

# **Homework 2**

CS51510-001, Fall 2025

Purdue University Northwest

**09/19/2025**

Name	Email
Raaj Atulkumar Patel	<b>Pate2682@pnw.edu</b>

## Table of Contents

ABSTRACT.....	4
INTRODUCTION .....	5
1. Download student-data.csv .....	7
Step 1: Open Link and Download File .....	7
Step 2: Unzip and verify the data .....	8
2. Develop a "memory database" with the SINGLE linked list.....	9
Python code.....	10
i. Install required python modules .....	11
ii. Write python code .....	11
Go code.....	14
i. Install required go modules .....	15
ii. Write go code .....	15
3. The "memory database" has three major functionS .....	17
(3a) Load the student-data.csv file into the memory database through recursive function .....	17
i. Python Implementation .....	17
ii. Go Implementation .....	19
(3b) When a row is added or removed from the CSV file, the memory database will adapt to the changes.....	21
i. Python Implementation .....	21
ii. Go Implementation .....	23
(3c) Export the memory database into a csv file through recursive function .....	25
i. Python Implementation .....	25
ii. Go Implementation .....	27
HYPERLINK TO ONLINE GDB .....	29
DISCUSSION .....	30
LIMITATIONS AND FUTUREWORK .....	31
CONCLUSION .....	32
ACKNOWLEDGE .....	32
REFERENCES .....	33
APPENDIX.....	34
1. Python code .....	34
2. Go code .....	39

## Table of Figures

Figure 1 – Download student-data.csv file successfully.....	7
Figure 2 – Download and extra file and get data .....	8
Figure 3 – Terminal that shows unzip file and verify database is download is correct .....	8
Figure 4 – Make folder name homework2.....	10
Figure 5 – Move csv file from download folder to homework2.....	10
Figure 6 – Make memory_database_using_ll.py file using touch command .....	11
Figure 7 – Install cvskit for python.....	11
Figure 8 – Screenshot of python code for task2 .....	13
Figure 9 – Output of python code for task2 .....	13
Figure 10 – Download environment for go.....	14
Figure 11 – Make memory_database_using_ll.go file using touch command .....	14
Figure 12 – Screenshot of code go for task2.....	15
Figure 13 – Output of code go for task2 .....	16
Figure 14 – Screenshot of python code until task 3a .....	18
Figure 15 – Screenshot of terminal that shows code is run and print loaded linklist .....	19
Figure 16 – Screenshot of terminal that and all node is loaded .....	19
Figure 17 – screenshot of go code until 3a .....	20
Figure 18 – screenshot of output code that shows all data of csv file loaded into linklist .....	20
Figure 19 – Screenshot of python code until task 3b.....	22
Figure 20 – output of the code that shows that linklist updated as per csv file for python .....	23
Figure 21 – Screenshot of go code that shows implemented function until 3b .....	24
Figure 22 – Output of the code that shows that linklist updated as per csv file .....	25
Figure 23 – Screenshot of python code until task 3c .....	26
Figure 24 – Output that shows line removed and new file created and that is updated.....	27
Figure 25 – Screenshot of go code until task 3c .....	28
Figure 26 – Screenshot of output where go file is executed and program run .....	28
Figure 27 – output that shows 4line added and new file created and that is updated .....	29

## ABSTRACT

This homework gave me practical experience in building a small memory database from scratch using linked lists in both Python and Go. Instead of relying on built-in data structures like lists or dictionaries, I created a custom linked list to store and process rows from a CSV file. This approach made me think carefully about how data is organized in memory, how nodes are connected, and how recursive logic can be applied for inserting and exporting data.

In Python, the implementation was more straightforward thanks to its simple syntax and rich standard library. Writing recursive functions to load the CSV into the linked list and export it back to a file was easier to test and debug. Python also allowed me to add features quickly, such as detecting when the CSV file changed and automatically reloading the data, which showed me how high-level languages simplify file handling and automation.

Go, on the other hand, required more attention to type definitions and pointer management. Creating custom structs for nodes and the overall list gave me a clearer picture of how memory is handled in a more explicit way. Using pointers and struct-based design revealed the balance between low-level control and readability. Implementing traversal and updates also highlighted how Go's strict typing and explicit memory handling provide both safety and clarity. Although Go was less familiar to me than Python, completing this part of the task built my confidence in applying data structure concepts in a different programming environment.

Overall, this assignment helped me connect theoretical concepts with practical implementation. I strengthened my understanding of linked lists and recursion, while also exploring how the same idea can be expressed in two different languages. Python demonstrated speed of development and ease of testing, while Go highlighted explicit memory management and type safety. Working with both languages gave me a broader perspective and improved my ability to adapt core data structure concepts across different platforms.

## INTRODUCTION

The purpose of this homework was to gain practical experience in building a small in-memory database using linked lists and recursive logic. Instead of depending on pre-built database systems or advanced libraries, I created a custom data structure that could read student records from a CSV file, store them in memory, and then write them back to a new file. This exercise was valuable because it made me think about how data is stored, connected, and retrieved at a lower level, while still showing how these concepts can be applied to real-world problems.

For this homework, I worked with two programming languages: Python and Go. Python is known for its simple syntax and strong standard library, which makes it a natural choice for quickly implementing algorithms and testing logic. Go, on the other hand, emphasizes type safety, concurrency, and explicit memory handling. By implementing the same homework in both Python and Go, I was able to compare their strengths and see how different languages approach the same problem.

Once the environments were set up, the next step was designing the memory database itself. Each record from the CSV file was stored inside a node of a singly linked list. The program then linked the nodes together so that new records could be added, displayed, or exported. Recursion was also used to traverse through the nodes and process the list row by row. To make the database dynamic, functionality was added to reload or update the list when changes were made to the data. This showed how even a simple custom-built database can adapt to modifications in its source file.

To verify the design, I tested the program by loading the dataset, inserting records, and displaying all nodes stored in memory. Running the homework in both Python and Go confirmed that the underlying logic of linked lists remains the same, but the way the code is written changes with each language. These experiments not only deepened my understanding of linked lists and recursion but also improved my ability to adapt data structure concepts to different programming environments.

This introduction outlines the foundation for the rest of the report, which explains the process step by step: setting up the Python and Go environments, building the linked list, writing the

functions to add and display records, and testing the program with sample data. The overall goal of the assignment was not just to write working code, but to connect theory with hands-on practice and strengthen problem-solving skills across languages.

# 1. DOWNLOAD STUDENT-DATA.CSV

As the first step of the homework, I needed to download the dataset that would be used to build the memory database. The file provided was [student-data.csv](#), which contains rows of student information. This dataset served as the input for testing the linked list operations and made the homework more realistic compared to writing dummy data manually.

The professor shared a download link for the dataset, and I accessed it directly from inside the Ubuntu virtual machine. Using the browser in Ubuntu, I logged into the link and downloaded the file. Once the compressed file was saved, I used the terminal to unzip it and place the CSV in my working directory. After extraction, I verified the dataset by checking its name, size, and previewing the first few rows.

