PROJECT REPORT

Submitted by

SWARNA TRISHA GHANTY (RA2211026010333)

Under the Guidance of

Dr.M.Vimaladevi

Assistant Professor , CINTEL

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING

with specialization in Artificial Intelligence and Machine Learning

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

MAY 2023

BONAFIDE CERTIFICATE

Certified that this project report titled "Phonebook Application" is the bonafide work of Swarna Trisha Ghanty(RA2211026010333)who carried out the project work under my supervision.Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

CONTENTS

PROBLEM STATEMENT

Design and implement a phone book application that allows users to store and manage their contacts. The application should provide the following functionality:

- Add a new contact: Users should be able to add new contacts to the phone book by providing their name and phone number
.
- Delete a contact: Users should be able to delete a contact from the phone book.
- Search for a contact: Users should be able to view a contact's name and phone number by searching for the contact using their name.
- View all contacts: Users can view all of the contacts in their phonebook in a list.
- Edit a contact: Users should be able to edit a contact's name and/or phone number.
- Sort contacts: Users can sort their contacts by name, phone number, or email address to make it easier to find the contact they need.
- Group contacts: Users can group their contacts by category or tag to organise them and make it easier to find contacts with specific attributes.

The phone book application should be implemented in C++ and should use appropriate data structures (such as arrays, linked lists, or maps) to store the contacts.Additionally, the application should provide a user-friendly interface that allows users to easily navigate and use the different features.

MODULES

This code is a simple phonebook application written in C++. The program uses a vector of structs to store contacts, and allows the user to perform various operations on the contacts, such as adding, deleting, searching, and updating.

The main data structure used in this code is a vector of structs. The struct, named "Contact", has three fields: name, phoneNumber, and email. The vector, named "phoneBook", is a vector of Contact structs.

The program has several functions that allow the user to interact with the phone book. Here is a brief description of each function:

addContact(): This function prompts the user to enter the name, phone number, and email address of a new contact, and adds the contact to the phone book vector.

deleteContact(): This function prompts the user to enter a search query (a name, phone number, or email address), and deletes all contacts that match the query from the phone book vector.

searchContacts(): This function prompts the user to enter a search query (a name, phone number, or email address), and displays all contacts in the phone book vector that match the query.

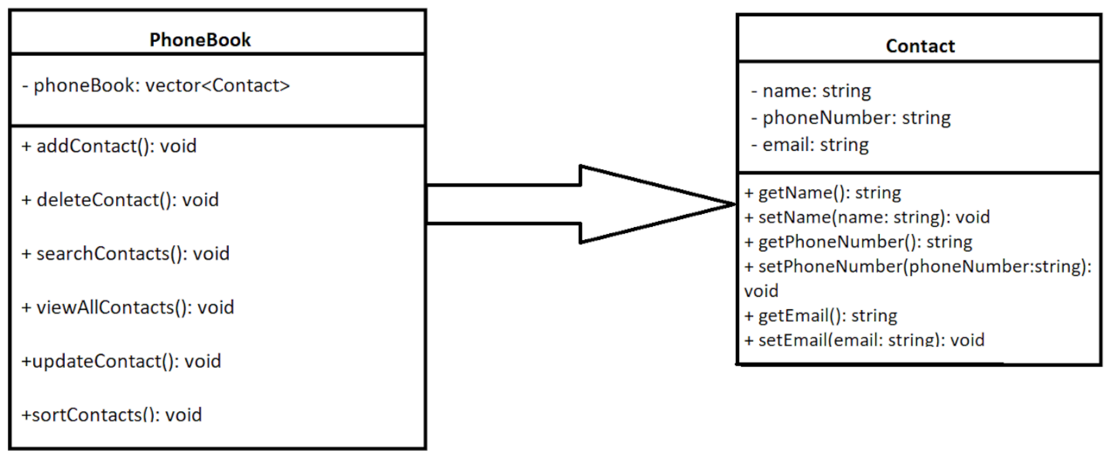
viewAllContacts(): This function displays all contacts in the phone book vector.

updateContact(): This function prompts the user to enter a search query (a name, phone number, or email address) to select a contact to update. It then displays the current information for the contact and prompts the user to enter updated information for the contact. The function then updates the contact in the phone book vector.

