**PROJECT REPORT**

# *CONTACT BOOK MANAGEMENT*

***21CSC201J – DATA STRUCTURES & ALGORITHM***

**Submitted By:**

**PRANAV SINGH -RA2211003010540**
**MANISH TIWARI - RA2211003010546**
**SWASTIK RANA -RA2211003010564**

**BACHELOR OF TECHNOLOGY**
**in**
**COMPUTER SCIENCE ENGINEERING**

**SCHOOL OF COMPUTING**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR - 603203**

**OCTOBER 2023**

## TABLE OF CONTENTS

## 1)a. 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.

## 1)b. OBJECTIVE OF THE PROGRAM:

The objective of contact book management in C++ is to provide a structured way of storing and organizing contact information for individuals or organizations. This can include names, phone numbers and total number of contacts. The goal is to make it easy to access and manage this information, whether it's for personal or professional use.

## 2) DATA STRUCTURE USED:

Doubly linked lists are a common data structure used in computer science. They are similar to singly linked lists, but with an additional pointer that points to the previous node. This allows for efficient insertion and deletion of nodes at the beginning or end of the list. In terms of performance, doubly linked lists generally outperform singly linked lists, especially when it comes to insertion and deletion at the beginning or end of the list. However, they can be slower than arrays for certain operations, such as searching or sorting.

## 3)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 neighboring 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.