The process was completed in two simple steps. In **Step 1**, I opened the download link in Ubuntu and saved the file to the Downloads folder. In **Step 2**, I used terminal commands to unzip the file and confirm its contents. Screenshots of each step are included to show the process clearly.

## Step 1: Open Link and Download File

- Opened the professor's download link inside the Ubuntu virtual machine.
- Logged in as required and saved the compressed dataset file to the system.

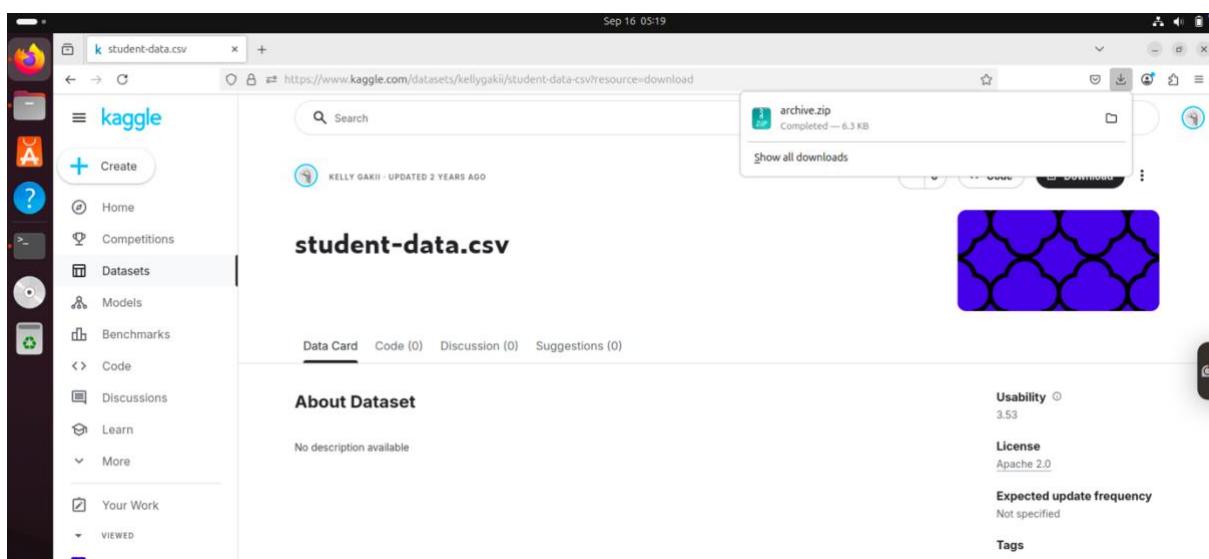
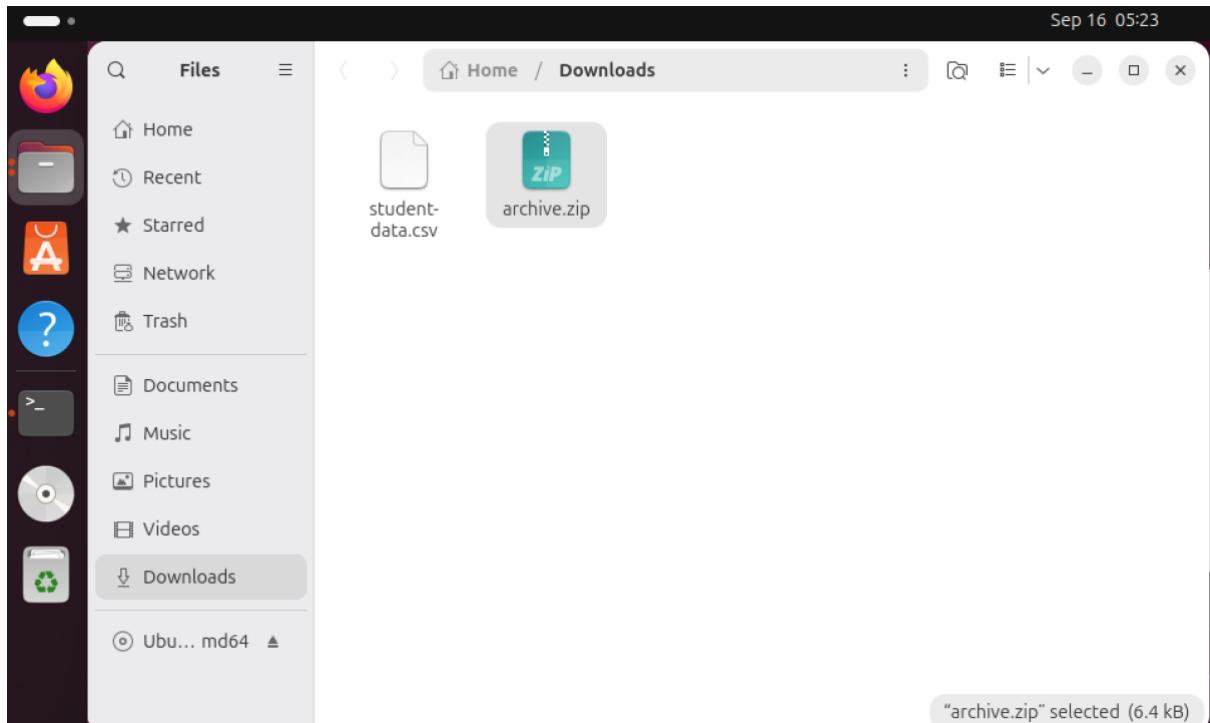


Figure 1 – Download student-data.csv file successfully

## Step 2: Unzip and verify the data

- Navigated to the folder where the file was saved.



*Figure 2 – Download and extra file and get data*

- Unzipped the downloaded file using the unzip command “unzip archive.zip”.
  - Verified the dataset by running “head -n 5 student-data.csv”. In command ‘n’ flag stands for number of rows want to see in terminal.