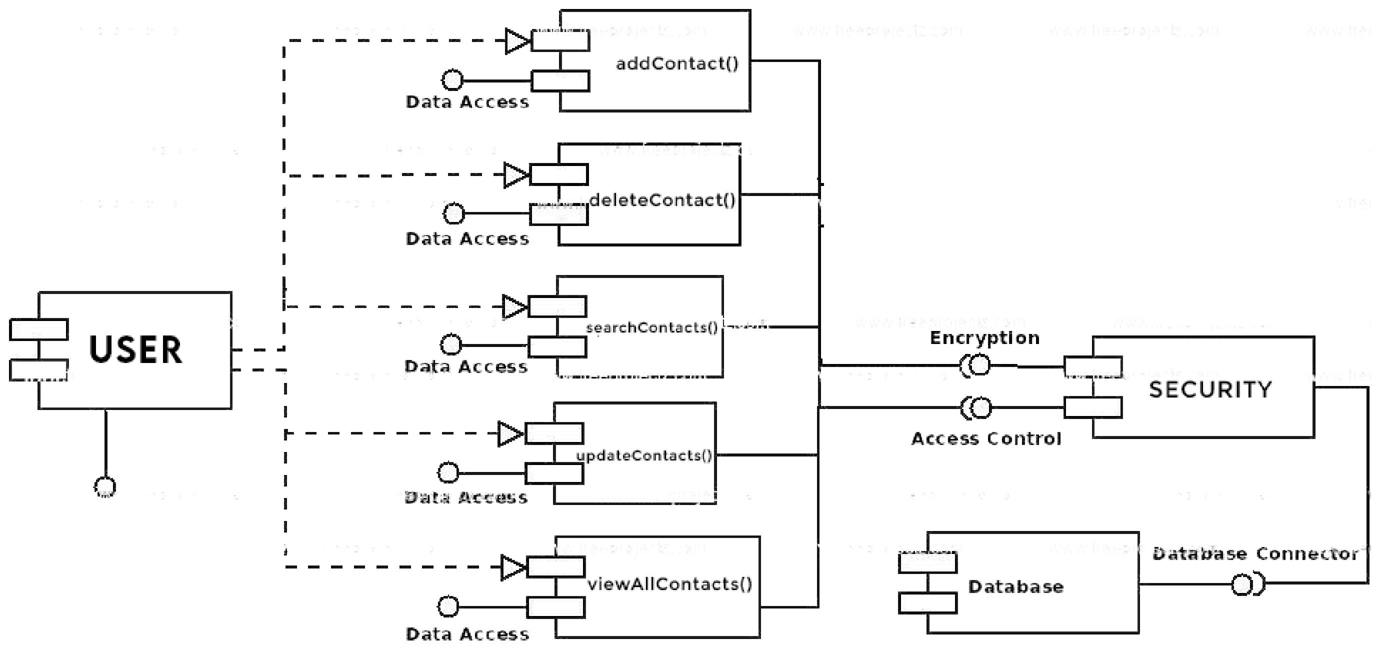
sortContacts(): This function prompts the user to choose a sort option (by name, phone number, or email address), and sorts the phone book vector accordingly using the sort function from the algorithm library.

