**Date: 04 Oct 2019**

**Expt. No. : 16**

**UCS 1312 : DATA STRUCTURES**

**MINI PROJECT**

**TOPIC:** Telephone Directory Management Using **HASH TABLE**

**TEAM MEMBERS:**

1. S . Vishakan

2. V. Vikram

3. H.K Vishal

**AIM:**

The aim of this mini-project is to implement a real-time Telephone Directory Management System, and to perform various operations of that system using the **Hash Table** data structure.

**DESCRIPTION:**

* The underlying idea behind implementing the **Telephone Directory Management System(TDMS)** is the Hash Table data structure.
* We store the Name, Phone Number, Address and PIN Code of a Person as a **record** in the directory.
* Hash Table was preferred over other linear data structures (like Linked List) and non-linear data structures (like Trees) due to its efficiency in insertion and retrieval times i.e **O(1)** time complexity compared to **O(n)** in Linked List and **O(log n)** in Trees.
* Collision resolution in the Hash Table was done using **Separate Chaining** so that the Hash Table can hold a large dataset before its storage space gets exhausted.
* Since Separate Chaining was implemented, there is a good chance that the Hash Table slowly transforms into an array of Linked Lists, thus making the Hashing redundant.
* Thus, a good hash function and a large table is necessary for utilisation of the Hash Table efficiently.
* Table size was thus chosen to be 1,00,000 and a custom **hash function** was implemented. The hash function generates a hash value based on the **NAME** of the Person in the record.
* Hash Function : **Hash = Hash + (3\*<index>\*<ASCII of char\_at\_index>)**

In case the hash value exceeds 100000, we divide it by **723** to prevent overflow.

* Thus, the deviation from the usual hash value of **value%<table\_size>** is justified due to its better performance with respect to insertion, fetching and deletion. Clustering of data is also significantly low. The custom hash function is somewhat like an **Open Addressing** strategy, as the hash value is somewhat random.
* Thus, insertion operation is performed in the Hash Table, and also the insertion is performed into the database so that the record is stored permanently in secondary storage.
* The inserted records can be viewed in **Hash Table** format, **DatabaseRecord** format and as a **Sorted Database**.
* **Sorting** the database is done based on **Lexicographical Order** of name, and is sorted using the **Quick Sort** algorithm, which has an average time complexity of **O(log n)** making it one of the fastest available sorting algorithms.
* The inserted records can also be **updated/ removed** if the user wishes to make changes to the directory. The changes will be reflected in the Hash Table as well as the Database.
* The user can also **search** for a particular record based on **Name**.
* Thus, a complete **Telephone Directory Management System** was implemented, along with its required operations.