*Figure 3 – Terminal that shows unzip file and verify database is download is correct*

## **2. DEVELOP A "MEMORY DATABASE" WITH THE SINGLE LINKED LIST.**

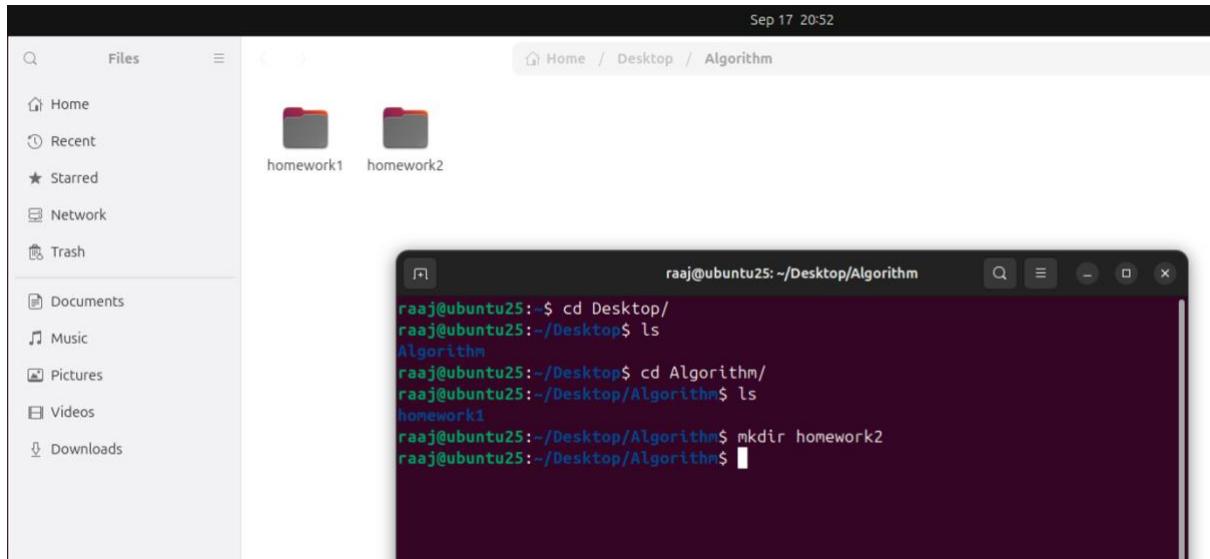
The goal of this task was to design a simple in-memory database from scratch using the concept of a singly linked list. A singly linked list is a basic data structure where each element, called a node, contains two parts: the actual data and a pointer to the next node. The list starts at a special node called the head, and every operation—whether inserting, deleting, or searching—happens by traversing node by node until the right element is found. This structure was chosen because it demonstrates the underlying mechanics of data management in the simplest form, and it is particularly useful for learning how records are connected in memory.

To keep the homework organized, I first created a dedicated folder named “homework2” under the Algorithm directory. The dataset, “student-data.csv”, was moved from the “Downloads” folder into this working folder so we could access it with relative paths instead of typing long absolute paths each time. After preparing the workspace, implemented the memory database in two languages: Python and Go, to compare how different languages handle the same logic.

In Python, I created a file “memory\_database\_using\_ll.py” where the linked list was implemented. That defined a Node class to hold data and a pointer next. To work with CSV files, we used Python’s built-in csv module, and we also installed the csvkit package for quick terminal inspection of CSV data. In Go, the same concept was applied using structs and pointers. A Node struct contained a map for row data and a pointer to the next node, while a Database struct managed the head of the list. Go’s standard encoding/csv library handled file operations.

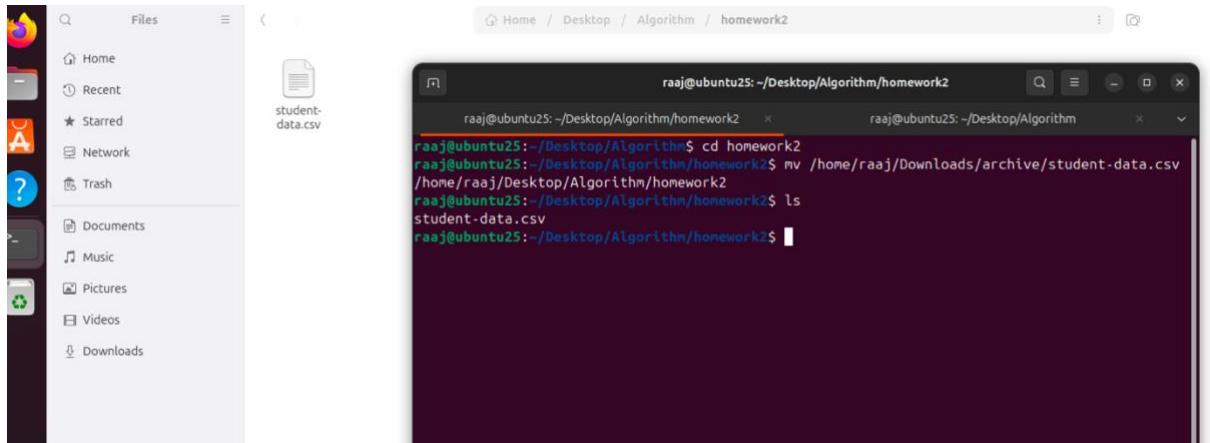
Overall, this exercise was not about building a high-performance database, but about understanding how fundamental data structures can be used to simulate database-like functionality. By walking through each node, manually managing pointers, and explicitly coding insertions and deletions, we gained practical insight into how memory is organized and how higher-level database systems operate under the hood.

- Create a folder name “homework 2” at algorithm.



*Figure 4 – Make folder name homework2*

- Move “student-data.csv” file from download folder to homework2 folder for easy to access while fetching data from csv file and need not require giving full path.



*Figure 5 – Move csv file from download folder to homework2*

## Python code

Now create python folder using “mkdir” command which stands for make directory and called that “homework2”. Then using “touch” command for creating empty file name “memory\_database\_using\_ll.py”