UML DIAGRAMS

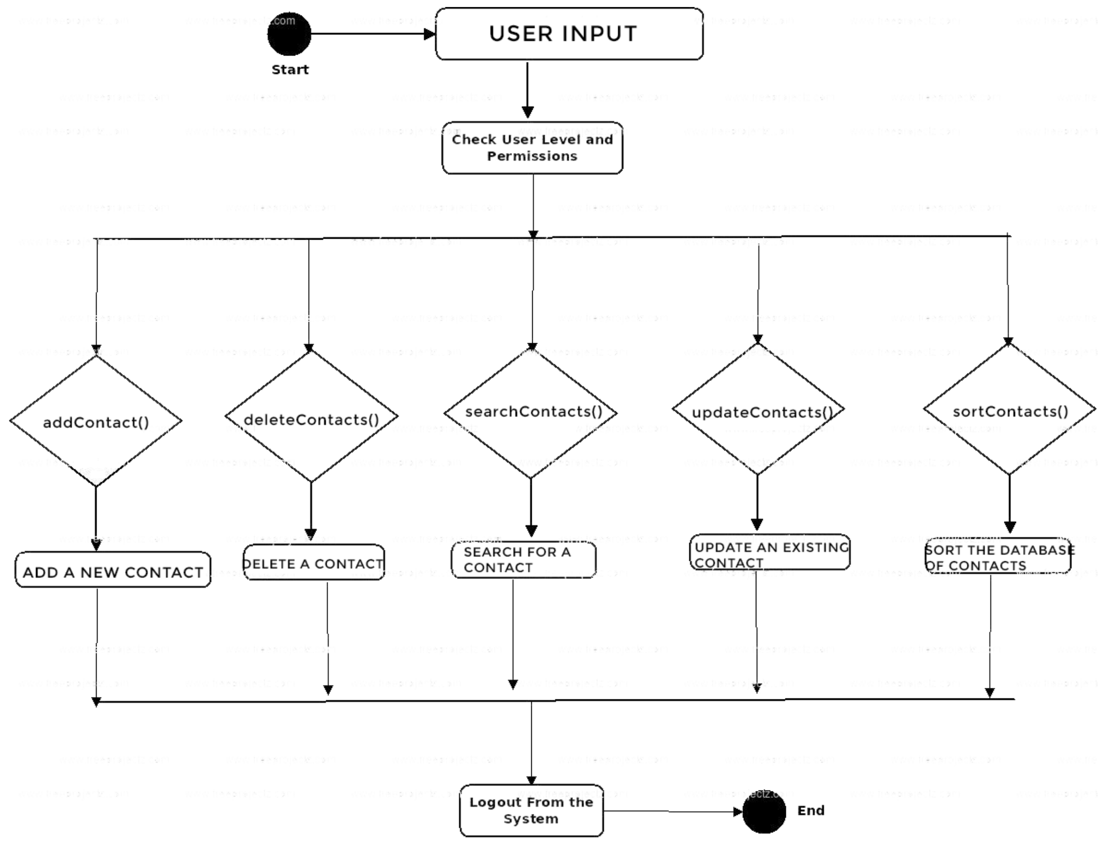
Class Diagram



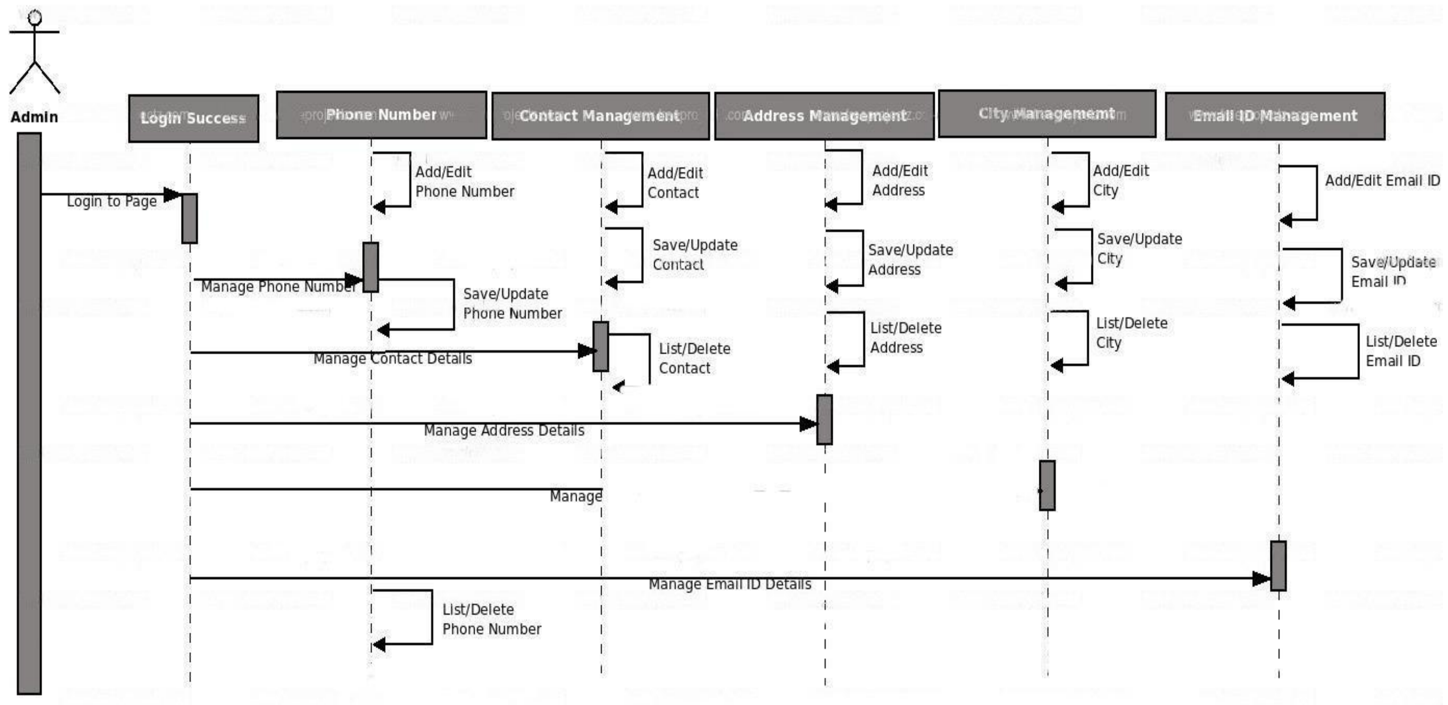
Block/Architecture Diagram



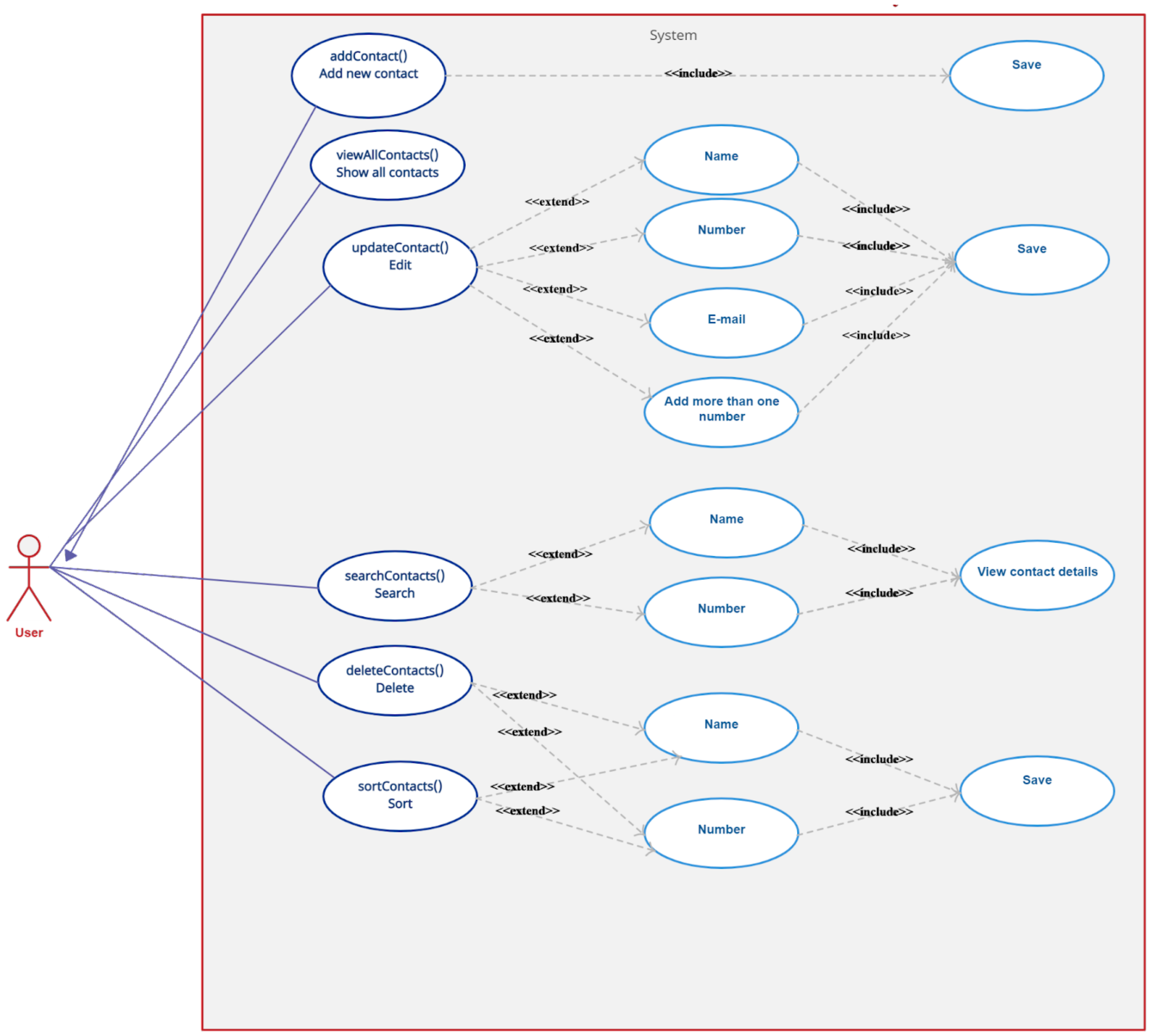
Activity Diagram



Sequence Diagram



Use Case Diagram



Source Code

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Contact
{
  string name;
  string phoneNumber;
  string email;
};

vector < Contact > phoneBook;

void addContact ()
{
  Contact newContact;
  cout << "Enter the name of the new contact: ";
  getline (cin, newContact.name);
  cout << "Enter the phone number of the new contact: ";
  getline (cin, newContact.phoneNumber);
  cout << "Enter the email address of the new contact: ";
  getline (cin, newContact.email);
  phoneBook.push_back (newContact);
  cout << "Contact added successfully!" << endl;
}

void deleteContact ()
{
  string searchQuery;
  int numContactsFound = 0;
  cout <<"Enter the name, phone number, or email address of the contact you
wish to   delete: ";
  getline (cin, searchQuery);
  for (int i = 0; i < phoneBook.size (); i++)
    {
      if (phoneBook[i].name == searchQuery
              || phoneBook[i].phoneNumber == searchQuery
              || phoneBook[i].email == searchQuery)
            {
              phoneBook.erase (phoneBook.begin () + i);
              numContactsFound++;
              i--;     // since the element at i was removed, the next element now
occupies the ith position
            }
    }
  if (numContactsFound == 0)
    {
      cout << "No contacts found matching the search query." << endl;
    }
  else
    {
      cout << numContactsFound <<" contact(s) matching the search query
deleted successfully!" << endl;
    }
}

void searchContacts ()
{
  string searchQuery;
  int numContactsFound = 0;
  cout << "Enter the name, phone number, or email address of the contact you
wish to search for: ";
  getline (cin, searchQuery);
  for (int i = 0; i < phoneBook.size (); i++)
    {
      if (phoneBook[i].name == searchQuery
              || phoneBook[i].phoneNumber == searchQuery
              || phoneBook[i].email == searchQuery)
            {
              cout << "Name: " << phoneBook[i].name << endl;
              cout << "Phone number: " << phoneBook[i].phoneNumber << endl;
              cout << "Email: " << phoneBook[i].email << endl;
              numContactsFound++;
            }
    }
  if (numContactsFound == 0)
    {
      cout << "No contacts found matching the search query." << endl;
    }
```