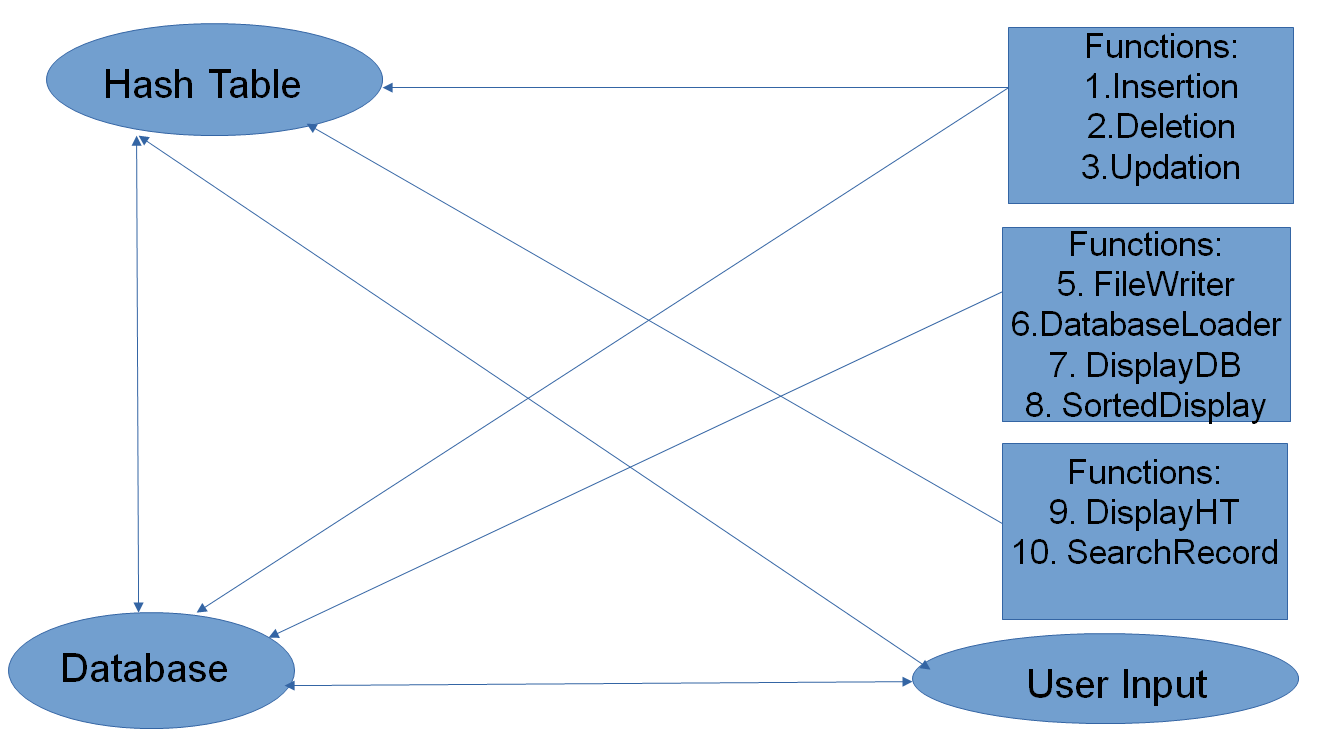
**DATA STRUCTURES USED:**

* Hash Table
* Linked List ( for Collision Resolution in Hash Table)