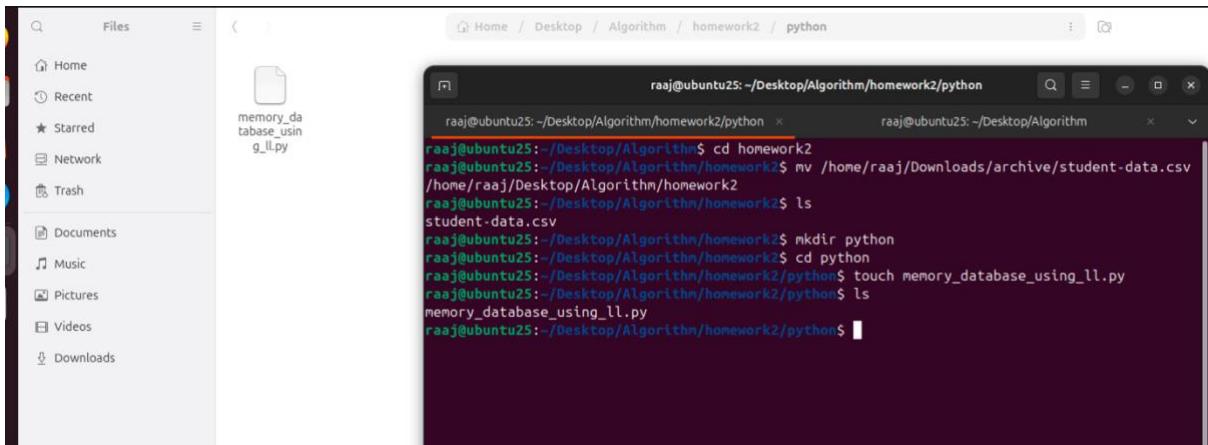


Figure 6 – Make `memory_database_using_ll.py` file using `touch` command

### i. Install required python modules

- **Csvkit** modules help to read and write csv file in python so installing csvkit by “`apt-get install python3-csvkit`” these command.

```

raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ sudo apt-get install python3-csvkit
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Solving dependencies... Done
The following additional packages will be installed:
  python3-agate python3-agatedbf python3-agateexcel python3-agatesql python3-bs4 python3-cssselect python3-dbfread python3-et-xmlfile python3-html5lib python3-icu python3-isodate python3-leather python3-lxml python3-openpyxl python3-parsedatetime python3-pytimeparse python3-soupsieve python3-sqlalchemy python3-sqlalchemy-ext python3-unidecode python3-webencodings python3-xlrd
Suggested packages:
  python-agate-doc python-agatedbf-doc python-agateexcel-doc python-agatesql-doc python-dbfread-doc python-greenlet-dev python-greenlet-doc
  python-leather-doc python-lxml-doc python-openpyxl-doc python-sqlalchemy-doc python3-asynccpg python3-pg8000 python3-psycopg2 python3-psycopg2-doc
  python3-aiosqlite python3-mariadb-connector python3-mysqldb python3-mysql.connector python3-pyodbc python3-pymssql python3-cx-oracle python3-cx-oracle-doc
The following NEW packages will be installed:
  python3-agate python3-agatedbf python3-agateexcel python3-agatesql python3-bs4 python3-cssselect python3-csvkit python3-dbfread python3-et-xmlfile python3-html5lib python3-icu python3-isodate python3-leather python3-lxml python3-openpyxl python3-parsedatetime python3-pytimeparse python3-soupsieve python3-sqlalchemy python3-sqlalchemy-ext python3-unidecode python3-webencodings python3-xlrd
0 upgraded, 25 newly installed, 0 to remove and 55 not upgraded.
Need to get 4,462 kB of archives.
After this operation, 23.6 MB of additional disk space will be used.
Do you want to continue? [Y/n] y

```

Figure 7 – Install cvskit for python

### ii. Write python code

- In this task we implemented a simple memory database using a singly linked list in Python. Each record of data is stored in a node, and all nodes are connected one after another, starting from the head of the list.
- The code shown in Figure 8 is a simple demonstration of how to build a memory database using a singly linked list.

## 1. Node Class

- The Node class is the basic building block of the list. Each node holds two things: the actual record, which in this case is a data, and a pointer called next, which links to the following node. When a node is first created, its next pointer is set to None, meaning it does not yet connect to anything. This setup makes it easy to store one row of data at a time and later link it to other nodes, forming a continuous chain.

## 2. SingleLinkedListDatabase Class

The SingleLinkedListDatabase class is responsible for managing the list of nodes. When it is initialized, the list is empty with head = None and the size set to zero. It provides two main functions:

- `add_data()`: This function inserts a new record into the list. If the list is empty, the new node becomes the head. Otherwise, the function goes through the list until it finds the last node and then attaches the new one at the end. Each time a record is added, the size counter increases by one.
- `print_data()`: This function starts at the head node and prints out the data stored in every node, moving from one to the next using the next pointer. It continues until there are no more nodes left. This makes it easy to see all the records that are currently in memory.

## 3. Sample Data and Main Program

To test the program, a few sample dictionaries were created to represent student records, with fields like roll number, name, subject, marks, and password. A database object was then created from the SingleLinkedListDatabase class. The sample records were added one by one using the `add_data()` function. Finally, the `print_data()` function was used to print all stored records, and the size attribute confirmed how many records were saved in total.

```
Open ▾ A
memory_database_using_ll.py
-Desktop\Algorithm\Homework\Python

# create nodes for single link list
class Node:
    def __init__(self,list):
        # store one row from and next node is point to none
        self.list = list
        self.next= None

#create database using single linklist
class SingleLinkedListDatabase:
    def __init__(self):
        # start with an empty single link list here head stores row ,size for length
        self.head = None
        self.size = 0

    # add a new node at the end for that we take input as current node and data
    def add_data(self, data):
        # make new node and store data
        new_node = Node(data)
        # if the list is completely empty then add head
        if self.head is None:
            self.head = new_node
        # add last node's next new node and new's next is none
        else:
            n = self.head
            while n.next is not None:
                n = n.next
            n.next = new_node

    def print_data(self):
        # start from head node and find until next is none
        n=self.head
        while n is not None:
            print(n.list)
            n = n.next

if __name__ == "__main__":
    main()

# school:{GP,F,M,18,U,GT3,A,4,4,at_home,teacher,course,mother,2,2,0,yes,no,paid,no,no,yes,higher=yes,internet,no,romantic,no,famrel:4,freetime:3,gout:4,Dalc:1,Walc:1,health:3,absences:6,passed:yes}
# sample1 = {
#     "school": "GP", "sex": "F", "age": "18", "address": "U", "famsize": "GT3", "Pstatus": "A",
#     "Medu": "4", "Fedu": "4", "Mjob": "at_home", "Fjob": "teacher", "reason": "course", "guardian": "mother",
#     "traveltime": "2", "studytime": "2", "failures": "0", "schoolsup": "yes", "famsup": "no", "paid": "no",
#     "activities": "no", "nursery": "yes", "higher": "yes", "internet": "no", "romantic": "no", "famrel": "4",
#     "freetime": "3", "gout": "4", "Dalc": "1", "Walc": "1", "health": "3", "absences": "6", "passed": "no"
# }

# sample2 = {
#     "school": "GP", "sex": "M", "age": "19", "address": "U", "famsize": "GT3", "Pstatus": "A",
#     "Medu": "4", "Fedu": "4", "Mjob": "at_home", "Fjob": "teacher", "reason": "course", "guardian": "mother",
#     "traveltime": "2", "studytime": "2", "failures": "0", "schoolsup": "yes", "famsup": "no", "paid": "no",
#     "activities": "no", "nursery": "yes", "higher": "yes", "internet": "no", "romantic": "no", "famrel": "4",
#     "freetime": "3", "gout": "4", "Dalc": "1", "Walc": "1", "health": "3", "absences": "6", "passed": "yes"
# }

# sample3 = {
#     "school": "GP", "sex": "F", "age": "17", "address": "U", "famsize": "GT3", "Pstatus": "A",
#     "Medu": "4", "Fedu": "4", "Mjob": "at_home", "Fjob": "teacher", "reason": "course", "guardian": "mother",
#     "traveltime": "2", "studytime": "2", "failures": "0", "schoolsup": "yes", "famsup": "no", "paid": "no",
#     "activities": "no", "nursery": "yes", "higher": "yes", "internet": "no", "romantic": "no", "famrel": "4",
#     "freetime": "3", "gout": "4", "Dalc": "1", "Walc": "1", "health": "3", "absences": "6", "passed": "yes"
# }

# create an empty linked list database
db = SingleLinkedListDatabase()
# load sample data
db.add_data(sample1)
db.add_data(sample2)
db.add_data(sample3)
# print data
db.print_data()
print(f"Total node: {db.size}")

if __name__ == "__main__":
    main()
```

*Figure 8 – Screenshot of python code for task2*

- Output of code displays the result after running the program. The terminal output shows each student record that was added to the linked list, printed one after another as the `add_data()` function traverses through the nodes. At the bottom, the program also prints the total number of nodes, confirming that all the sample records were successfully stored in memory. This output verifies that the linked list structure works correctly for adding and retrieving student data.