```
 else
    {
      cout << numContactsFound <<" contact(s) found matching the search
query." << endl;
    }
}

void viewAllContacts ()
{
  if (phoneBook.size () == 0)
    {
       cout << "No contacts found in the phone book." << endl;
    }
  else
    {
      cout << "All contacts in the phone book:" << endl;
      for (int i = 0; i < phoneBook.size (); i++)
           {
            cout << "Name: " << phoneBook[i].name << endl;
            cout << "Phone number: " << phoneBook[i].phoneNumber << endl;
            cout << "Email: " << phoneBook[i].email << endl;
            cout << endl;
           }
    }
}

void updateContact ()
{
  string searchQuery;
  int numContactsFound = 0;
  cout <<"Enter the name, phone number, or email address of the contact you
wish to update: ";
  getline (cin, searchQuery);
  for (int i = 0; i < phoneBook.size (); i++)
    {
      if (phoneBook[i].name == searchQuery
            || phoneBook[i].phoneNumber == searchQuery
            || phoneBook[i].email == searchQuery)
           {
            cout << "Current information for the selected contact:" << endl;
            cout << "Name: " << phoneBook[i].name << endl;
            cout << "Phone number: " << phoneBook[i].phoneNumber << endl;
            cout << "Email: " << phoneBook[i].email << endl;
            cout << endl;
```

```
         cout << "Enter the updated information for the contact:" << endl;
         cout << "Name (leave blank to keep current name): ";

         string updatedName;
         getline (cin, updatedName);
         if (updatedName != "")
           {
            phoneBook[i].name = updatedName;
           }
         cout << "Phone number (leave blank to keep current phone number):
";
         string updatedPhoneNumber;
         getline (cin, updatedPhoneNumber);
         if (updatedPhoneNumber != "")
           {
             phoneBook[i].phoneNumber = updatedPhoneNumber;
           }
         cout << "Email (leave blank to keep current email): ";
         string updatedEmail;
         getline (cin, updatedEmail);
         if (updatedEmail != "")
           {
            phoneBook[i].email = updatedEmail;
           }
         cout << "Contact updated successfully!" << endl;
         numContactsFound++;
        }
  }
 if (numContactsFound == 0)
```

```
    {
      cout << "No contacts found matching the search query." << endl;
    }
}

void sortContacts ()
{
  int sortOption;
  cout << "Enter 1 to sort by name, 2 to sort by phone number, or 3 to sort by
email address: ";
  cin >> sortOption;
  cin.ignore ();    // ignore the newline character left in the input stream by cin
  if (sortOption == 1)
    {
```

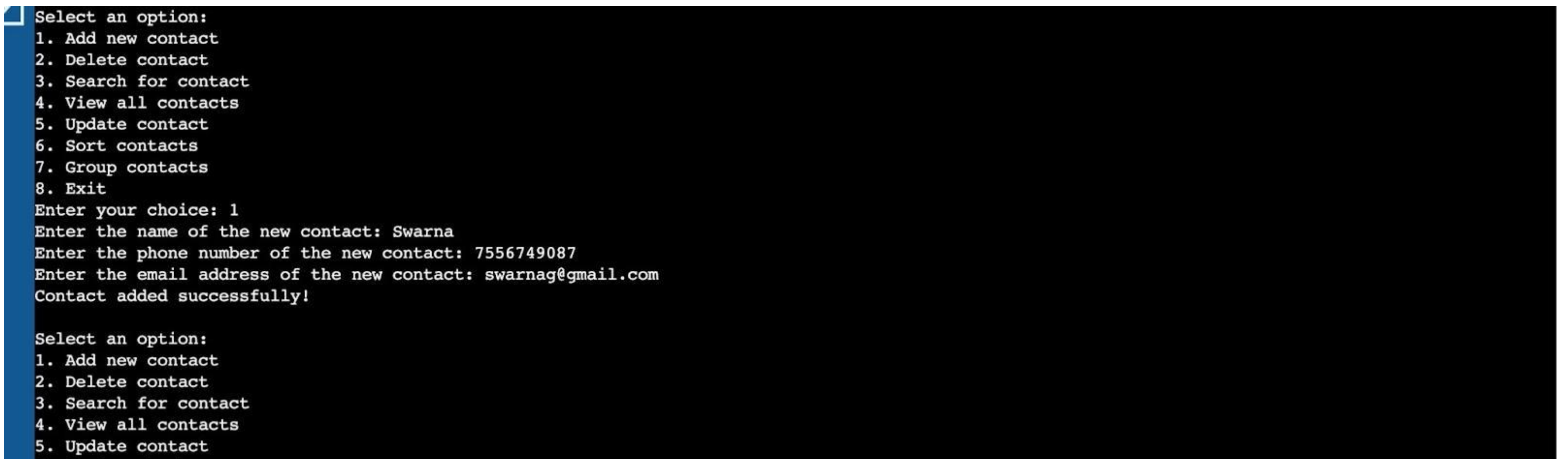
```
      sort (phoneBook.begin (), phoneBook.end (), [](const Contact & a, const
Contact & b)
     {
        return a.name < b.name;}
      );
      cout << "Phone book sorted by name!" << endl;
    }
  else if (sortOption == 2)
    {
      sort (phoneBook.begin (), phoneBook.end (),
            [](const Contact & a, const Contact & b)
            {
            return a.phoneNumber < b.phoneNumber;}
      );
      cout << "Phone book sorted by phone number!" << endl;
    }
  else if (sortOption == 3)
    {
      sort (phoneBook.begin (), phoneBook.end (),
            [](const Contact & a, const Contact & b)
            {
            return a.email < b.email;}
      );
      cout << "Phone book sorted by email address!" << endl;
    }
  else
    {
      cout << "Invalid input. Phone book not sorted." << endl;
    }
}