**ALGORITHMS USED:**

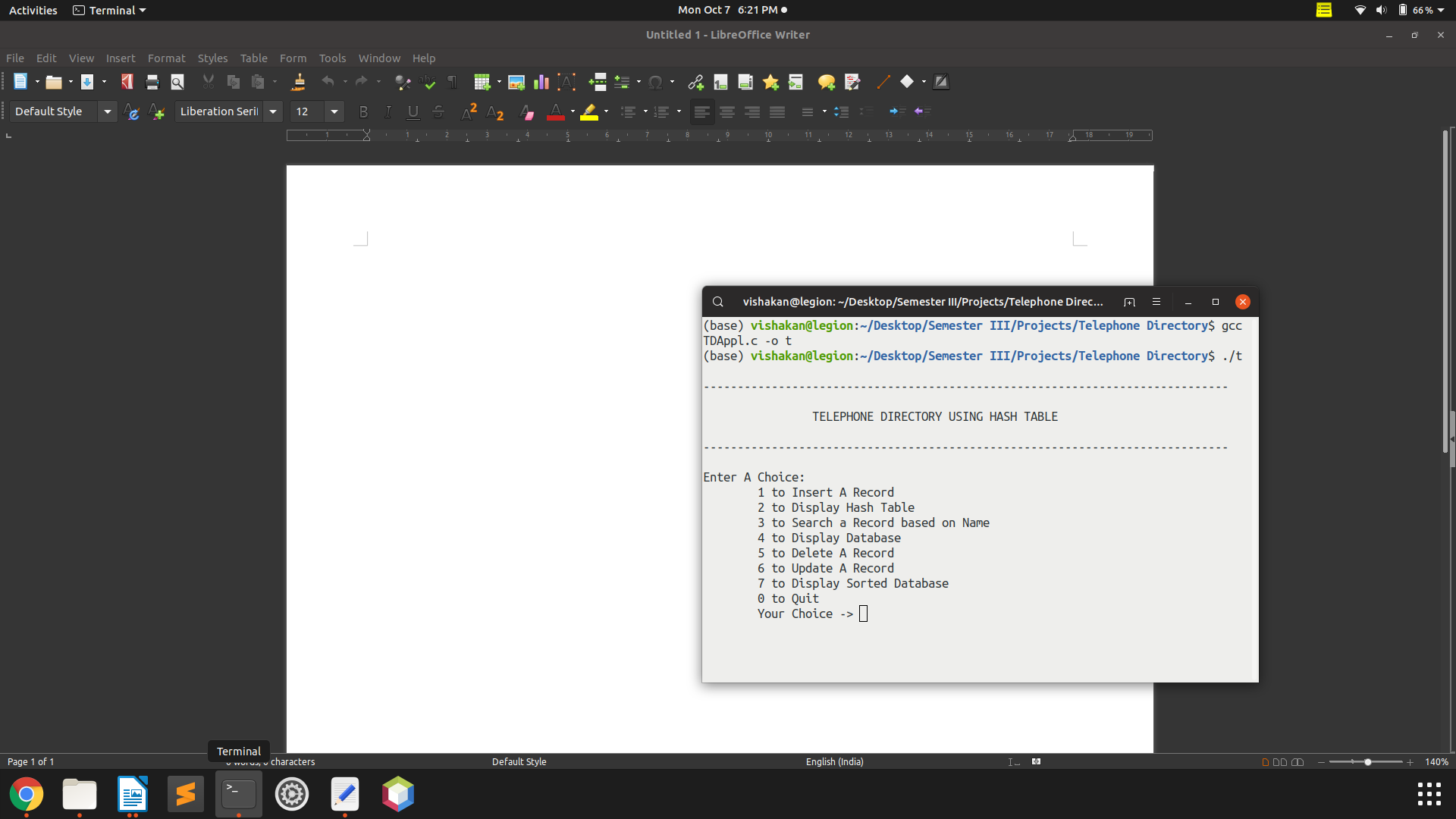
* Quick Sort