## 4)TOOLS USED:

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

**5)a. SOURCE CODE:**

```
#include<iostream>
using namespace std;
// Define a struct named Node to represent a contact.
struct Node
{
        string name;   // Member variable to store the contact's name
        long long int phone_number; // Member variable to store the contact's phone number.
        Node *next;// Pointer to the next contact (linked list).
        Node *prev;// Pointer to the previous contact (doubly linked list).
};
class ContactBook
{
        Node *head;
        string x;
        long long int y;
        public:
   //constructor to intialise the values.
                ContactBook()
                {
                        head=NULL;
                        x="";
                        y=0;
                }
        void CreateNode()
           {// Create a new node when the contact book is empty.
                if(head==NULL)
                {
                Node *newer= new Node;
                   cout<<"  Name of Contact: ";
                     cin>>x;
newer->name=x;
                    cout<<"  Phone Number: ";
                    cin>>y;
                       newer->phone_number=y;
                       newer->next=NULL;
                       head=newer;
                       cout<<"  Contact Added"<<endl;
                    }
                    else
    // Create a new node when the contact book is not empty.
                    {
                       Node *newer= new Node;
                    cout<<"  Name of Contact: ";
                       cin>>x;
                       newer->name=x;
                    cout<<"  Phone Number: ";
                    cin>>y;
                       newer->phone_number=y;
                       newer->next=NULL;
                    //using a temporary pointer to transver the ll and add new node.
                          Node *temp=head;
                          while(temp->next!=NULL)
                          {
                                   temp=temp->next;
                          }
                          temp->next=newer;
                          newer->prev=temp;
                          cout<<"  Contact Added"<<endl;
                          }
                 }
```

```
void Display()
 {
            Node *temp=head;// Start at the beginning of the contact book
            int count=0;// we have Used this to show the no. of contact in the ll.
            // If the contact book is empty
            if(temp==NULL)
            {
                        cout<<"  No Contacts... Please Add Some Contacts"<<endl;
            }
            else// If there are contacts in the book, sort them and display the list
            {
                        BubbleSort();// Sort the contacts
                        cout<<"  Name: "<<"      Number: \n"<<endl;
            while(temp!=NULL)
            {
               count++;
                        cout<<"  "<<temp->name;
                        cout<<"          "<<temp->phone_number<<endl;
                        temp=temp->next;
            }
            cout<<"  Total contacts: "<<count<<endl;
            }
 }
 void Search()
 {
            bool check=false;
            Node *temp=head;
            cout<<"**********"<<endl;
            cout<<"  Press 1 for Search By Name."<<endl;
            cout<<"  Press 2 for Search By Number."<<endl;
```

```
int choice;
cout<<"  Enter the Choice: ";
cin>>choice;
if(choice==1 && temp!=NULL)
{
        cout<<"  Enter the Name to Search: ";
        cin>>x;// Looping through the contact book to search by name.
        while(temp!=NULL)
        {
        if(temp->name==x)
        {
                cout<<"  Name: "<<temp->name<<endl;
                cout<<"  Phone Number:"<<temp->phone_number<<endl;
                check=true;
                break;
        }
        temp=temp->next;
}
if(check==false)
{
        cout<<"  Name Not Found"<<endl;
                }
}
else if(choice==2)
{
        cout<<"  Enter the Number to Search: ";
        cin>>y;
                // Loop through the contact book to search by phone number.
        while(temp!=NULL)
        {
```

```
                    if(temp->phone_number==y)
                    {
                          cout<<"  Name: "<<temp->name<<endl;
                          cout<<"  Phone Number: "<<temp->phone_number<<endl;
                          check=true;
                          break;
                    }
                        temp=temp->next;
              }
                          if(check==false)
                          {
                                cout<<"  Number Not Found"<<endl;
                          }
              }
          }
          void DeleteAllContacts()
          {
              Node *temp=head;// Initialize a temporary pointer
              Node *temp2;// Another temporary pointer for deletion.
              if(head==NULL)
              {
// If the contact book is already empty
                    cout<<"  Already Contact Book is Empty"<<endl;
                    cout<<"***********"<<endl;
                    }
                    else// If the contact book is not empty
                    {
                          while(temp!=NULL)
                    {
                              temp2=temp;// Store the current contact in temp2.
                              temp=temp->next;// Move to the next contact.
                              delete temp2;// Delete the contact stored in temp2.
                    }
                    head=NULL;
                    cout<<"  Successfully Deleted All Contacts"<<endl;
          cout<<"Thanks For Using "<<endl;
          }
}
void DeleteContactBySearch()
{

Node *temp=head;
cout<<"  Press 1 for Search By name"<<endl;
cout<<"  Press 2 for Search By Number"<<endl;
int command;
cout<<"  Enter the Command: ";
cin>>command;
if(command==1)
{
          bool Dcheck=false;
          cout<<"  Enter the Name to Delete: ";