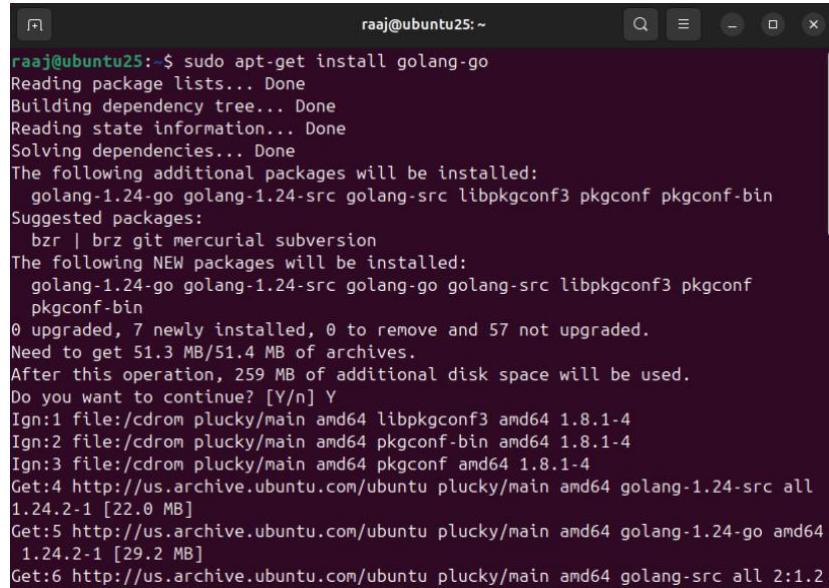
```
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ cd python/
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ python3 memory_database_using_ll.py

{'school': 'GP', 'sex': 'F', 'age': '18', 'address': 'U', 'famsize': 'GT3', 'Pstatus': 'A', 'Medu': '4', 'Fedu': '4', 'Mjob': 'at_home', 'Fjob': 'teacher', 'reason': 'course', 'guardian': 'mother', 'traveltime': '2', 'studytme': '2', 'failures': '0', 'schoolsup': 'yes', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'internet': 'no', 'romantic': 'no', 'famrel': '4', 'freetime': '3', 'goout': '4', 'Dalc': '1', 'Walc': '1', 'health': '3', 'absences': '6', 'passed': 'no'}
{'school': 'GP', 'sex': 'M', 'age': '19', 'address': 'U', 'famsize': 'GT3', 'Pstatus': 'A', 'Medu': '4', 'Fedu': '4', 'Mjob': 'at_home', 'Fjob': 'teacher', 'reason': 'course', 'guardian': 'mother', 'traveltime': '2', 'studytme': '2', 'failures': '0', 'schoolsup': 'yes', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'internet': 'no', 'romantic': 'no', 'famrel': '4', 'freetime': '3', 'goout': '4', 'Dalc': '1', 'Walc': '1', 'health': '3', 'absences': '6', 'passed': 'yes'}
{'school': 'GP', 'sex': 'F', 'age': '17', 'address': 'U', 'famsize': 'GT3', 'Pstatus': 'A', 'Medu': '4', 'Fedu': '4', 'Mjob': 'at_home', 'Fjob': 'teacher', 'reason': 'course', 'guardian': 'mother', 'traveltime': '2', 'studytme': '2', 'failures': '0', 'schoolsup': 'yes', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'internet': 'no', 'romantic': 'no', 'famrel': '4', 'freetime': '3', 'goout': '4', 'Dalc': '1', 'Walc': '1', 'health': '3', 'absences': '6', 'passed': 'yes'}
Total node: 3
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$
```

*Figure 9 – Output of python code for task2*

## Go code

- Install go in ubuntu by using command “sudo apt-get install golang-go”.



```
raaj@ubuntu25: $ sudo apt-get install golang-go
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Solving dependencies... Done
The following additional packages will be installed:
  golang-1.24-go golang-1.24-src golang-src libpkgconf3 pkgconf pkgconf-bin
Suggested packages:
  bzr | brz git mercurial subversion
The following NEW packages will be installed:
  golang-1.24-go golang-1.24-src golang-go golang-src libpkgconf3 pkgconf
  pkgconf-bin
0 upgraded, 7 newly installed, 0 to remove and 57 not upgraded.
Need to get 51.3 MB/51.4 MB of archives.
After this operation, 259 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Ign:1 file:/cdrom plucky/main amd64 libpkgconf3 amd64 1.8.1-4
Ign:2 file:/cdrom plucky/main amd64 pkgconf-bin amd64 1.8.1-4
Ign:3 file:/cdrom plucky/main amd64 pkgconf amd64 1.8.1-4
Get:4 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 golang-1.24-src all
 1.24.2-1 [22.0 MB]
Get:5 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 golang-1.24-go amd64
 1.24.2-1 [29.2 MB]
Get:6 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 golang-src all 2:1.2
```

Figure 10 – Download environment for go

- Now create go folder using “mkdir” command which stands for make directory and called that “homework2”. Then using “touch” command for creating empty file name “memory\_database\_using\_ll.go”.

