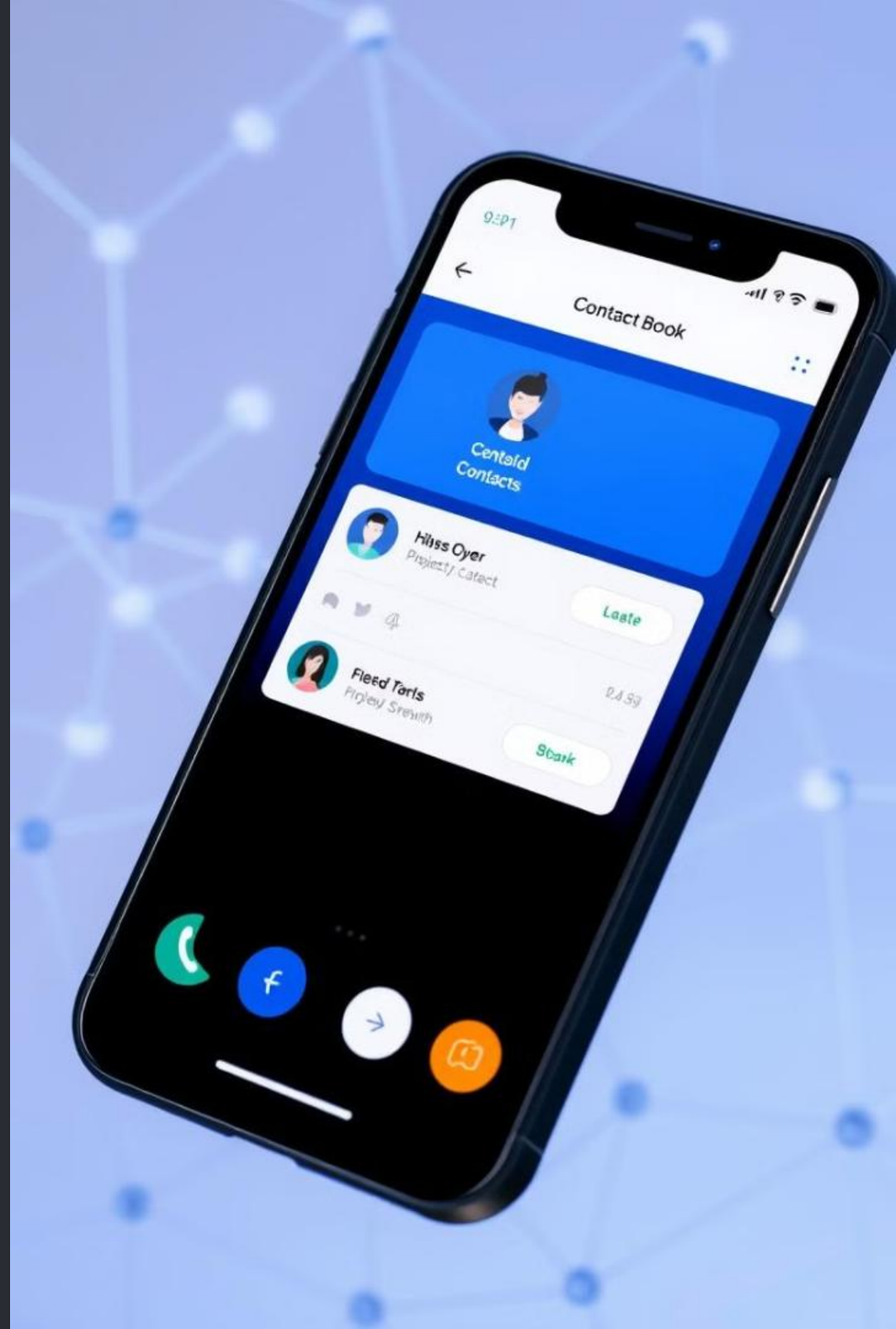
Contact Book: A C++ Implementation of Doubly Linked Lists

This Contact Book program showcases the practical application of doubly linked lists in C++. It demonstrates how these data structures can be used to create a functional contact management system, allowing users to add, edit, delete, and search contacts efficiently. The program also incorporates file handling for data persistence and implements sorting algorithms for organized display of contacts.

Through this implementation, students can gain hands-on experience with complex data structures and algorithms, while also learning about object-oriented programming principles and file I/O operations in C++.