          cin>>x;
          while(temp!=NULL)
          {
          if(temp->name==x)
```

```
                {
                    cout<<"  Name: "<<temp->name<<endl;
                    cout<<"  Phone Number: "<<temp->phone_number<<endl;
                    Dcheck=true;
                    break;
                }
                    temp=temp->next;
            }
            if(Dcheck==true)
            {
                int command;
                cout<<"  Press 1 to Delete the Contact: ";
                cin>>command;
                if((command==1)&(temp==head))//if contact is the starting node
                {
                    Node *temp1;
                    temp1=temp;
                    temp=temp->next;
                    delete temp1;

                    temp->prev=NULL;
                    head=temp;
                    cout<<"  Contact Deleted Success Fully"<<endl;
                    }
                    else if((command==1)&(temp->next==NULL))//if contact is
the last node
                    {
                        temp->prev->next=NULL;
                        delete temp;
                        cout<<"  Contact Deleted Success Fully"<<endl;
                        }
                        else if(command==1)//if contact is the middle node
                        {
                                Node *temp1;
                                temp1=temp;
                                temp->prev->next=temp1->next;
                                temp->next->prev=temp1->prev;
                                delete temp1;
                                cout<<"  Contact Deleted Success Fully"<<endl;
                        }
                        else
                        {
                                cout<<"  You Enter Wrong Command"<<endl;
                        }
                }
                else if(Dcheck==false)
                        {
                                cout<<"  Contact of This Name Not Found."<<endl;//if
there is no contact by this name.
                     }
                }
                else if(command==2)
                {
                        bool Dcheck=false;
                        cout<<"  Enter the Number to Delete: ";
                        cin>>y;
                        while(temp!=NULL)
                        {
                        if(temp->phone_number==y)
                        {
```

```
                    cout<<"name: "<<temp->name<<endl;
                    cout<<"Phone Number: "<<temp->phone_number<<endl;
                    Dcheck=true;
                    break;
                }
                  temp=temp->next;
            }
            if(Dcheck==true)
            {
                int command;
                cout<<"  Press 1 to Delete the Contact: ";
              cin>>command;
            if((command==1)&(temp==head))//if contact is the starting node
                {
                    Node *temp1;
                    temp1=temp;
                temp=temp->next;
                  delete temp1;
                temp->prev=NULL;
                    head=temp;
                    cout<<"  Contact Deleted Success Fully"<<endl;
                    }
                      else if((command==1)&(temp->next==NULL))//if contact is
the last node
                    {
                        temp->prev->next=NULL;
                        delete temp;
                        cout<<"  Contact Deleted Success Fully"<<endl;
                      }
                      else if(command==1)//if contact is the middle node
```

```
                        {
                                Node *temp1;
                                temp1=temp;
                                temp->prev->next=temp1->next;
                                temp->next->prev=temp1->prev;
                                  delete temp1;
                                cout<<"  Contact Deleted Success Fully"<<endl;
                         }
                            else
                            {
                                cout<<"  You Enter Wrong Command"<<endl;
                            }
                 }
                 else if(Dcheck==false)
                         {
                                cout<<"  Contact of This Name Not Found."<<endl;
                         }
                 }
                 else
                 {
                     cout<<"  You Enter wrong Command"<<endl;//if there is no contact by
this phone number
                     }
              }
              void BubbleSort()// we will check adjacent element if adjacent element is greater
we will swap.
        {
            Node *temp=head;
            Node *i,*j;
            string n;
```

```
   long long int n2;
   if(temp==NULL)//if contact book is empty
   {
            cout<<"  Empty contact Book"<<endl;
                     }
                     else//if contact book is not empty
                     {
      for(i=head;i->next!=NULL;i=i->next)
      {
         for(j=i->next;j!=NULL;j=j->next)
         {
            if(i->name>j->name)
                        {
               n=i->name;
               i->name=j->name;
               j->name=n;

               n2=i->phone_number;
               i->phone_number=j->phone_number;
               j->phone_number=n2;
            }
         }
      }
   }
}


void EditContacts()
   {
            Node *temp=head;  // Initialize a temporary pointer
            cout<<"  Press 1 for search By Name"<<endl;
            cout<<"  Press 2 for search By Number"<<endl;
            int Ecommand;
            cout<<"  Enter the Command: ";
            cin>>Ecommand;

            if(Ecommand==1)
            {
                bool Echeck=false;
                cout<<"  Enter the Name to Edit: ";
                cin>>x;
                while(temp!=NULL)// Iterate through the contacts to find the one with
the specified name.
