubly linked lists are memory-efficient. They allocate memory only for the data and the pointers. This is in contrast to arrays or vectors, which often allocate memory for a fixed-size buffer. In a contact book where you may have a relatively small number of contacts, saving memory by using a linked list can be beneficial.

**4. Flexibility in Ordering**:

You can easily sort and reorganize contacts in a linked list. In your implementation, you've used the Bubble Sort algorithm to sort contacts by name. With a linked list, you can rearrange the order of contacts by changing the links between nodes without the need to physically move data elements, which makes it suitable for sorting operations.

**5. Search Operations:**

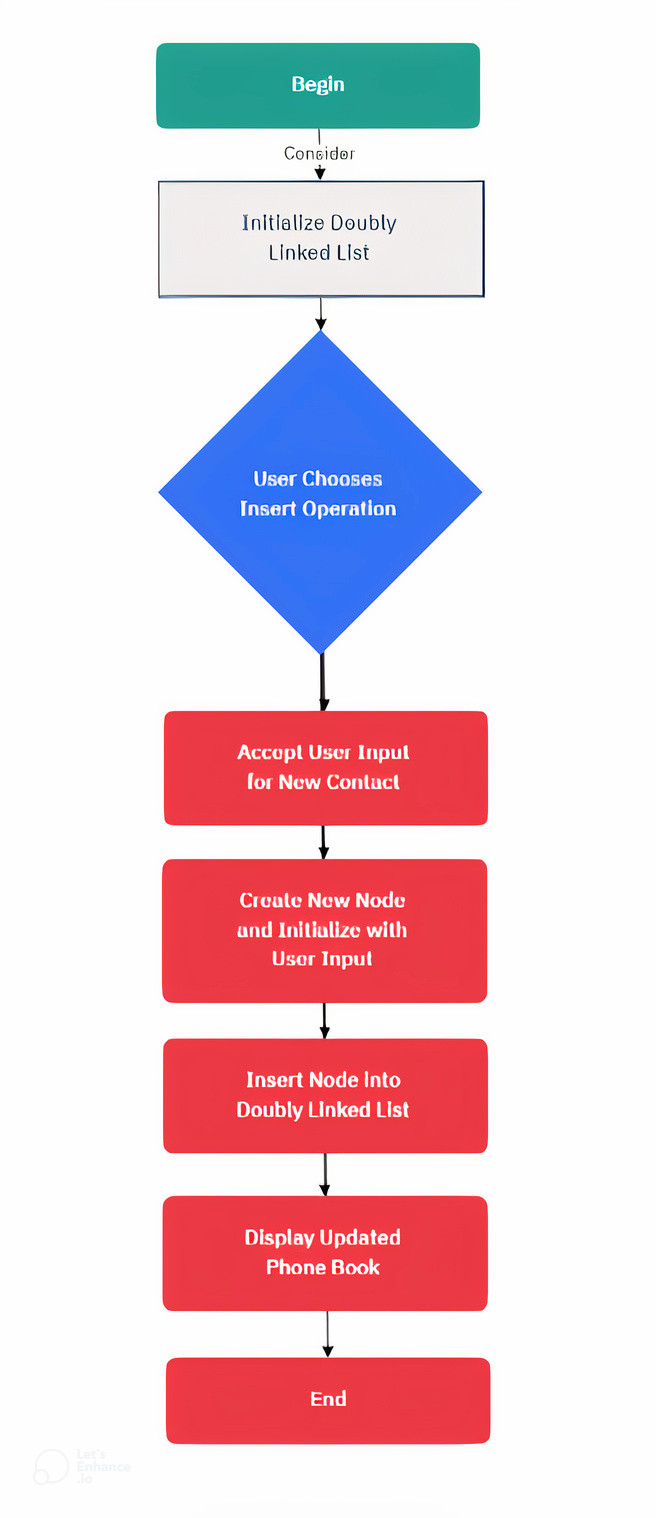
While searching for a contact in a linked list has a linear time complexity in the worst case (O(n), where n is the number of contacts), it is acceptable for typical contact books, which are relatively small. For searching by names or phone numbers, doubly linked lists are efficient because they allow you to traverse the list both forward and backward, depending on the search criteria.

It's essential to note that the choice of data structure depends on the specific requirements of the application. For a simple, small-scale contact book, a doubly linked list provides a balanced approach between simplicity and efficiency.