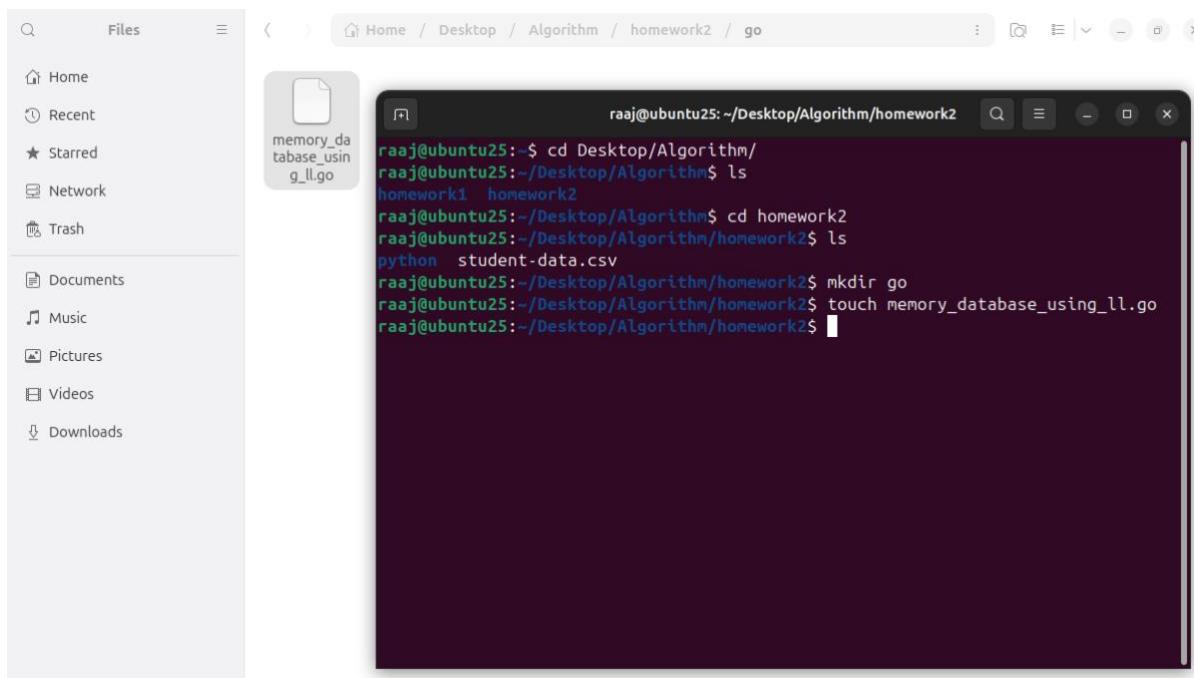


Figure 11 – Make memory\_database\_using\_ll.go file using touch command

## i. Install required go modules

The go module that are required for this code is already available while installing so not require to download any other modules.

## ii. Write go code

In this I implemented a simple memory database in Go using a singly linked list. Each record was stored inside a Node struct, which contains two parts: a map for the student data and a pointer to the next node. When a new node is created, its pointer is set to nil, and it is later connected to the chain of nodes that begins at the head of the list. To manage all the nodes, I created a SingleLinkedListDatabase struct with two fields: head, which marks the start of the list, and size, which keeps track of how many records are stored.

This struct also provides two main methods. The first one, addData, is used to insert new records. If the list is empty, the new node becomes the head; otherwise, the method walks through the list until it finds the last node and attaches the new node at the end, increasing the size count. The second method, printData, starts from the head and follows the pointers from node to node, printing the data until the end of the list is reached.

The screenshot shows a code editor with a file named `memory_database_using_ll.go`. The code implements a singly linked list database using Go's standard library. It includes a `Node` struct for storing student data (represented as a map) and a pointer to the next node. A `SingleLinkedListDatabase` struct manages the list, keeping track of the head and size. The `addData` method adds a new node to the end of the list. The `printData` method prints all nodes in order. Three sample records are defined as maps:

```
func main() {
    // sample data
    sample1 := map[string]string{
        "school": "GP", "sex": "F", "age": "18", "address": "U", "famsize": "GT3", "Pstatus": "A",
        "Medu": "4", "Fedu": "4", "Mjob": "at_home", "Fjob": "teacher", "reason": "course", "guardian": "mother",
        "traveltime": "2", "studtime": "2", "failures": "0", "schoolsup": "yes", "famsup": "no", "paid": "no",
        "activities": "no", "nursery": "yes", "higher": "yes", "internet": "no", "romantic": "no", "famrel": "4",
        "freetime": "3", "gout": "4", "Dalc": "1", "Walc": "1", "health": "3", "absences": "6", "passed": "no",
    }

    sample2 := map[string]string{
        "school": "GP", "sex": "M", "age": "19", "address": "U", "famsize": "GT3", "Pstatus": "A",
        "Medu": "4", "Fedu": "4", "Mjob": "at_home", "Fjob": "teacher", "reason": "course", "guardian": "mother",
        "traveltime": "2", "studtime": "2", "failures": "0", "schoolsup": "yes", "famsup": "no", "paid": "no",
        "activities": "no", "nursery": "yes", "higher": "yes", "internet": "no", "romantic": "no", "famrel": "4",
        "freetime": "3", "gout": "4", "Dalc": "1", "Walc": "1", "health": "3", "absences": "6", "passed": "yes",
    }

    sample3 := map[string]string{
        "school": "GP", "sex": "F", "age": "17", "address": "U", "famsize": "GT3", "Pstatus": "A",
        "Medu": "4", "Fedu": "4", "Mjob": "at_home", "Fjob": "teacher", "reason": "course", "guardian": "mother",
        "traveltime": "2", "studtime": "2", "failures": "0", "schoolsup": "yes", "famsup": "no", "paid": "no",
        "activities": "no", "nursery": "yes", "higher": "yes", "internet": "no", "romantic": "no", "famrel": "4",
        "freetime": "3", "gout": "4", "Dalc": "1", "Walc": "1", "health": "3", "absences": "6", "passed": "yes",
    }
}
```

Figure 12 – Screenshot of code go for task2

To test the program, I created a few sample records using Go maps with fields like roll number, name, subject, marks, and pass status. These records were added one by one to the database using `addData`. Finally, I called `printData` to display the records in order, followed by printing

the size of the list to confirm how many nodes were stored in memory. The output showed each record clearly and ended with the total node count, proving that the singly linked list was functioning correctly. This implementation demonstrates how linked lists can be built and traversed in Go using structs and pointers, achieving the same behaviour as the Python version but with stricter typing and explicit memory handling.

Output of code shows the result after running the Go program. The terminal prints each student record that was inserted into the linked list, one after another, as the printData() function traverses through the nodes. At the end, the program also displays the total number of nodes, confirming that all sample records were added successfully. This output demonstrates that the linked list in Go is functioning as intended, with nodes linked correctly, and verifies that data can be stored, traversed, and retrieved from memory in sequence.

A screenshot of a terminal window titled "raaj@ubuntu25: ~/Desktop/Algorithm/homework2/go". The window contains the command "go run memory\_database\_using\_ll.go" followed by three map structures representing student records. Each map has fields like Dalc, Fedu, Fjob, teacher, Medu, Mjob, at\_home, Pstatus, Walc, absences, activities, address, age, failures, famrel, famsize, GT3, fmsup, freetime, goout, guardian, mother, health, higher, internet, nursery, yes, paid, no, passed, reason, course, romantic, school, GP, schoolsup, yes, sex, F, studytime, 2, traveltim, 2. The final line shows "Total node: 3".

```
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/go
raaj@ubuntu25:~/Desktop/Algorithm/homework2/go$ go run memory_database_using_ll.go
map[Dalc:1 Fedu:4 Fjob:teacher Medu:4 Mjob:at_home Pstatus:A Walc:1 absences:6 activities:no address:U age:18 failures :0 famrel:4 famsize:GT3 fmsup:no freetime:3 goout:4 guardian:mother health:3 higher:yes internet:no nursery:yes paid: no passed:no reason:course romantic:no school:GP schoolsup:yes sex:F studytime:2 traveltim:2]
map[Dalc:1 Fedu:4 Fjob:teacher Medu:4 Mjob:at_home Pstatus:A Walc:1 absences:6 activities:no address:U age:19 failures :0 famrel:4 famsize:GT3 fmsup:no freetime:3 goout:4 guardian:mother health:3 higher:yes internet:no nursery:yes paid: no passed:yes reason:course romantic:no school:GP schoolsup:yes sex:M studytime:2 traveltim:2]
map[Dalc:1 Fedu:4 Fjob:teacher Medu:4 Mjob:at_home Pstatus:A Walc:1 absences:6 activities:no address:U age:17 failures :0 famrel:4 famsize:GT3 fmsup:no freetime:3 goout:4 guardian:mother health:3 higher:yes internet:no nursery:yes paid: no passed:yes reason:course romantic:no school:GP schoolsup:yes sex:F studytime:2 traveltim:2]
Total node: 3
raaj@ubuntu25:~/Desktop/Algorithm/homework2/go$
```