                {
                if(temp->name==x)
                {
                    cout<<"**********"<<endl;
                    cout<<"Name: "<<temp->name<<endl;
                    cout<<"Phone Number: "<<temp->phone_number<<endl;
                    Echeck=true;
                    break;
                }
                   temp=temp->next;
            }
            if(Echeck==true)
            {
                int command;
              cout<<"  Press 1 to Edit the Contact: ";
                cin>>command;
              if(command==1)
```

```
                         {
                            cout<<"  Enter New Name: ";
                          cin>>x;
                                cout<<"  Enter New Number: ";
                                cin>>y;
                          // Update the contact's name and phone number with the new values.
                                temp->name=x;
                          temp->phone_number=y;

                            cout<<"  Contact Edited Success Fully"<<endl;
                                }
                                else
                                {
                                            cout<<"  You Enter Wrong Command ... Try
Again"<<endl;
                        }
                      }
                      else if(Echeck==false)
                      {
                          cout<<"  Contact Not Found"<<endl;//There is no contact by this
name.
                                }
                  }
                    else if(Ecommand==2)
                    {
                          bool Echeck=false;
                          cout<<"  Enter the Number to Edit: ";
                          cin>>y;
                          while(temp!=NULL)// Iterate through the contacts to find the one with
the specified phone number.
```

```
        {
            if(temp->phone_number==y)
            {
                    cout<<"**********"<<endl;
                    cout<<"Name: "<<temp->name<<endl;
                    cout<<"Phone Number: "<<temp->phone_number<<endl;
                    Echeck=true;
                    break;
            }
                 temp=temp->next;
    }
    if(Echeck==true)
    {
            int command;
       cout<<"  Press 1 to Edit the Contact: ";
            cin>>command;
       if(command==1)
       {
               cout<<"  Enter New Name: ";
            cin>>x;
                    cout<<"  Enter New Number: ";
                    cin>>y;
// Update the contact's name and phone number with the new values.
                    temp->name=x;
            temp->phone_number=y;

               cout<<"  Contact Edited Success Fully"<<endl;
                    }
                    else
                    {
                                cout<<"  You Enter Wrong Command"<<endl;
                    }
                }
                else if(Echeck==false)
                {
                   cout<<"  There is No Contact of this Number."<<endl;// There is no contact
by this phone number.
                }
            }
            else
                    {
                        cout<<"  You Enter Wrong Command Try Again"<<endl;
                    }
          }
                void Structure()
        {
                cout<<"------------"<<endl;
                cout<<"  1. Add Contact"<<endl;
                cout<<"  2. Edit the Contact"<<endl;
                cout<<"  3. Delete Contact"<<endl;
                cout<<"  4. Search Contact"<<endl;
                cout<<"  5. Display All Contacts"<<endl;
                cout<<"  6. Delete All Contacts"<<endl;
                cout<<"-------------"<<endl;

                int Scommand;
                cout<<"  Enter the Command: ";
                cin>>Scommand;
                try
                {
                        if(Scommand>=1&&Scommand<=6)
```

```
{
            if(Scommand==1)
    {
            CreateNode();// Execute the method to add a new contact.
    Structure();// Present the menu again.
    }
    else if(Scommand==2)
    {
            EditContacts();// Execute the method to edit a contact.
            Structure();// Present the menu again.
        }
    else if(Scommand==3)
    {
    DeleteContactBySearch();// Execute the method to delete a contact.
    Structure(); // Present the menu again.
    }
    else if(Scommand==4)
    {
            Search(); // Execute the method to search for a contact.
            Structure();
    }
    else if(Scommand==5)
    {
            Display();// Execute the method to display all contacts.
            Structure();
    }
    else if(Scommand==6)// Execute the method to delete all contacts.
    {
            DeleteAllContacts();
            Structure();
```

```
                                    }
                                        }
                                        else // If the entered command is out of range, throw an
exception.
                                        {
                                                throw(Scommand);
                                        }
                                }
                                catch(int Scommand)
                                {
                                        cout<<"  You Enter wrong Command Run the Code
Again"<<endl;
                                        Structure();// Handle the exception by presenting the menu
again.
                                }
                }


};
int main()
{
   ContactBook cb;// Create an instance of the ContactBook class
        string n;// Declariung a stringto store the user's name.
        cout<<"  What is Your Name: ";
        cin>>n;


   cout<<"DSA MINI PROJECT"<<endl;
        cout<<"  "<<n<<"  WELCOME TO CONTACTBOOK      "<<endl;
        cout<<"This thing makes our life easier"<<endl;
        cb.Structure();// Call the Structure method of the ContactBook to start the contact
book interface.
        return 0;
}
```

