**Introduction**

A linked list is a linear data structure where each element, known as a node, contains a data part and a reference (or link) to the next node in the sequence. Unlike arrays, linked lists do not require contiguous memory allocation, making them efficient for insertion and deletion operations. However, linked lists do not support direct access to elements, necessitating traversal from the head node to access specific elements. This characteristic makes linked lists particularly suitable for applications where frequent insertions and deletions are required, as is the case with a phone directory.

**Background**

This project aims to demonstrate the practical implementation of a linked list in a real-world application by developing a phone directory application. Utilizing a linked list, we efficiently store names and phone numbers while exploring fundamental operations such as insertion, searching, and traversal. Integration of file handling ensures data persistence between program executions, crucial for maintaining the directory's information despite the volatile nature of linked lists. This approach highlights the efficiency of linked lists in managing dynamic datasets, particularly suited for applications requiring frequent data modifications like a phone directory.

**Problem Statement**

The objective is to create a phone directory using a linked list. The directory should:

1. Allow users to input names and phone numbers.
2. Store these entries efficiently in a linked list.
3. Enable to search for entries by name.
4. Display all stored entries.
5. Persist data between program executions using file handling.

**Methodology**

The program employs a singly linked list as the core data structure to manage the phone directory. Here is a brief overview of how the program operates:

### **Initialization**

Upon startup, the program reads data from data.txt to populate the linked list. This ensures that previously stored contacts are loaded into memory.

### **Menu-Driven Interface**

The program provides a menu with the following options:

1. Single Insert
2. Multi Insert
3. Search
4. Print All
5. Exit

### **Single and Multiple Insertions**

Users can insert new contacts either one at a time or in batches. Each entry is appended to the linked list and simultaneously written to data.txt to ensure persistence.

### **Search Functionality**

Users can search for a contact by name. The program traverses the linked list to find and display the corresponding phone number.

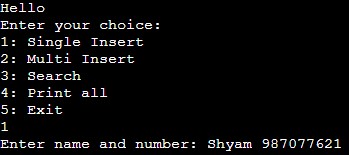
### **Display All Entries**

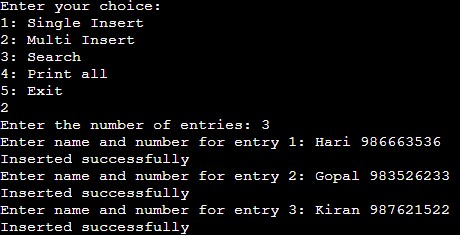
Users can view all contacts stored in the linked list, displaying each contact's name and phone number.

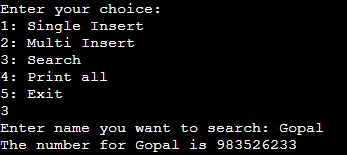
**File Handling**

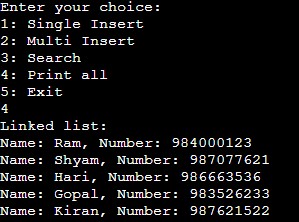
The program reads from data.txt at startup to initialize the linked list. It appends new entries to the file whenever new data is inserted, ensuring that the directory is updated.

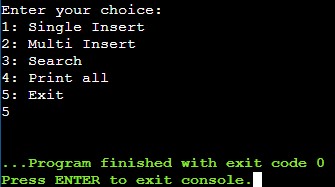
**Results**











**Discussion**

This project demonstrates the practical use of linked lists in managing a dynamic dataset, such as a phone directory. It shows how linked lists can efficiently handle data that frequently changes, like adding or removing contacts. The inclusion of file handling ensures data persistence between sessions, making the directory functional even after the program terminates. Linked lists offer significant advantages, such as efficient insertion and deletion operations and dynamic sizing, which are crucial for a phone directory. Future enhancements could include developing a graphical user interface to improve user experience, implementing automatic file creation for better robustness, and adding memory management routines to prevent memory leaks and enhance performance.

**Limitations**

* **No Graphical User Interface (GUI)**: The program operates solely in a console environment, limiting user interaction and usability for those accustomed to graphical interfaces. Enhancing the program with GUI could significantly improve user experience.
* **File Dependency**: If the data.txt file does not exist, the program will notify the user but will not create the file automatically. This requires manual file creation or prior runs that generate the file. Future versions could include automatic file creation if the file is not found.
* **Memory Management**: The program does not free the dynamically allocated memory for the nodes upon exit, leading to potential memory leaks if the program is run for extended periods or frequently restarted. Implementing proper memory management practices, such as freeing allocated memory, would be beneficial.

**Conclusion**

This project successfully demonstrates the implementation of a phone directory using a linked list in C. It integrates essential operations such as insertion, searching, and traversal, alongside file handling for data persistence. Despite some limitations, this project provides a solid foundation for understanding linked lists and their practical applications in managing dynamic datasets. Future enhancements could further improve the robustness and user-friendliness of the application, such as adding a graphical user interface, implementing automatic file creation, and enhancing memory management. This project exemplifies how fundamental data structures can be applied to create functional and useful applications, showcasing the importance of both theoretical knowledge and practical skills in computer science.

**Code**

https://github.com/gautam-manish/Phone\_Directory