**DESIGN OF THE APPLICATION:**

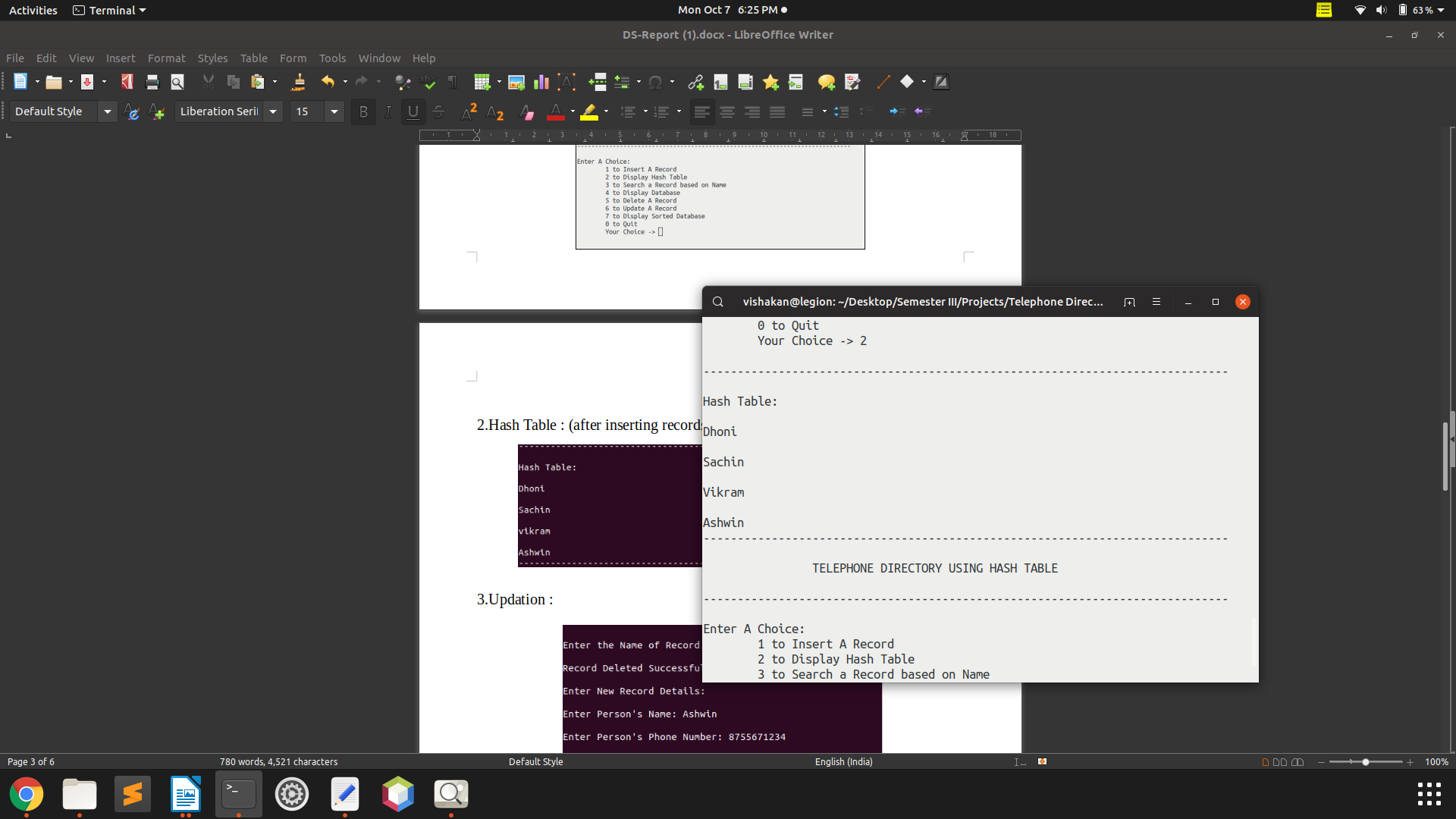