void groupContacts ()
{
// This function could be implemented to group contacts by category or tag
using a map or another data structure,
// but for simplicity's sake, we will just group contacts by first letter of their
name.
  vector < vector < Contact >> groupedContacts (26);
  for (int i = 0; i < phoneBook.size (); i++)
    {
      char firstLetter = phoneBook[i].name[0];
      if (isupper (firstLetter))
	{
	  firstLetter = tolower (firstLetter);	// convert to lowercase for ease of
indexing the vector
```

```
	}
      groupedContacts[firstLetter - 'a'].push_back (phoneBook[i]);
    }
  cout << "Contacts grouped by first letter of name:" << endl;
  for (int i = 0; i < groupedContacts.size (); i++)
    {
      if (groupedContacts[i].size () > 0)
	{
	  cout << (char) ('a' + i) << endl;	// convert back to uppercase for
display
	  for (int j = 0; j < groupedContacts[i].size (); j++)
	    {
	      cout << "  " << groupedContacts[i][j].name << endl;
	    }
	}
    }
}
```

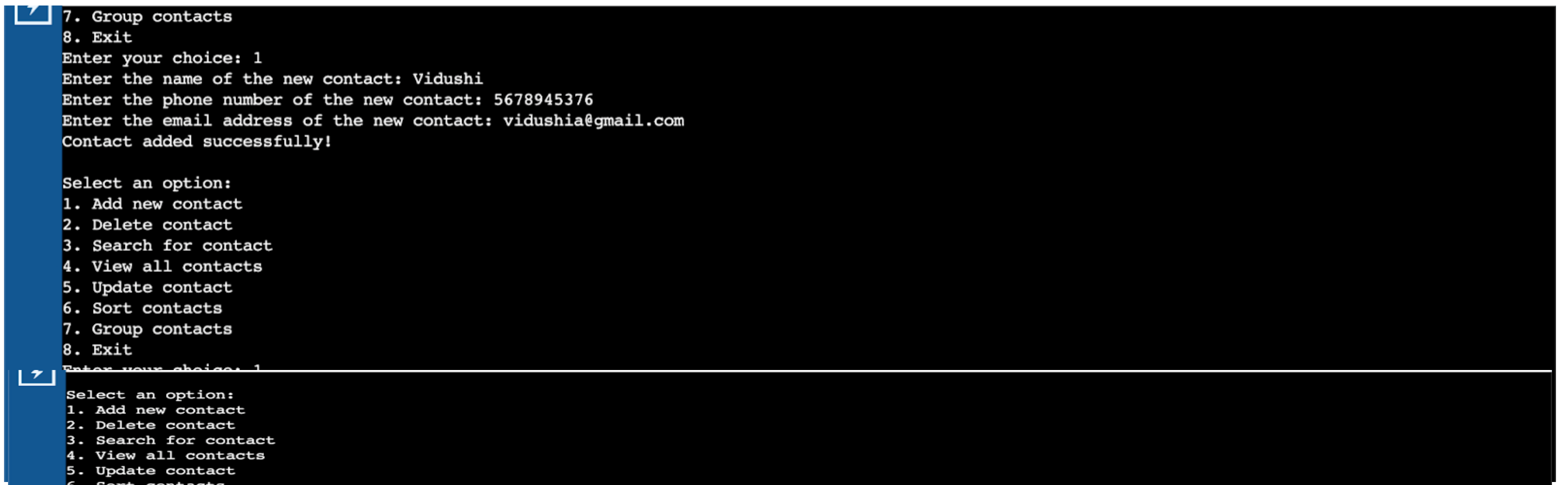
```
int main ()
{
  int userChoice;
  do
    {
      cout << "Select an option:" << endl;
      cout << "1. Add new contact" << endl;
      cout << "2. Delete contact" << endl;
      cout << "3. Search for contact" << endl;
      cout << "4. View all contacts" << endl;
      cout << "5. Update contact" << endl;
      cout << "6. Sort contacts" << endl;
      cout << "7. Group contacts" << endl;
      cout << "8. Exit" << endl;
      cout << "Enter your choice: ";
      cin >> userChoice;
      cin.ignore ();                    // ignore the newline character left in the input
stream by cin
      switch (userChoice)
            {
            case 1:
              addContact ();
              break;
            case 2:
              deleteContact ();
              break;
```