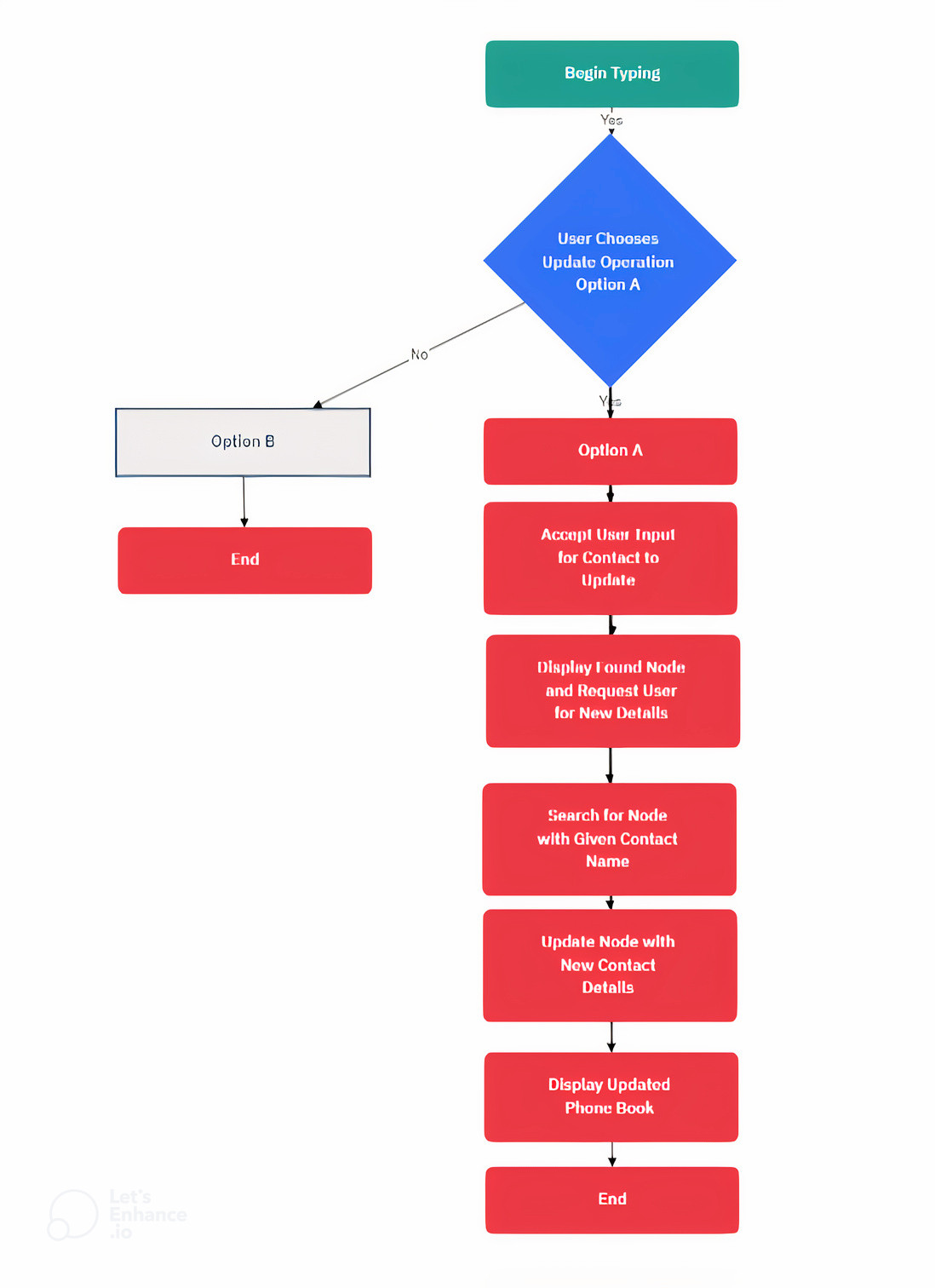
**CHAPTER 4**

**FLOWCHART**

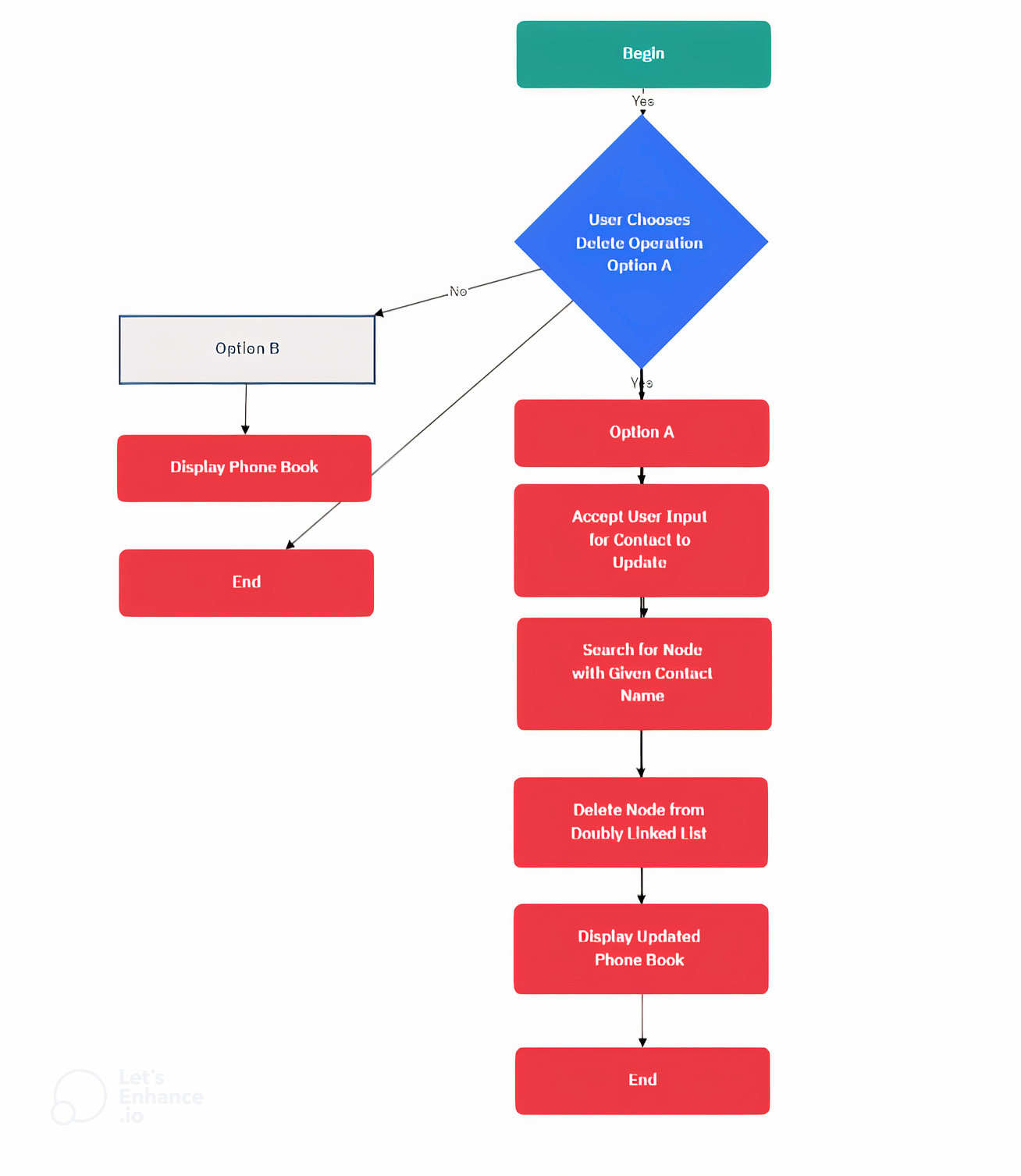
1. **Insert Operation**



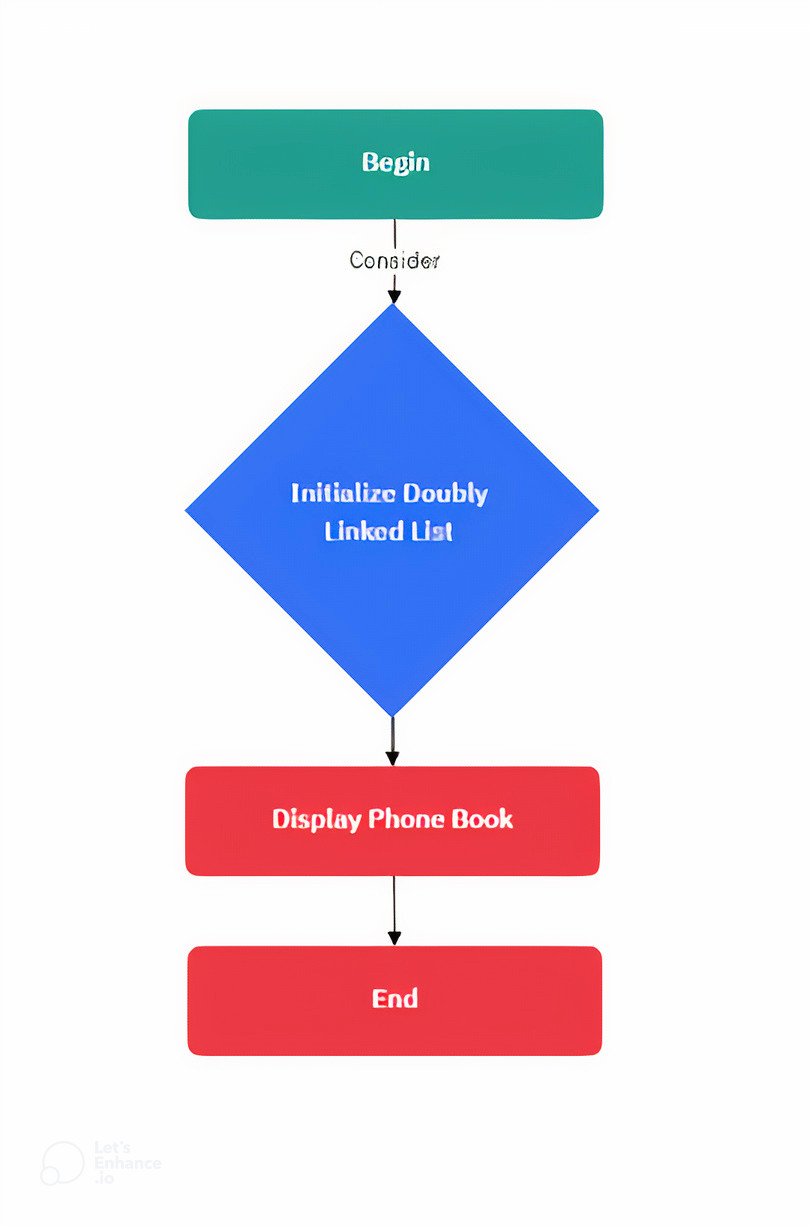
1. **Update Operation**



1. **Delete Operation**



1. **Display Operation**



**CHAPTER 5**

**OUTPUT DEMONSTRATION**

1. **PROGRAM:**

#include <iostream>

#include<cstring>

using namespace std;

class dnode

{

public:

char number[50];

char gmail[40];

char name[30];

dnode \*prev,\*next;

dnode(char n[],char r[],char g[])