**5)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.

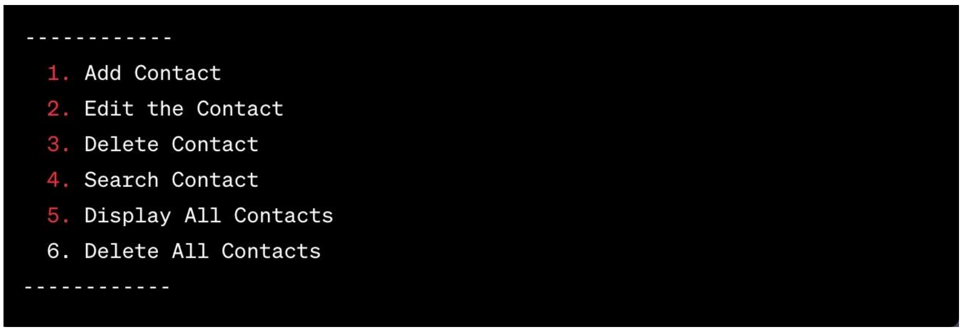
6. Delete All Contacts: This option deletes all the contacts in the contact book.

* The code uses a simple bubble sort algorithm to sort contacts by name when displaying all contacts.

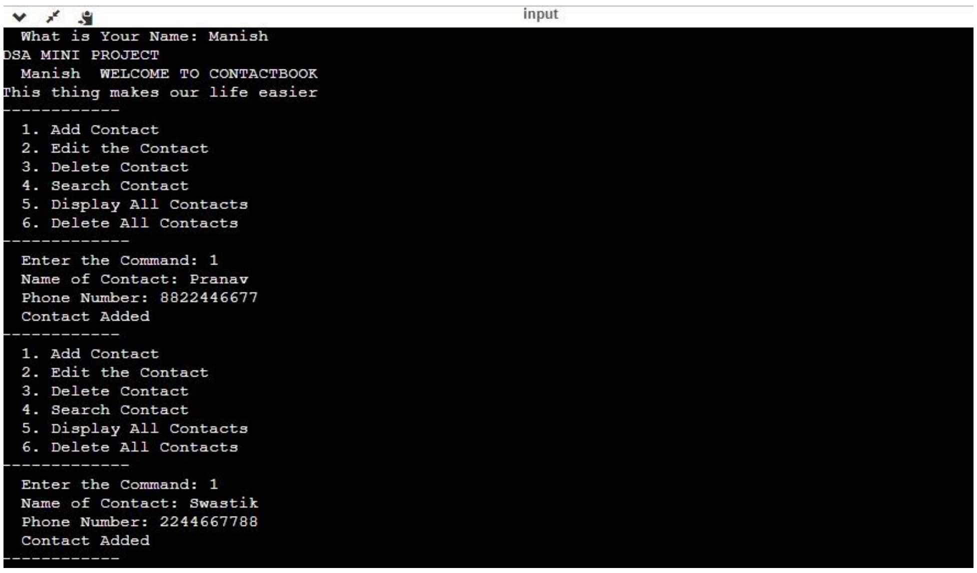
* If you enter a command that is not between 1 and 6, it will display a message telling you to try again and present the menu again.

* The program will keep running until you decide to exit.