```
            case 3:
              searchContacts ();
              break;
            case 4:
              viewAllContacts ();
              break;
            case 5:
              updateContact ();
              break;
            case 6:
              sortContacts ();
              break;
            case 7:
              groupContacts ();
              break;
            case 8:
              cout << "Goodbye!" << endl;
              break;
            default:
              cout << "Invalid choice. Please enter a number from 1 to 8." <<
                endl;
              break;
            }
      cout << endl;
    }
  while (userChoice != 8);
  return 0;
}
```

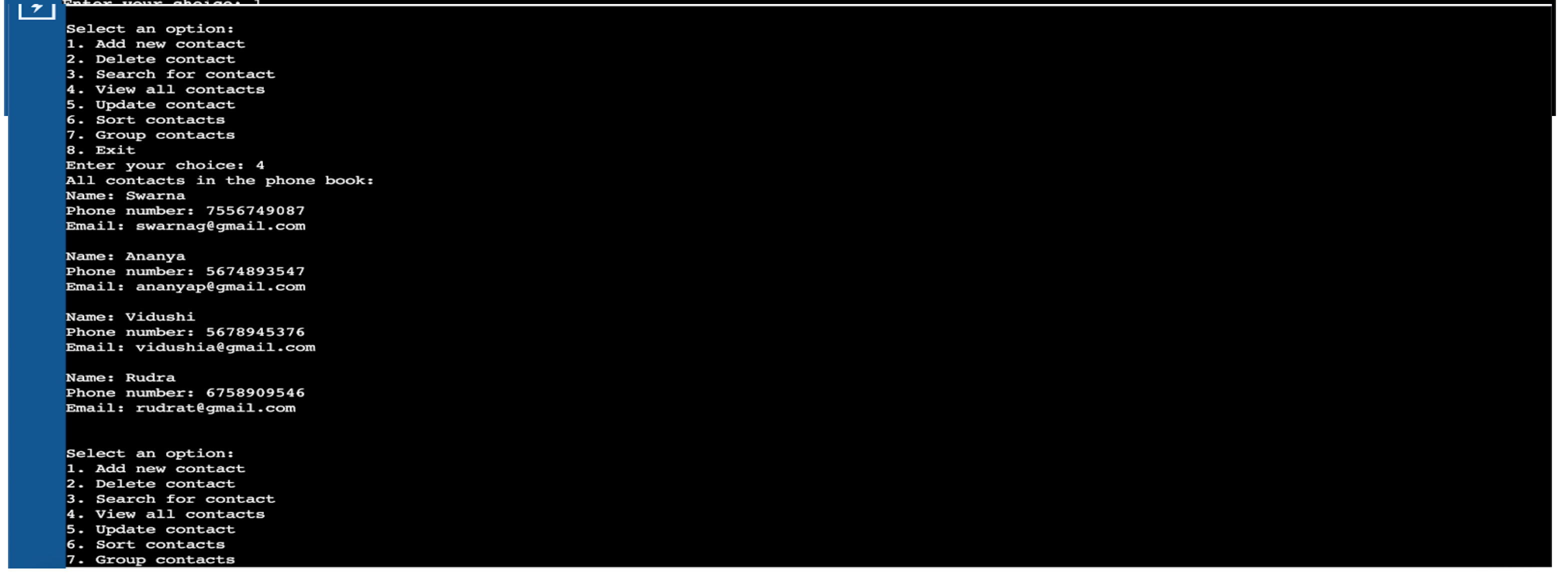
RESULT



```
Select an option:
1. Add new contact
2. Delete contact
3. Search for contact
4. View all contacts
5. Update contact
6. Sort contacts
7. Group contacts
8. Exit
Enter your choice: 1
Enter the name of the new contact: Swarna
Enter the phone number of the new contact: 7556749087
Enter the email address of the new contact: swarnag@gmail.com
Contact added successfully!