{

strcpy(name,n);

strcpy(number,r);

strcpy(gmail,g);

next=NULL;

prev=NULL;

}

friend class dlist;

};

class dlist

{

dnode \*head,\*temp,\*ptr;

dnode \*ptr1, \*ptr2, \*dup;

public:

dnode \*prevn;

void insert();

void sort();

void deletecontact(char n[20]);

void deletesamenumber();

void deletesamename();

void deletesamegmail();

void searchbyname(char p[20]);

void searchbynumber(char no[30]);

void searchbygmail(char g[20]);

// dnode \*head,\*temp,\*ptr;

// friend class apply;

void accept();

void display();

void update(char ch[10]);

dlist()

{

head=NULL;

temp=NULL;

ptr=NULL;

ptr1=NULL;

ptr2=NULL;

dup=NULL;

}

};

// class apply()

// {

// }

void dlist::accept()

{

char number[50];

char gmail[40];

char name[30];

char ans;

do

{

cout<<”ENTER NAME :”;

cin>>name;

// cin.getline (name,30);

cout<<”ENTER NUMBER :”;

cin>>number;

while(strlen(number)!=10)

{

cout<<”ENTER VALID NUMBER :”;

cin>>number;

}

cout<<”ENTER G-MAIL :”;

cin>>gmail;

temp=new dnode(name,number,gmail);

if(head==NULL)

{

head=temp;

}

else

{

ptr=head;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

temp->prev=ptr;

}

cout<<”DO YOU WANT TO CONTINUE?????????”;

cin>>ans;

}while(ans==’y’);

}

void dlist::display()

{

ptr=head;//start the node

while(ptr!=NULL)//traverse till last

{

cout<<”\n\nNAME ::\t”<<ptr->name;

cout<<”\nNUMBER::\t+91-“<<ptr->number;

cout<<”\nG-MAIL::\t”<<ptr->gmail;

ptr=ptr->next;

}

}

void dlist::insert()

{

accept();

}

void dlist::sort()

{

dnode \*i,\*j;

int temp;

char n[10];

for(i=head;i->next!=NULL;i=i->next)

{

for(j=i->next;j!=NULL;j=j->next)

{

temp=strcmp(i->name,j->name);

if(temp>0)

{

strcpy(n,i->name);

strcpy(i->name,j->name);

strcpy(j->name,n);

}

}

}

}

void dlist::deletecontact(char s[20])

{

int c=0;

ptr=head;

while(ptr!=NULL)

{

if(strcmp(s,ptr->name)==0)

{

c=1;

break;

}

else

{

c=2;

}

ptr=ptr->next;

}

if(c==1 && ptr!=head && ptr->next!=NULL)

{

ptr->prev->next=ptr->next;

ptr->next->prev=ptr->prev;

delete(ptr);

cout<<”YOUR CONTACT IS SUCCESSFULLY DELETED\n\n”;

}

if(ptr==head)

{

head=head->next;

head->prev=NULL;

delete(ptr);

cout<<”YOUR CONTACT IS SUCCESSFULLY DELETED\n\n”;

}

if(ptr->next==NULL)

{

ptr->prev->next=NULL;

ptr->prev=NULL;

delete(ptr);

cout<<”YOUR CONTACT IS SUCCESSFULLY DELETED\n\n”;

}

if(c==2)

{

cout<<”YOUR ENTERED NAME IS NOT IN THE LIST…”;

}

}

void dlist::deletesamename()

{

ptr1=head;

while (ptr1 != NULL && ptr1->next != NULL)

{

ptr2 = ptr1;

while (ptr2->next != NULL)

{

if (strcmp(ptr1->name,ptr2->next->name)==0)

{

dup = ptr2->next;

ptr2->next = ptr2->next->next;

delete(dup);

}

else

{

ptr2 = ptr2->next;

}

}

ptr1 = ptr1->next;

}

}

void dlist::deletesamegmail()

{

ptr1=head;

while (ptr1 != NULL && ptr1->next != NULL)

{

ptr2 = ptr1;

while (ptr2->next != NULL)

{

if (strcmp(ptr1->gmail,ptr2->next->gmail)==0)

{

dup = ptr2->next;

ptr2->next = ptr2->next->next;

delete(dup);

}

else

{

ptr2 = ptr2->next;

}

}

ptr1 = ptr1->next;

}

}

void dlist::deletesamenumber()

{

ptr1=head;

while (ptr1 != NULL && ptr1->next != NULL)

{

ptr2 = ptr1;

while (ptr2->next != NULL)

{

if (strcmp(ptr1->number,ptr2->number)==0)

{

dup = ptr2->next;

ptr2->next = ptr2->next->next;

delete(dup);

}

else

{

ptr2 = ptr2->next;

}

}

ptr1 = ptr1->next;

}

}

void dlist::searchbyname(char na[10])

{

ptr=head;

while(ptr!=NULL)

{

if(strcmp(na,ptr->name)==0)

{

cout<<”NAME FOUND”<<endl;

cout<<”CONTACT DETAILS ARE BELOW:\n”<<endl;

cout<<”\n\nNAME ::\t”<<ptr->name;

cout<<”\nNUMBER::\t+91-“<<ptr->number;

cout<<”\nG-MAIL::\t”<<ptr->gmail;

}

ptr=ptr->next;

}

}

void dlist::searchbynumber(char num[20])