By Angad kumar



Data Structure: Doubly Linked List

The Contact Book program utilizes a doubly linked list as its core data structure. This choice offers several advantages for managing contacts:

1 Bidirectional Traversal

Doubly linked lists allow for efficient traversal in both directions, facilitating operations like searching and sorting contacts.

2 Dynamic Memory Allocation

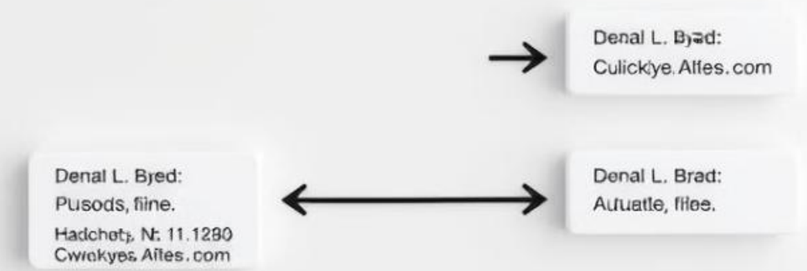
Contacts can be added or removed easily without the need for contiguous memory, allowing for flexible memory management.

3 Efficient Insertions and Deletions

Adding or removing contacts at any position in the list can be done in constant time, O(1), once the position is known.

4 Simplified Implementation of Operations

The bidirectional nature of the list simplifies implementations of operations like reverse traversal and deletion of nodes.



Key Classes and Structures

The program is built around two main components: the Node structure and the ContactBook class. These form the backbone of the contact management system:

Node Structure

Represents a single contact with fields for name and phone number. It also contains pointers to the next and previous nodes in the list, enabling the doubly linked list functionality.

ContactBook Class

Encapsulates all operations on the contact list. It manages the head of the list and provides methods for adding, editing, deleting, searching, and displaying contacts. This class also handles file I/O operations for data persistence.

Main Function

Acts as the entry point of the program, creating a ContactBook object and providing a user interface to interact with the contact management system.

Core Functionalities

The Contact Book program offers a range of essential features for managing contacts effectively:

1. **Add Contact**

   Users can add new contacts by providing a name and phone number. The program creates a new Node and inserts it at the end of the list.

2. **Edit Contact**

   Existing contacts can be modified by searching for them using either name or phone number, then updating the information.

3. **Delete Contact**

   Users can remove individual contacts or clear the entire list. The program handles memory deallocation to prevent memory leaks.

4. **Search Contact**

   Contacts can be searched by name or phone number, demonstrating linear search on a linked list.

5. **Display Contacts**

   All contacts are displayed in alphabetical order, showcasing the implementation of a sorting algorithm.

Sorting Algorithm: Bubble Sort

The Contact Book implements a bubble sort algorithm to arrange contacts alphabetically by name. While not the most efficient for large datasets, bubble sort is chosen for its simplicity and effectiveness for small to medium-sized contact lists:

1

Comparison

The algorithm compares adjacent elements (contacts) in the list.

2

Swapping

If contacts are in the wrong order, their data (name and phone number) is swapped.