Select an option:
1. Add new contact
2. Delete contact
3. Search for contact
4. View all contacts
5. Update contact
```



```
7. Group contacts
8. Exit
Enter your choice: 1
Enter the name of the new contact: Vidushi
Enter the phone number of the new contact: 5678945376
Enter the email address of the new contact: vidushia@gmail.com
Contact added successfully!

Select an option:
1. Add new contact
2. Delete contact
3. Search for contact
4. View all contacts
5. Update contact
6. Sort contacts
7. Group contacts
8. Exit
Enter your choice: 1
```



```
Select an option:
1. Add new contact
2. Delete contact
3. Search for contact
4. View all contacts
5. Update contact
6. Sort contacts
7. Group contacts
8. Exit
Enter your choice: 4
All contacts in the phone book:
Name: Swarna
Phone number: 7556749087
Email: swarnag@gmail.com

Name: Ananya
Phone number: 5674893547
Email: ananyap@gmail.com

Name: Vidushi
Phone number: 5678945376
Email: vidushia@gmail.com

Name: Rudra
Phone number: 6758909546
Email: rudrat@gmail.com


Select an option:
1. Add new contact
2. Delete contact
3. Search for contact
4. View all contacts
5. Update contact
6. Sort contacts
7. Group contacts
```

TESTING

The code provided is a simple command-line phonebook application implemented in C++. The program allows the user to add, delete, search for, update, sort, group by and view all contacts in the phonebook.

The contacts are stored as instances of a Contact struct, which has three members: name, phoneNumber, and email. The phonebook is implemented using a vector of Contact structs, called phoneBook.

The program provides a menu-based interface to the user, allowing them to choose from various options by entering a number corresponding to the option they wish to select.

Here is a summary of the functionality provided by the program:

Add Contact: Allows the user to add a new contact to the phonebook. The user is prompted to enter the name, phone number, and email address of the new contact.

Delete Contact: Allows the user to delete an existing contact from the phonebook. The user is prompted to enter the name, phone number, or email address of the contact they wish to delete.

Search Contacts: Allows the user to search for a contact in the phonebook. The user is prompted to enter the name, phone number, or email address of the contact they wish to search for. If any contacts are found matching the search query, their details are displayed.

Update Contact: Allows the user to update an existing contact in the phonebook. The user is prompted to enter the name, phone number, or email address of the contact they wish to update. If any contacts are found matching the search query, their current details are displayed and the user is prompted to enter the updated details for the contact.

View All Contacts: Displays the details of all contacts in the phonebook.

Sort Contacts: Allows the user to sort the contacts in the phonebook by name, phone number, or email address. The user is prompted to enter a number corresponding to the sorting option they wish to select.

Overall, the program provides basic phonebook functionality and can be extended to add more features or improve the user interface.

LIMITATIONS

There are several limitations in this program, including:

1.Lack of input validation: The program assumes that the user will always input valid data. However, there is no input validation to ensure that the data entered by the user is in the correct format.

2.Limited search functionality: The search functionality in the program only searches for contacts based on name, phone number, or email address. It does not allow for more complex searches, such as searching for contacts based on a combination of criteria.

3.Lack of error handling: The program does not handle errors or exceptions very well. For example, if the user tries to add a contact with the same name as an existing contact, the program will add the new contact without any warning or error message.

4.Limited sorting options: The program only allows the user to sort the contacts by name, phone number, or email address. It does not allow for more complex sorting, such as sorting by multiple criteria or custom sorting.

5.Limited user interface: The program is a command-line interface, which may not be very user-friendly for some users. It does not have a graphical user interface (GUI) or any other interactive elements that could improve the user experience.

CONCLUSION

In conclusion, the provided code implements a simple phone book program using C++ and STL vectors. The program allows users to add, delete, search, view, update and sort contacts in the phone book. The code is well-structured and easy to understand, and uses good programming practices such as encapsulation and code reusability through functions.

However, the program has some limitations, such as the lack of data validation for input fields, which can lead to unexpected behaviour or errors when the user inputs invalid data. Additionally, the program does not provide an option to save or load the phone book data to/from a file, which can limit its usefulness as a practical phone book tool.

Overall, the provided code serves as a good starting point for a basic phone book program, but additional improvements can be made to enhance its functionality and reliability.

REFERENCE

C++ Language Tutorial - This tutorial covers the basics of the C++ programming language and can help you learn how to write the code for your phonebook application.

C++ Standard Library - This reference provides documentation for the standard library functions and containers used in the phonebook application, such as vectors and the sort function.

GeeksforGeeks - This website provides articles and tutorials on a variety of programming topics, including C++.

Github - This is a code hosting platform where you can find open source projects and repositories related to the phonebook application.