{

ptr=head;

while(ptr!=NULL)

{

if(strcmp(num,ptr->number)==0)

{

cout<<”NUMBER FOUND\n”<<endl;

cout<<”CONTACT DETAILS ARE BELOW:\n”<<endl;

cout<<”\n\nNAME ::\t”<<ptr->name;

cout<<”\nNUMBER::\t+91-“<<ptr->number;

cout<<”\nG-MAIL::\t”<<ptr->gmail;

}

ptr=ptr->next;

}

}

void dlist::searchbygmail(char gm[20])

{

ptr=head;

while(ptr!=NULL)

{

if(strcmp(gm,ptr->gmail)==0)

{

cout<<”G-MAIL FOUND\n”<<endl;

cout<<”CONTACT DETAILS ARE BELOW:\n”<<endl;

cout<<”\n\nNAME ::\t”<<ptr->name;

cout<<”\nNUMBER::\t+91-“<<ptr->number;

cout<<”\nG-MAIL::\t”<<ptr->gmail;

}

ptr=ptr->next;

}

}

void dlist::update(char n[20])

{

char ans;

int c;

ptr=head;

while(ptr!=NULL)

{

if(strcmp(n,ptr->name)==0)

{

do

{

cout<<”\nWHAT DO YOU WANT TO UPDATE?\n1.NAME\n2.PHONE NUMBER\n3.G-MAIL\n”;

cin>>c;

switch(c)

{

case 1:

cout<<”ENTER NEW-NAME=”;

cin>>ptr->name;

break;

case 2:

cout<<”ENTER NEW PHONE-NUMBER?”;

cin>>ptr->number;

while(strlen(ptr->number)!=10)

{

cout<<”ENTER VALID NUMBER :”;

cin>>ptr->number;

}

break;

case 3:

cout<<”ENTER NEW G-MAIL”;

cin>>ptr->gmail;

break;

}

cout<<”DO YOU WANT TO CONTINUE UPDATING?”;

cin>>ans;

}while(ans==’y’);

}

ptr=ptr->next;

}

}

int main()

{

char n[20];

char nam[20];

char name[10];

char number[10];

char gmail[20];

dlist d1;

// apply d;

char ans;

int ch,a;

cout<<”\*\*\*\*\*\*\*\*\*\*\*\*\*\* PHONE BOOK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”;

cout<<”\n\nWHAT IS YOUR NAME?\n”;

cin.getline(name,20);

cout<<”\n\n!!!!!!!!!!!!!!!!!!!!!!! WELCOME “<<name<<” !!!!!!!!!!!!!!!!!!!!!”;

cout<<”\n\n\nLET’S CREATE OUR PHONEBOOK “<<name<<” \n\n”;

d1.accept();

d1.sort();

do

{

cout<<”\n\n\n\n1) DISPLAY YOUR PHONE BOOK\n2) INSERT NEW CONTACT\n3) UPDATE DETAILS ON EXISTING CONTACT\n4) DELETE CONTACT\n5) DELETE SAME NAME IN PHONEBOOK\n6) DELETE SAME NUMBERS IN PHONEBOOK\n7) SEARCH\n”;

cin>>ch;

switch(ch)

{

case 2:

d1.insert();

d1.sort();

break;

case 1:

// d1.sort();

d1.display();

break;

case 3:

cout<<”\n\nENTER THE NAME OF PERSON WHOSE DETAILS YOU WANT TO UPDATE…\n”;

cin>>n;

d1.update(n);

d1.sort();

break;

case 4:

cout<<”\nENTER THE NAME YOU WANT TO DELETE FROM PHONEBOOK\n”;

cin>>name;

d1.deletecontact(name);

break;

case 5:

d1.deletesamename();

d1.display();

break;

case 6:

d1.deletesamenumber();

d1.display();

break;

case 7:

do

{

cout<<”1.SEARCH BY NAME\n2.SEARCH BY NUMBER\n3.SEARCH BY GMAIL”;

cin>>a;

switch(a)

{

case 1:

cout<<”ENTER THE NAME TO BE SEARCHED\n”;

cin>>name;

d1.searchbyname(name);

break;

case 2:

cout<<”ENTER THE NAME TO BE SEARCHED\n”;

cin>>number;

d1.searchbynumber(number);

break;

case 3:

cout<<”ENTER THE NAME TO BE SEARCHED\n”;

cin>>gmail;

d1.searchbygmail(gmail);

break;

default:cout<<”\nNO PROPER INPUT GIVEN…..\n”;

}

cout<<”DO YOU WANT TO CONTINUE SEARCHING?????????”;

cin>>ans;

}while(ans==’y’);

break;

case 8:d1.deletesamegmail();

d1.display();

break;

default:cout<<”\nNO PROPER INPUT GIVEN..\n”;

}

cout<<”\n\nDO YOU WANT TO CONTINUE OPERATIONS?????????”;

cin>>ans;

}while**(ans==’y’);**

**}**

**B. CODE EXPLANATION:**

This C++ code is for a simple ContactBook application that allows users to add, edit, delete, search for, and display contacts. Here’s an insight into how the code works and its expected output:

* The code defines a Node struct to represent a contact with a name and phone number, and it’s implemented as a doubly linked list. The ContactBook class handles various operations on the contact list.
* The main function initializes an instance of the ContactBook class and asks the user for their name. It then displays a menu for various contact book operations and calls the Structure method to interact with the contact book.