**OUTPUT SCREENSHOTS :**

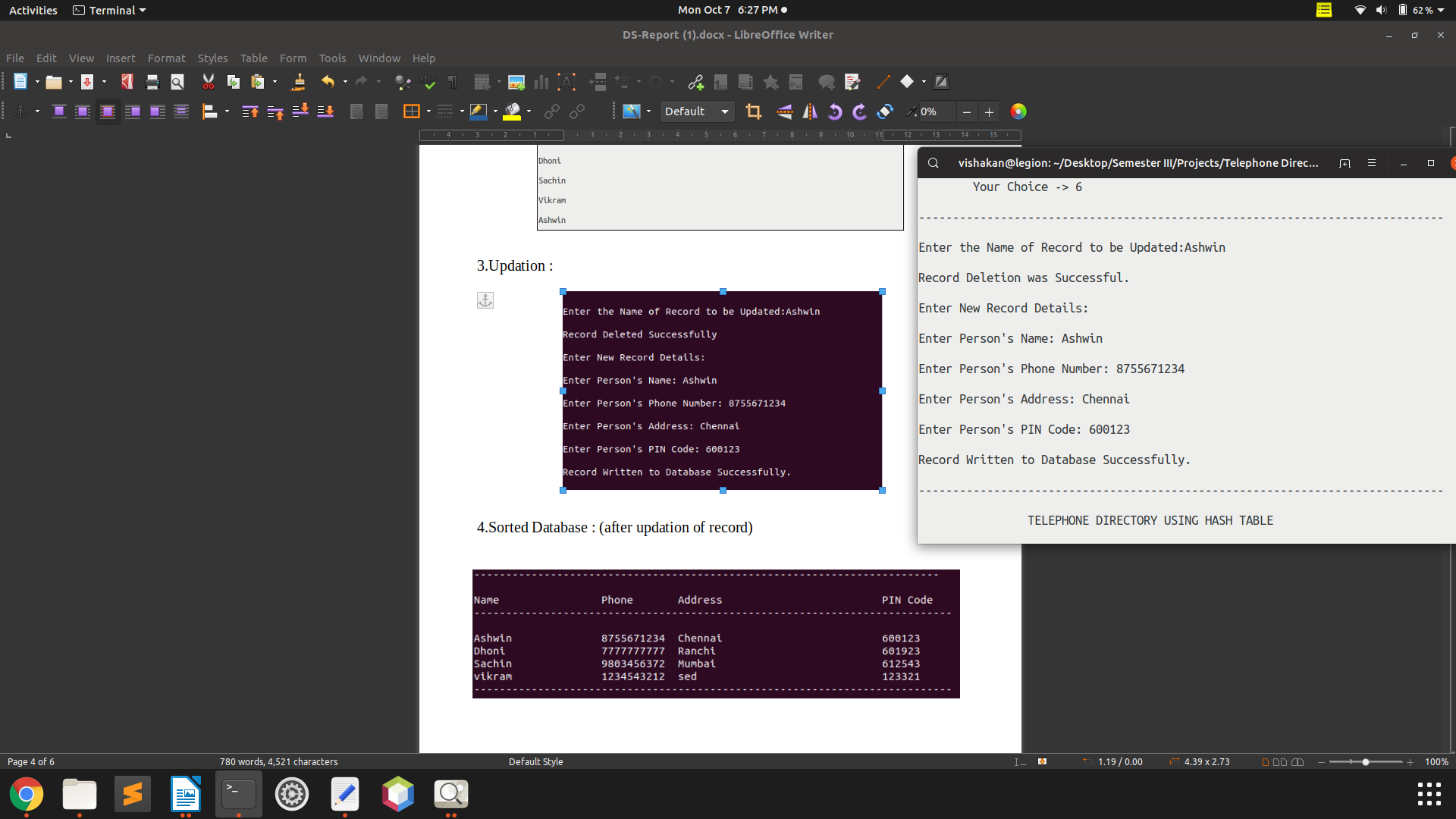
1.Options :