3

Iteration

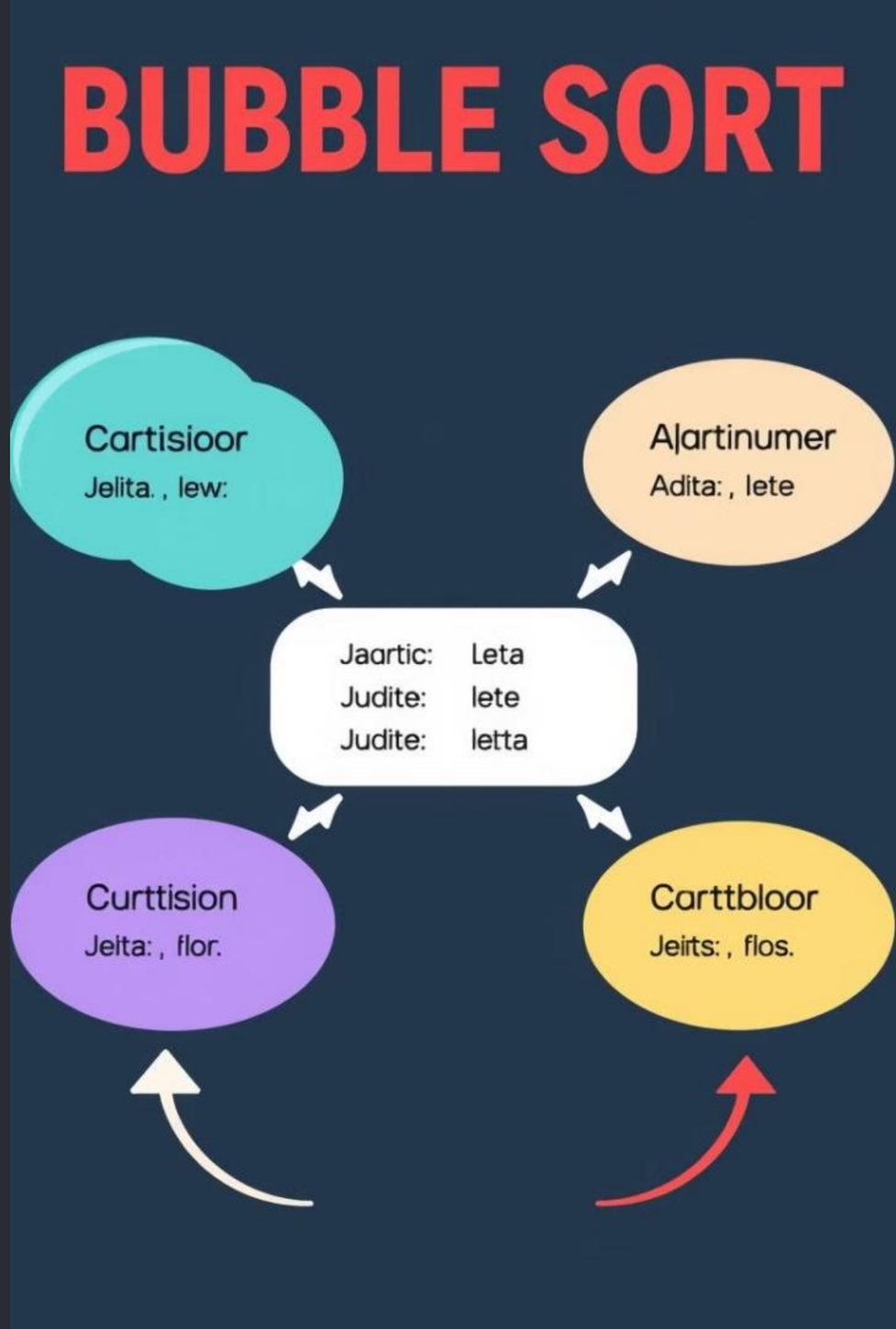
The process repeats for each pair of adjacent items until no swaps are needed.