## 5)c. OUTPUT:



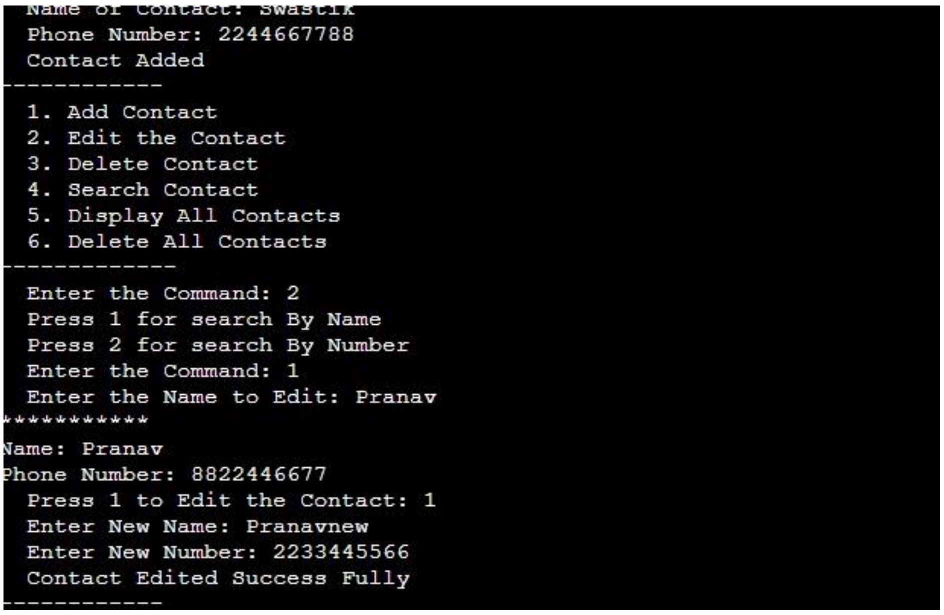
```
------------
  1. Add Contact
  2. Edit the Contact
  3. Delete Contact
  4. Search Contact
  5. Display All Contacts
  6. Delete All Contacts
------------
```



```
 What is Your Name: Manish
DSA MINI PROJECT
 Manish  WELCOME TO CONTACTBOOK
This thing makes our life easier
------------
 1. Add Contact
 2. Edit the Contact
 3. Delete Contact
 4. Search Contact
 5. Display All Contacts
 6. Delete All Contacts
-------------
 Enter the Command: 1
 Name of Contact: Pranav
 Phone Number: 8822446677
 Contact Added
------------
 1. Add Contact
 2. Edit the Contact
 3. Delete Contact
 4. Search Contact
 5. Display All Contacts
 6. Delete All Contacts
-------------
 Enter the Command: 1
 Name of Contact: Swastik
 Phone Number: 2244667788
 Contact Added
------------
```

**EXPLANATION**

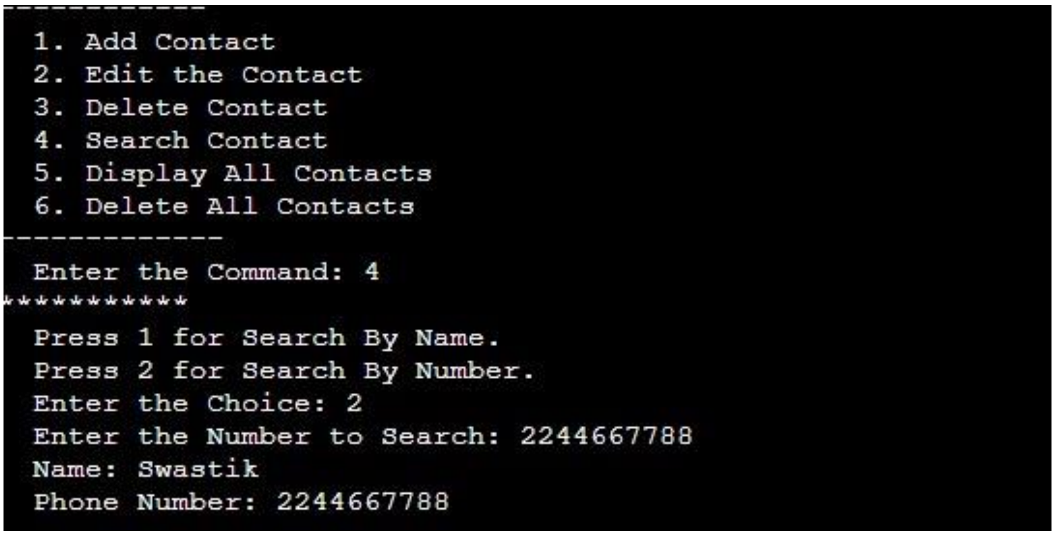
This Output demonstrates the starting screen of our Contact Book and we start by asking the name of user and in this we have created 2 contacts.



```
 Name of Contact: Swastik
 Phone Number: 2244667788
 Contact Added
------------
 1. Add Contact
 2. Edit the Contact
 3. Delete Contact
 4. Search Contact
 5. Display All Contacts
 6. Delete All Contacts
-------------
 Enter the Command: 2
 Press 1 for search By Name
 Press 2 for search By Number
 Enter the Command: 1
 Enter the Name to Edit: Pranav
***********
Name: Pranav
Phone Number: 8822446677
 Press 1 to Edit the Contact: 1
 Enter New Name: Pranavnew
 Enter New Number: 2233445566
 Contact Edited Success Fully
------------
```