Figure 13 – Output of code go for task2

### **3. THE "MEMORY DATABASE" HAS THREE MAJOR FUNCTIONS**

When we start building the memory database, the very first step is to take the CSV file and read its contents into memory. Each row in the CSV represents one student record, and we want to transform each of those rows into a node in our linked list. The list grows one node at a time, with every node holding a single student's details and a pointer to the next. This gives us a structure that mirrors the file but lives completely in memory.

The database then provides three main functions that make it useful and dynamic. The first function is loading data from CSV into the linked list. This is done recursively: the program reads a row, creates a node for it, attaches the node to the chain, and then calls itself for the next row. This continues until every row has been turned into a node, so the in-memory database is an exact reflection of the CSV file.

The second function is adapting to changes in the CSV file. Real data is not static, so rows may be added, deleted, or updated. To handle this, the program re-reads the CSV and compares it with the current linked list. If a row is missing, its node is removed. If a row's values have changed, the corresponding node is updated. If a completely new row appears, a new node is created and linked in. This ensures that the memory database always stays in sync with the external CSV, even after edits.

The third function is exporting the linked list back into a CSV file. This is done by traversing the list node by node and writing each record into a new file. The export process also makes use of recursion, with each call writing one row and moving to the next until the end is reached. The result is a fresh CSV that contains the most up-to-date version of the in-memory database, ready to be inspected or used again later.

**(3a) Load the student-data.csv file into the memory database through recursive function**

#### **i. Python Implementation**

The process starts in the `ReadAndWriteCSV.read_csv(path)` function. This function uses Python's `csv.DictReader` to read the file and convert each line into a dictionary, where the column headers are the keys. It also assigns a unique "key" value to each row so that every record can be identified later. It finally returns two things: a list of row dictionaries and the headers.

Once the rows are prepared, the `SingleLinkedListDatabase.load_data(rows, i=0)` method is called. This is where recursion comes in. The function checks:

1. If  $i$  is greater than or equal to the number of rows, recursion stops (base case).
  2. Otherwise, it calls `self.add_data(rows[i])` to insert the current row as a new node at the end of the list.
  3. Then it calls itself again with  $i+1$ , moving to the next row.

Step by step, the recursion creates one node for each student record until all rows are processed. At the end, the linked list in memory mirrors the CSV file completely.

```
Open ▾  memory_database_using_ll.py  ~/Desktop/Algorithm/homework2/python

import csv
# create nodes for single link list
class Node:
    def __init__(self,list):
        # store one row from and next node is point to none
        self.list = list
        self.next= None

#create database using single linklist
class SingleLinkedListDatabase:
    def __init__(self):
        # start with an empty single link list here head stores row ,size for length
        self.head = None
        self.size = 0

    # add a new node at the end for that we take input as current node and data
    def add_data(self, data):
        # make new node and store data
        new_node = Node(data)
        # if the list is completely empty then add head
        if self.head is None:
            self.head = new_node
        # add last node's next new node and new's next is none
        else:
            n = self.head
            while n.next is not None:
                n = n.next
            n.next = new_node
            self.size += 1

    # load data row by row into link list and default index value is 0
    def load_data(self,rows,i=0):
        # stop if index is greater or equal to number of rows then return
        if i >= len(rows):
            return
        # add row to link list and increase index to one for next
        self.add_data(rows[i])
        self.load_data(rows, i + 1)

    # print database loaded in link list
    def print_data(self):
        # start from head node and find until next is none
        n=self.head
        while n is not None:
            print(n.list)
            n = n.next

    # read and write for CSV file for that make class
class ReadAndWriteCSV:
    def read_csv(path):
        # rows for data and headers for column name
        rows = []
        headers = None
        # open the CSV file and read line as dictionary and first line for header
```

```
with open(path, newline="") as f:
    reader = csv.DictReader(f)
    headers = reader.fieldnames
    # add a indexing key for identify row
    i = 1
    for r in reader:
        r["key"] = str(i)
        rows.append(r)
        i += 1
    # check that key is there or not
    if "key" not in headers:
        headers = ["key"] + headers
    # return the list that contain data
    return rows, headers

# main function
def main():
    # read csv file rows is data and header is all column name
    rows,header= ReadAndWriteCSV.read_csv("../student-data.csv")
    # use lambda function for data so that add first header and then other data
    rows = [(h: h for h in header)] + rows
    # create an empty linked list database
    db = SingleLinkedListDatabase()
    # recursive load data
    db.load_data(rows, 0)
    # print data
    db.print_data()
    print(f"Total node: {db.size}")

if __name__ == "__main__":
    main()
```