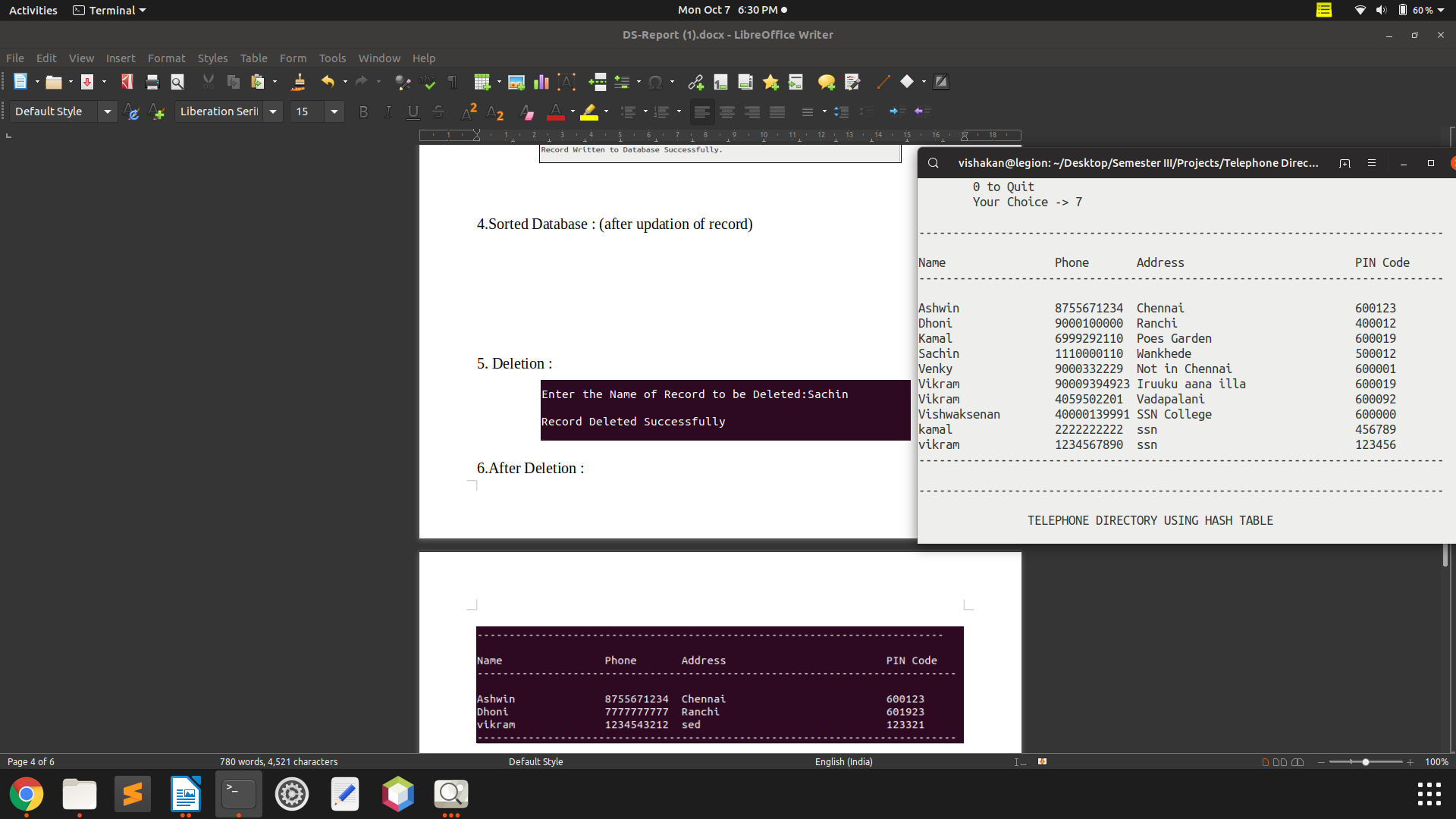
2.Hash Table : (after inserting records)