**EXPLANATION**

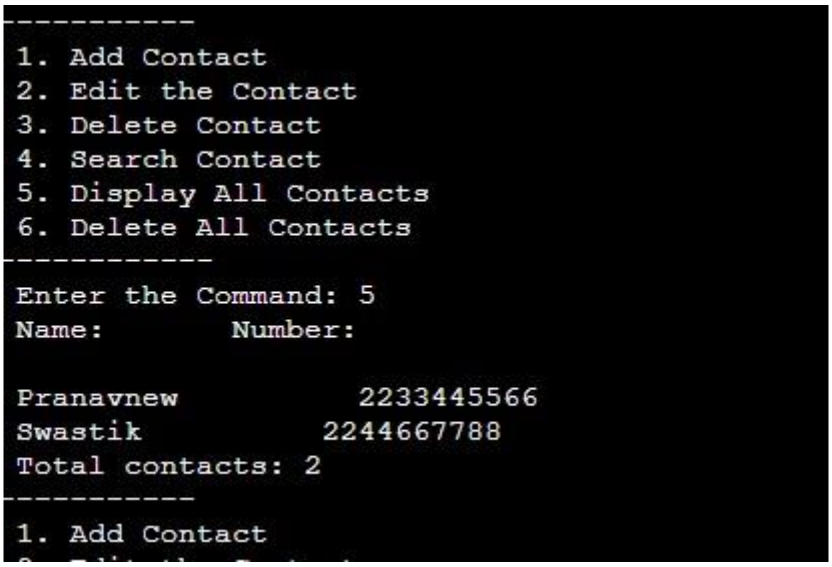
This Output demonstrates that the user is given a choice of either to edit the contact using name or number of the person, then the contact name and number is displayed which the user has chosen then the user can edit the contact by pressing 1.



```
------------
 1. Add Contact
 2. Edit the Contact
 3. Delete Contact
 4. Search Contact
 5. Display All Contacts
 6. Delete All Contacts
------------
 Enter the Command: 4
***********
 Press 1 for Search By Name.
 Press 2 for Search By Number.
 Enter the Choice: 2
 Enter the Number to Search: 2244667788
 Name: Swastik
 Phone Number: 2244667788
```