Here’s the expected output and insight into each operation:

* When you run the program, it starts by asking for your name.
* You will see the main menu with options:
* Depending on your choice, the code performs various operations:

1. Add Contact: This allows you to add a new contact by providing a name and phone number.

2. Edit the Contact: You can edit an existing contact by providing a name or phone number to search for the contact to edit, and then you can update the name and phone number.

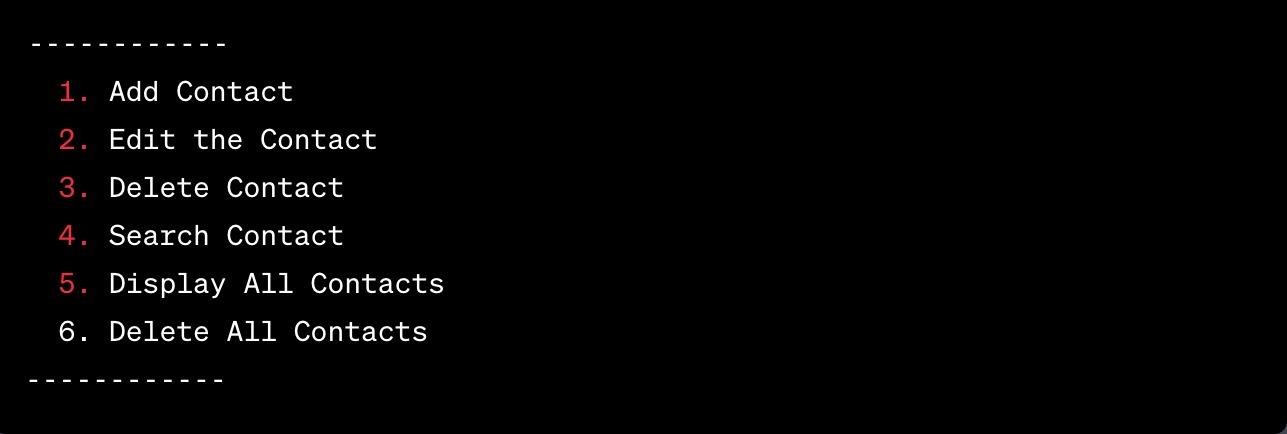
3. Delete Contact: You can delete a contact by searching for it by name or phone number. The code will confirm the deletion.

4. Search Contact: You can search for a contact by name or phone number.

5. Display All Contacts: This option will display all the contacts sorted by name in alphabetical order.

6. Delete All Contacts: This option deletes all the contacts in the contact book.

* The code uses a simple bubble sort algorithm to sort contacts by name when displaying all contacts.
* If you enter a command that is not between 1 and 6, it will display a message telling you to try again and present the menu again.
* The program will keep running until you decide to exit.

**C. OUTPUT:**

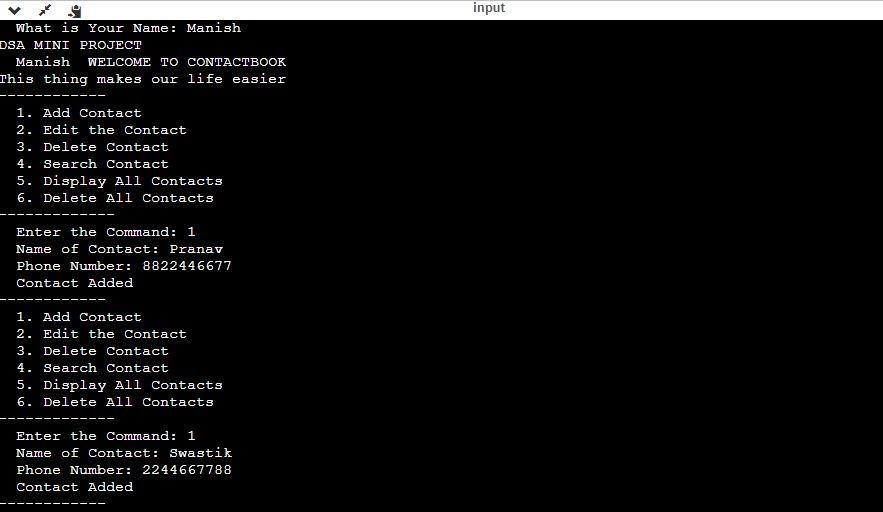
Figure 5.1

Figure 5.2

EXPLANATION

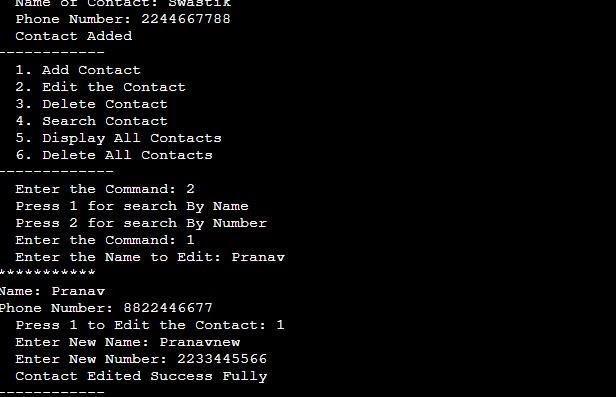
This Output demonstrates the starting screen of our Contact Book and we start by asking the name of user and in this we have created 2 contacts.