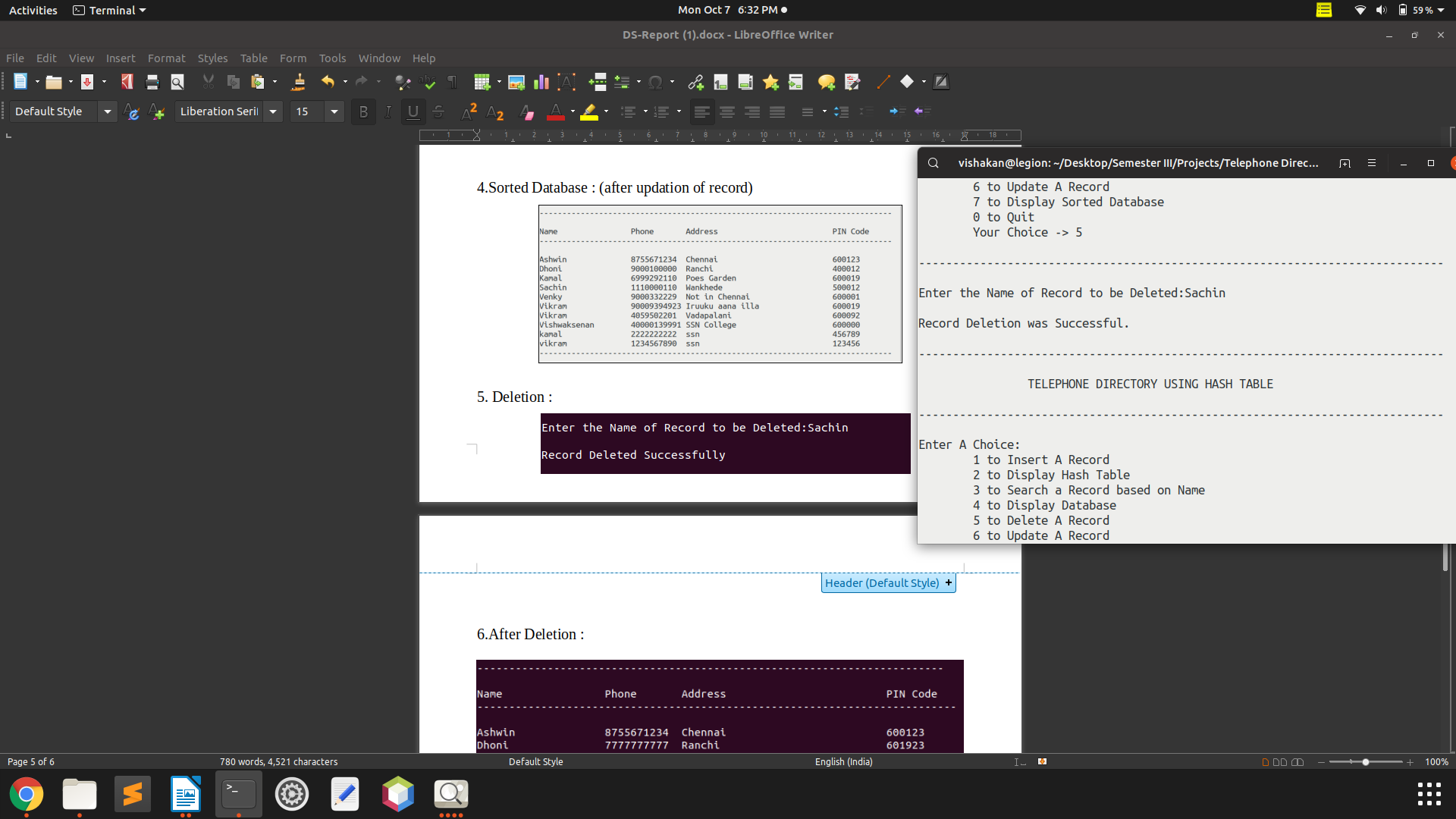
3.Updation :