**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.

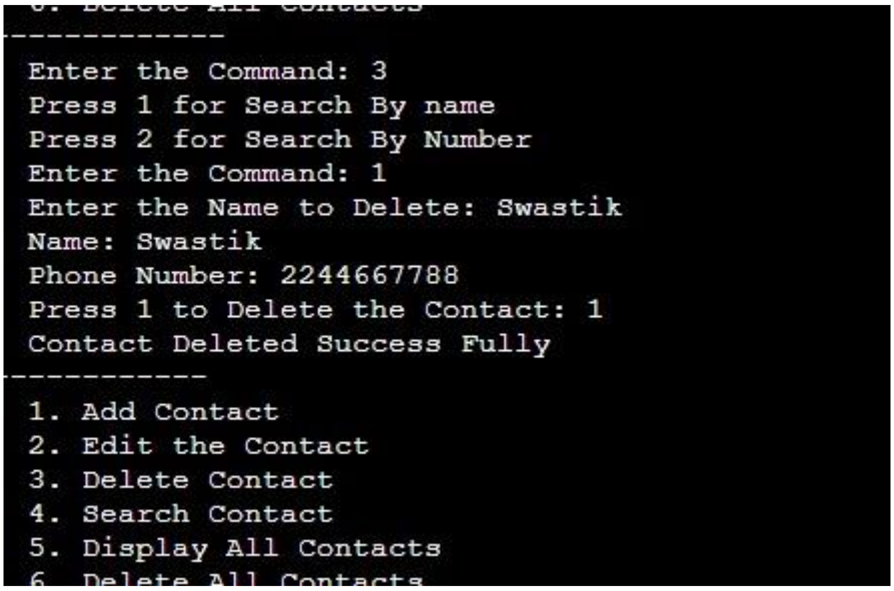


```
-----------
1. Add Contact
2. Edit the Contact
3. Delete Contact
4. Search Contact
5. Display All Contacts
6. Delete All Contacts
-----------
Enter the Command: 5
Name:       Number:

Pranavnew         2233445566
Swastik          2244667788
Total contacts: 2
-----------
1. Add Contact
```

**EXPLANATION**

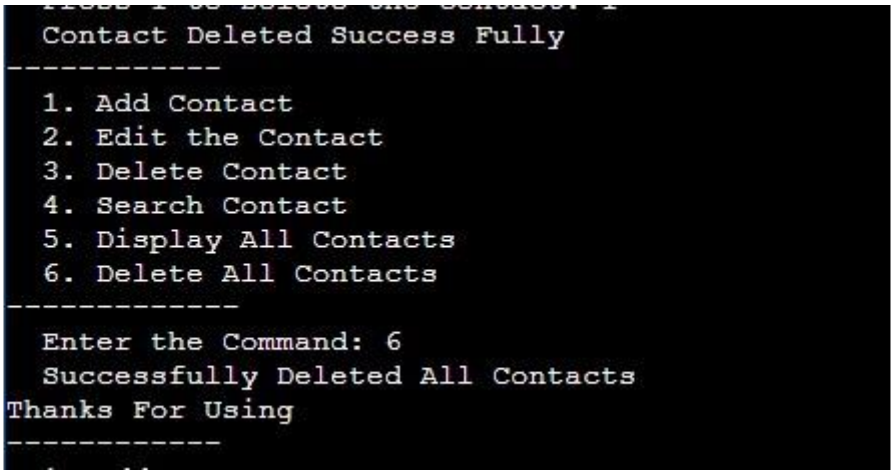
This Output demonstrates that if the user wants to display all the contact in the contact book that he/she can display using display all contacts command and all the contact including their counts are displayed.



```
------------
 Enter the Command: 3
 Press 1 for Search By name
 Press 2 for Search By Number
 Enter the Command: 1
 Enter the Name to Delete: Swastik
 Name: Swastik
 Phone Number: 2244667788
 Press 1 to Delete the Contact: 1
 Contact Deleted Success Fully
-----------
 1. Add Contact
 2. Edit the Contact
 3. Delete Contact
 4. Search Contact
 5. Display All Contacts
```

**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.



```
  Contact Deleted Success Fully
------------
  1. Add Contact
  2. Edit the Contact
  3. Delete Contact
  4. Search Contact
  5. Display All Contacts
  6. Delete All Contacts
-------------
  Enter the Command: 6
  Successfully Deleted All Contacts
Thanks For Using
------------
```

**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.

## 6)TIME COMPLEXITY:

**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:
a)Search Operation
b)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:
1.Outer loop
2.Inner loop

**3.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 factor 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.