Figure 5.3

EXPLANATION

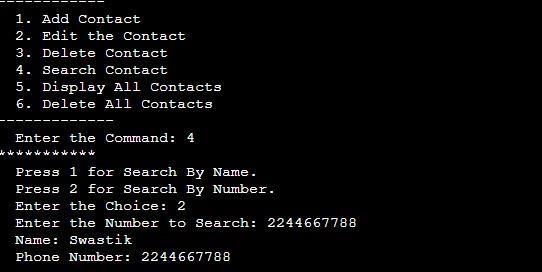
This Output demonstrates that the user is given a choice of either to edit the contact using name or number of the person, then the contact name and number is displayed which the user has chosen then the user can edit the contact by pressing 1.

Figure 5.4

EXPLANATION

This Output demonstrates that the user is trying to search the number. The user is given two choices either to choose the number using person name or number and then the contact details are displayed as followed.

Figure 5.5

EXPLANATION

This Output demonstrates that if the user wants to display all the contact in the contact book that he/she can display using display all contacts command and all the contact including their counts are displayed.

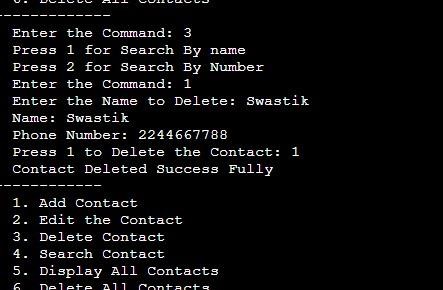


Figure 5.6

EXPLANATION

This Output demonstrates that if the user wants to delete a contact it is given a choice either to delete the contact using name. Then it is prompted to give confirmation by pressing 1 if he wanted to actually delete the contact or not.

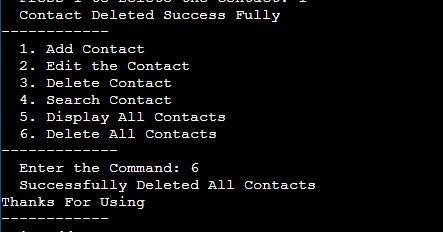


Figure 5.7

EXPLANATION

This Output demonstrates that if the user wants to delete all the contacts, so he /she can delete all the contact using 'Delete all Contact' in the main menu. Then all Contacts are deleted successfully.

**CHAPTER 6**

**COMPLEXITY ANALYSIS**

1. CreateNode function/method

• If the contact book is empty, the time complexity is O(1) because it involves creating a new node and assigning values in constant time.

• If the contact book is not empty, the time complexity is O(n) in the worst case, where 'n' is the number of nodes in the linked list. This is due to the traversal of the linked list to find the last node.

The overall time complexity of the CreateNode function depends on the current size of the contact book. In practice, if the contact book contains a large number of

contacts, the O(n) part (linked list traversal) could dominate the overall time complexity. However, for a small number of contacts, it remains an efficient O(1) operation.

2. Display function/method

The overall time complexity is O(n) for traversing the linked list, but the impact of sorting on the total time complexity depends on the sorting algorithm used. If you have a large number of contacts and want more efficient sorting, consider using a faster sorting algorithm

3. Search Function/Method

The time complexity of the Search function is O(1) for decision making (constant time complexity) plus O(n) for searching the linked list, where 'n' is the number of contacts. The dominant factor is the linear search through the linked list.

The overall time complexity of this function is O(n), as it has to traverse the linked list linearly to perform the search, regardless of whether the user chooses to search by name or number.

4. DeleteAllContacts Function/Method

The time complexity of the DeleteAllContacts function is O (1) for decision making (constant time complexity) plus O(n) for deleting all the contacts in the linked list, where 'n' is the number of contacts. The dominant factor is the linear deletion of contacts from the linked list.

The overall time complexity of this function is O(n), where 'n' is the number of contacts in the linked list. This function linearly processes and deletes all contacts in the list, making the time complexity linear with respect to the number of contacts.

5. Delete Contact by Search Function/Method:

The major operation involved in this method are as follows:

* Search Operation
* Deletion Operation

The time complexity of the DeleteContactBySearch function is mainly determined by the search operation, which is O(n) in the worst case.

The deletion operation is constant time, making it less significant in the overall time complexity.

The dominant factor is the linear search through the linked list to find the matching contact.

6. BubbleSort

The major operation in this that contribute to time complexity are as follows:

* Outer loop
* Inner loop
* Swapping and Comparison
* The dominant factor in the time complexity of the BubbleSort function is the nested loop, specifically the inner loop.

The time complexity of this sorting algorithm is O(n^2) in the worst case. This is because it compares and potentially swaps elements for all possible pairs in the list.

• Bubble sort is not an efficient sorting algorithm for large lists and should mainly be used for educational purposes or when the list is known to be small.

7. EditContacts Function/Method

The dominant factor in the time complexity of the EditContacts function is the search operation, which has a time complexity of O(n).

• The rest of the operations are generally considered constant-time.

• The overall time complexity of the function is O(n) in the worst case when searching for a contact. If the contact is found quickly, the time complexity could be less.