****

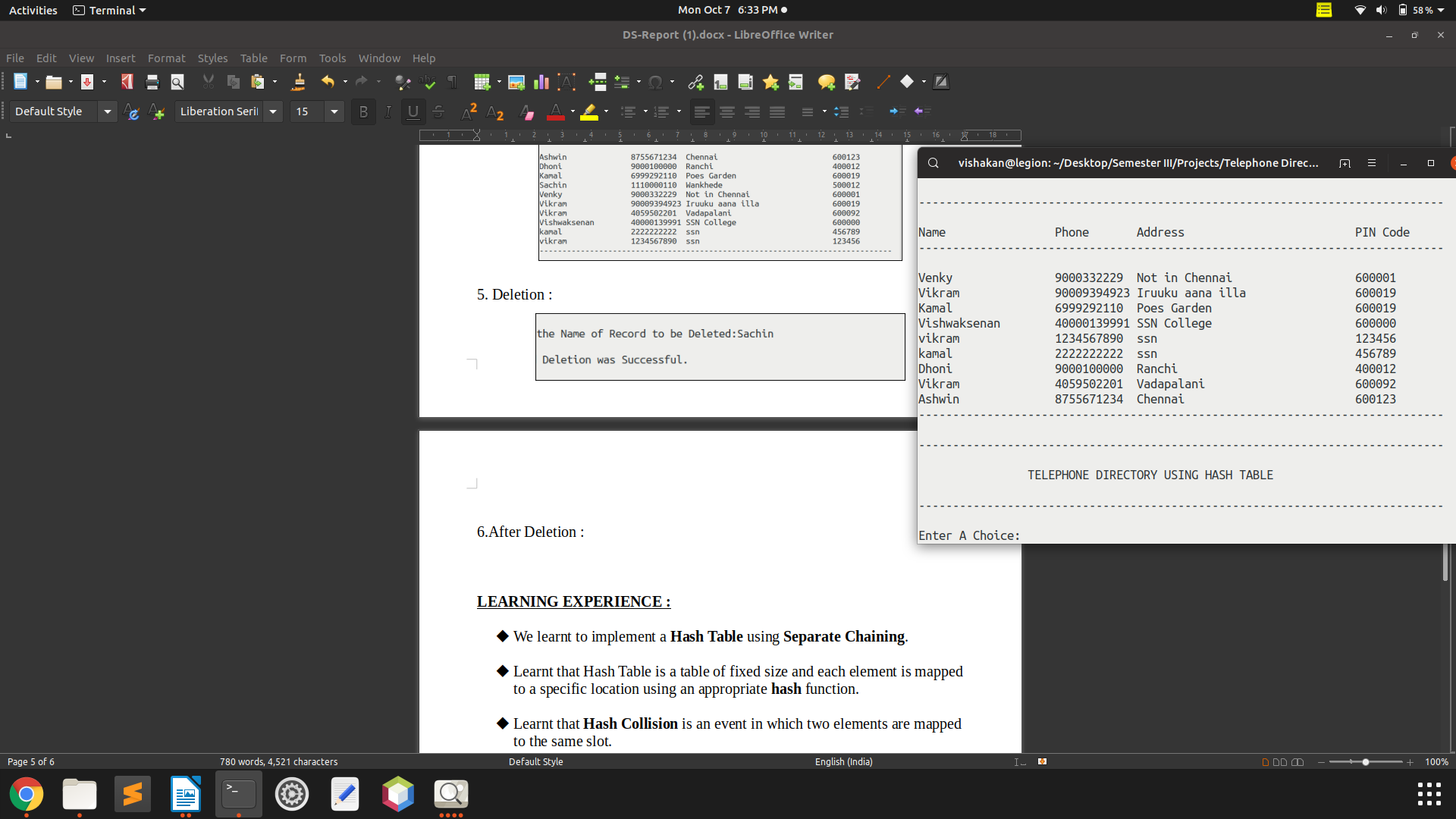
4.Sorted Database : (after updation of record)

****

5. Deletion :

****

6.After Deletion :



**LEARNING EXPERIENCE :**

* We learnt to implement a **Hash Table** using **Separate Chaining**.
* Learnt that Hash Table is a table of fixed size and each element is mapped to a specific location using an appropriate **hash** function.
* Learnt that **Hash Collision** is an event in which two elements are mapped to the same slot.
* Also learnt that this *collision* can be resolved using some strategies,which are
  + - Separate chaining
    - Open addressing

Linear probing

Quadratic probing

Double hashing

* Among these methods we chose separate chaining,

because there is no limit to the length of the chain, and we can add any number of records, as per our need. In other collision resolution strategies, the table becomes full at a point and retrieval becomes tedious.

* We stored the records inserted by the user in a database file so that they needn’t be inserted each time the program is executed. Once a record is inserted, it is stored permanently in the database, unless updated/deleted.
* The use of **Quick sort** made the sorted display of database more efficient, as it is one of the fastest sorting algorithms, with time complexity of **O(log n)**.
* Learnt to implement a database and its handling in C, using **FILE** Operations like **fread** and **fwrite.**
* Learnt to do operations like insertion, deletion, searching, and updating in a Hash Table.
* Understood concepts of Hash Table like Hash Functions, Collision Resolution etc.
* Stored records of many persons inside the Hash Table.