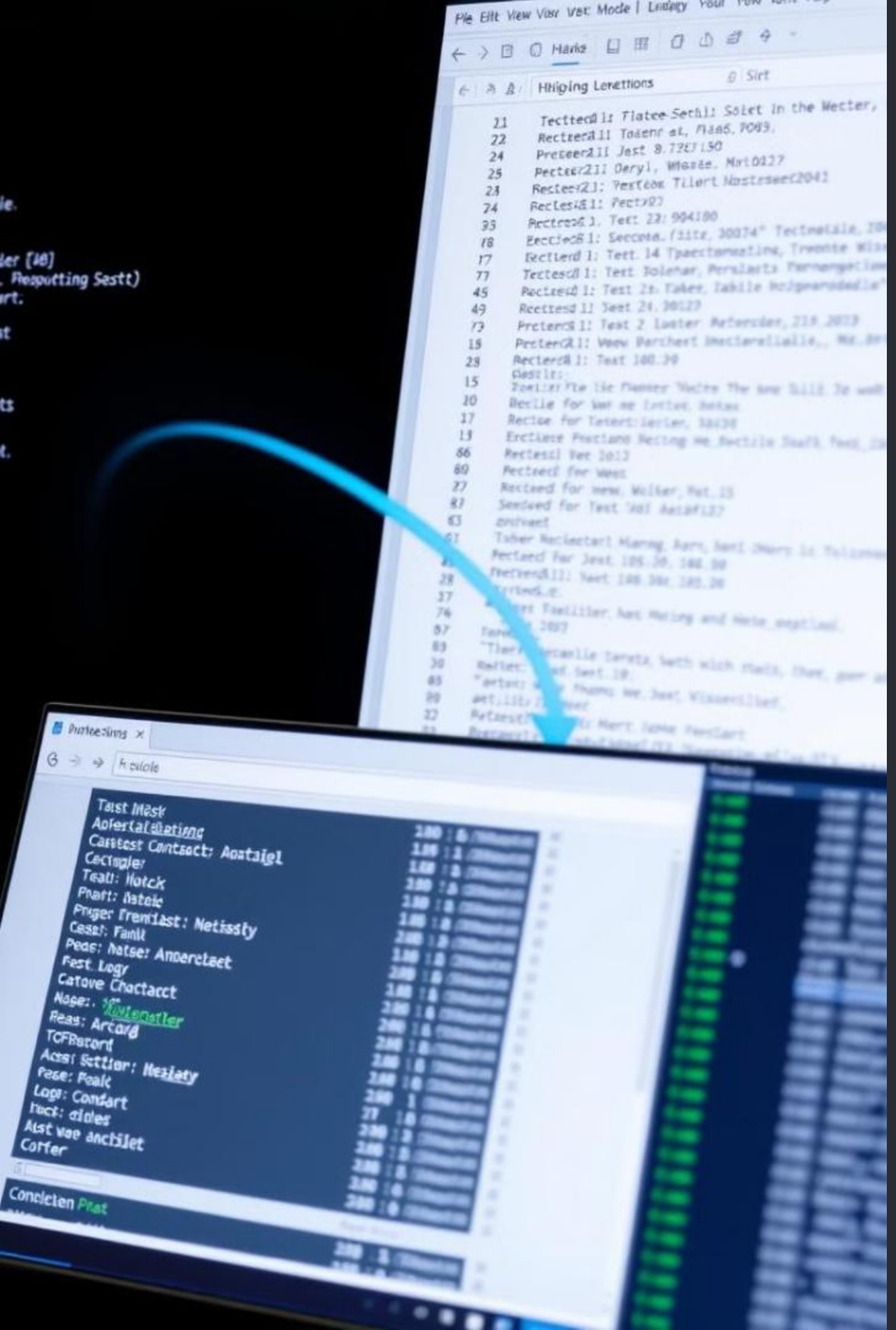
4

Completion

The list is sorted when a full pass requires no swaps.

This sorting ensures that contacts are always displayed in a user-friendly, alphabetical order, enhancing the usability of the contact book.





File Handling for Data Persistence

The Contact Book program implements file I/O operations to ensure data persistence between program executions. This feature demonstrates practical application of file handling in C++:

Operation	Description	C++ Streams Used
Saving Contacts	Writes contact data to "contactbook.txt"	ofstream
Loading Contacts	Reads contact data from "contactbook.txt"	ifstream
Data Format	Alternating lines of name and phone number	N/A

This implementation allows users to maintain their contact list across multiple sessions, enhancing the program's utility and demonstrating real-world application of file operations in software development.

Error Handling and User Interface

The Contact Book program incorporates error handling and a user-friendly interface to enhance its robustness and usability:

Exception Handling

Try-catch blocks are used to handle invalid user inputs, preventing crashes and providing informative error messages.

Menu-Driven Interface

A clear, menu-driven interface guides users through various operations, making the program accessible to users with varying levels of technical expertise.

Input Validation

The program validates user inputs to ensure data integrity, such as checking for valid phone numbers and non-empty names.

User Feedback

Clear messages are provided after each operation, confirming success or explaining failures to keep the user informed.

These features contribute to a more robust and user-friendly application, demonstrating important software engineering principles beyond just data structures and algorithms.

Learning Outcomes and Future Enhancements

The Contact Book program serves as an excellent learning tool for computer science students, offering practical experience with several key concepts:

Data Structures

Hands-on implementation of doubly linked lists, demonstrating their advantages in real-world applications.

Algorithms

Implementation of search and sort algorithms, providing insight into their workings and performance characteristics.

File I/O

Practical application of file handling for data persistence, a crucial skill in many software development scenarios.

OOP Principles

Demonstration of encapsulation, abstraction, and modular design through the use of classes and structures.

Future enhancements could include implementing more efficient sorting algorithms, adding data encryption for security, or extending the program to handle more complex contact information.