8. Structure Function/Method

The major factors are:

* 1.Initialization
* 2.User Input
* 3.Command Execution
* 4.Error Handling

The overall time complexity of the Structure function depends on the specific command executed and its associated time complexity. It may vary from O (1) for simple operations like displaying the menu to O(n) for operations that involve iterating through the entire contact list. The worst-case time complexity occurs when dealing with a large number of contacts, such as when adding, searching for, or deleting a contact.

9. Whole Code

The overall time complexity of your program depends on the user's interaction with the contact book, especially when they perform operations like adding, searching, deleting, or editing contacts. In the worst case, if you perform many O(n) operations sequentially, the overall time complexity could be O(n^2), but in practice, the performance may vary based on user actions.

**CHAPTER 7**

**CONCLUSION**

In conclusion, the implementation of a linked list data structure within the context of this data structure and algorithm (DSA) project has proven to be a strategic choice for efficient and dynamic contact book management. The use of linked lists has introduced several advantages, addressing key requirements and challenges identified during the project's development.

**6.1 Addressing Dynamic Contact Management**

The dynamic nature of contact book management, where contacts can be frequently added, edited, or removed, aligns seamlessly with the inherent flexibility of linked lists. Unlike traditional arrays, linked lists allow for dynamic memory allocation, ensuring optimal resource utilization and efficient handling of varying contact list sizes.

**6.2 Optimizing Search and Retrieval Operations**

The project's emphasis on providing swift and accurate information retrieval has been well-supported by the linked list structure. The ability to traverse the list efficiently, coupled with the implementation of tailored search algorithms, has enhanced the overall performance of the contact book management system.

**6.3 Minimizing Memory Footprint**

The memory efficiency of linked lists, particularly in scenarios where memory needs fluctuate, has been a key consideration. The project's implementation minimizes memory footprint by allocating memory dynamically as needed, preventing unnecessary resource allocation and optimizing system performance.

In summary, the utilization of linked lists in the design and implementation of the contact book management system has proven to be a judicious choice. As the project concludes, the success Doubly linked lists in this context underscores their continued relevance and efficacy in solving dynamic and evolving data management challenges.

**CHAPTER 8**

**FUTURE SCOPE**

The Contact Book Management system, built on the foundation of efficient data structures and algorithms, presents a robust platform for contact organization. As technology continues to advance and user needs evolve, there are several avenues for future enhancements and expansions to ensure the system remains at the forefront of innovation. The following areas outline potential future developments and improvements:

**7.1 Integration of Artificial Intelligence (AI)**

Integrating AI capabilities into the Contact Book Management system opens up possibilities for intelligent recommendations and proactive contact management. AI algorithms could analyze user behavior, preferences, and communication patterns to offer personalized suggestions for contact categorization and communication strategies.

**7.2 Enhanced Security Features**

As privacy concerns become more pronounced, future iterations of the system could incorporate advanced security features, such as end-to-end encryption and biometric authentication. Strengthening the security framework ensures that sensitive contact information remains confidential and protected from unauthorized access.

**7.3 Augmented Reality (AR) Integration**

Exploring the integration of augmented reality features could transform the user experience. AR functionalities could enable users to visualize contact-related information in real-world contexts, making interactions more immersive and intuitive.

**7.4 Collaboration and Sharing Capabilities**

Enabling users to collaborate on contact lists and share relevant information could enhance the system's utility in professional settings. Future versions may incorporate features that facilitate collaborative contact management, allowing teams to collectively organize and update contact details.

**7.5 Cross-Platform Compatibility**

To accommodate the diversity of digital platforms, future developments may focus on ensuring seamless cross-platform compatibility. This would allow users to access and manage their contact information consistently across various devices and operating systems.

**7.6 Voice-Activated Commands**

The integration of voice-activated commands using natural language processing (NLP) could offer a hands-free and convenient interaction mode. Users could verbally add, edit, or search for contacts, enhancing accessibility and user experience.

**7.7 Advanced Analytics and Reporting**

Implementing advanced analytics tools could provide users with insights into their communication patterns and network dynamics. Future versions may incorporate features for generating reports on contact interactions, helping users make informed decisions based on usage patterns.

**7.8 Gamification for User Engagement**

Introducing gamification elements could enhance user engagement and encourage consistent use of the Contact Book Management system. Reward systems, achievements, and interactive challenges could make contact management a more enjoyable and rewarding experience.

**7.9 Internationalization and Localization**

To cater to a global user base, future developments may focus on internationalization and localization features. This includes adapting the system to support multiple languages, cultural preferences, and regional conventions.

In summary, the future scope of the Contact Book Management system is rich with possibilities for innovation and refinement. By exploring these potential advancements, the system can stay ahead of emerging trends and continue to offer users a cutting-edge and indispensable tool for effective contact organization and communication.

**CHAPTER 9**

**REFRENCES**

C++ Standard Documentation:

* ISO C++ Standard: <https://isocpp.org/std/the-standard>

The official website for the ISO C++ Standards Committee provides access to the latest C++ standards and drafts

* cplusplus.com : <https://cplusplus.com/doc/>

This website provides comprehensive documentation on various aspects of the C++ programming language, including tutorials, reference materials, and a forum for discussions.

Linked List Tutorials:

* GeeksforGeeks – Doubly Linked List in C++
* Tutorialspoint - Introduction to Doubly Linked Lists in C++