*Figure 14 – Screenshot of python code until task 3a*

- Output of code after the program runs, the `add_data()` function keeps inserting each student record into the linked list one by one. After all the data is loaded, the `print_data()` function is called. This function begins at the head of the list and moves through every node until the end, printing the data stored in each node along the way. That can see in figure 15 and 16. Where 396 rows loaded in single linklist.

```

raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ python3 memory_database_using_ll.py
[{'school': 'GP', 'sex': 'F', 'age': '18', 'address': 'U', 'famsize': 'GT3', 'Pstatus': 'A', 'Medu': '4', 'Fedu': '4', 'M
job': 'at_home', 'Fjob': 'teacher', 'reason': 'course', 'guardian': 'mother', 'traveltime': '2', 'studytime': '2', 'fail
ures': '0', 'schoolsups': 'yes', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'in
ternet': 'no', 'romantic': 'no', 'famrel': '4', 'freetime': '3', 'goout': '4', 'Dalc': '1', 'Walc': '1', 'health': '3',
'absences': '6', 'passed': 'no', 'key': '1'}
[{'school': 'GP', 'sex': 'F', 'age': '17', 'address': 'U', 'famsize': 'GT3', 'Pstatus': 'T', 'Medu': '1', 'Fedu': '1', 'M
job': 'at_home', 'Fjob': 'other', 'reason': 'course', 'guardian': 'father', 'traveltime': '1', 'studytime': '2', 'failur
es': '0', 'schoolsups': 'no', 'famsup': 'yes', 'paid': 'no', 'activities': 'no', 'nursery': 'no', 'higher': 'yes', 'inter
net': 'yes', 'romantic': 'no', 'famrel': '5', 'freetime': '3', 'goout': '3', 'Dalc': '1', 'Walc': '1', 'health': '3', 'a
bsences': '4', 'passed': 'no', 'key': '2'}
[{'school': 'GP', 'sex': 'F', 'age': '15', 'address': 'U', 'famsize': 'LE3', 'Pstatus': 'T', 'Medu': '1', 'Fedu': '1', 'M
job': 'at_home', 'Fjob': 'other', 'reason': 'other', 'guardian': 'mother', 'traveltime': '1', 'studytime': '2', 'failure
s': '3', 'schoolsups': 'yes', 'famsup': 'no', 'paid': 'yes', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'inte
rnet': 'yes', 'romantic': 'no', 'famrel': '4', 'freetime': '4', 'goout': '1', 'Dalc': '3', 'Walc': '4', 'health': '5', 'a
bsences': '0', 'passed': 'yes', 'key': '394'}
[{'school': 'MS', 'sex': 'M', 'age': '18', 'address': 'R', 'famsize': 'LE3', 'Pstatus': 'T', 'Medu': '3', 'Fedu': '2', 'M
job': 'services', 'Fjob': 'other', 'reason': 'course', 'guardian': 'mother', 'traveltime': '3', 'studytime': '1', 'failu
res': '0', 'schoolsups': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'no', 'higher': 'yes', 'inter
net': 'yes', 'romantic': 'no', 'famrel': '4', 'freetime': '4', 'goout': '1', 'Dalc': '3', 'Walc': '4', 'health': '5',
'absences': '0', 'passed': 'yes', 'key': '395'}
Total node: 396
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ 

```

Figure 15 – Screenshot of terminal that shows code is run and print loaded linklist

```

[{'school': 'MS', 'sex': 'M', 'age': '18', 'address': 'R', 'famsize': 'LE3', 'Pstatus': 'T', 'Medu': '3', 'Fedu': '2', 'M
job': 'services', 'Fjob': 'other', 'reason': 'course', 'guardian': 'mother', 'traveltime': '3', 'studytime': '1', 'failu
res': '0', 'schoolsups': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'no', 'higher': 'yes', 'inter
net': 'yes', 'romantic': 'no', 'famrel': '4', 'freetime': '4', 'goout': '1', 'Dalc': '3', 'Walc': '4', 'health': '5',
'absences': '0', 'passed': 'yes', 'key': '394'}
[{'school': 'MS', 'sex': 'M', 'age': '19', 'address': 'U', 'famsize': 'LE3', 'Pstatus': 'T', 'Medu': '1', 'Fedu': '1', 'M
job': 'other', 'Fjob': 'at_home', 'reason': 'course', 'guardian': 'father', 'traveltime': '1', 'studytime': '1', 'failur
es': '0', 'schoolsups': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'inter
net': 'yes', 'romantic': 'no', 'famrel': '3', 'freetime': '2', 'goout': '3', 'Dalc': '3', 'Walc': '3', 'health': '5', 'a
bsences': '5', 'passed': 'no', 'key': '395'}
Total node: 396
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ 

```

Figure 16 – Screenshot of terminal that and all node is loaded

## ii. Go Implementation

In Go, the reading is handled by ReadAndWriteCSV\_read\_csv(path). This function uses Go's encoding/csv package to open the file and parse the rows. It then builds a map[string]string for each record, again attaching a "key" field so that rows can be uniquely identified. The function returns both the slice of row maps and the headers. After loading the rows, the recursive part happens inside the method SingleLinkedListDatabase.load\_data(rows []map[string]string, i int).

Just like in Python, it works by checking:

- 1.If i has reached the total number of rows, the recursion stops.
- 2.Otherwise, it calls db.add\_data(rows[i]) to add the current row as a new node.
- 3.Then it calls db.load\_data(rows, i+1) to continue with the next index.
- 4.This recursion continues until all rows are added. Each record is turned into a Node struct with a map[string]string for the row and a pointer to the next node.

```
// read and write for csv file make class (here struct with functions)
type ReadAndWriteCSV struct{}

// read_csv opens csv file, returns rows and headers
func (rw *ReadAndWriteCSV) read_csv(path string) ([]map[string]string, []string, error) {
    // open CSV file
    file, err := os.Open(path)
    if err != nil {
        return nil, nil, err
    }
    defer file.Close()

    reader := csv.NewReader(file)
    // read all rows
    allRows, err := reader.ReadAll()
    if err != nil {
        return nil, nil, err
    }

    // first line is header
    headers := allRows[0]
    rows := []map[string]string{}

    // read each line and convert to map
    for i, row := range allRows[1:] {
        data := map[string]string{'key': fmt.Sprintf("%d", i+1)}
        for j, h := range headers {
            data[h] = row[j]
        }
        rows = append(rows, data)
    }

    // check that key is there or not
    found := false
    for _, h := range headers {
        if h == "key" {
            found = true
            break
        }
    }
    if !found {
        headers = append([]string{"key"}, headers...)
    }

    return rows, headers, nil
}

// read_and_write_csv opens csv file, returns rows and headers
func (rw *ReadAndWriteCSV) read_and_write_csv(path string) ([]map[string]string, []string, error) {
    // open CSV file
    file, err := os.Open(path)
    if err != nil {
        return nil, nil, err
    }
    defer file.Close()

    reader := csv.NewReader(file)
    // read all rows
    allRows, err := reader.ReadAll()
    if err != nil {
        return nil, nil, err
    }

    // first line is header
    headers := allRows[0]
    rows := []map[string]string{}

    // read each line and convert to map
    for i, row := range allRows[1:] {
        data := map[string]string{'key': fmt.Sprintf("%d", i+1)}
        for j, h := range headers {
            data[h] = row[j]
        }
        rows = append(rows, data)
    }

    // check that key is there or not
    found := false
    for _, h := range headers {
        if h == "key" {
            found = true
            break
        }
    }
    if !found {
        headers = append([]string{"key"}, headers...)
    }

    return rows, headers, nil
}

// load data row by row into linked list and default index value is 0
func (db *SingleLinkedListDatabase) load_data(rows []map[string]string, i int) {
    // stop if index is greater or equal to number of rows then return
    if i >= len(rows) {
        return
    }
    // add row to linked list and increase index by one for next
    db.add_data(rows[i])
    db.load_data(rows, i+1)
}

// main function
func main() {
    // read csv file rows is data and header is all column name
    rw := ReadAndWriteCSV{}
    rows, headers, err := rw.read_csv("../student-data.csv")
    if err != nil {
        fmt.Println("Error:", err)
        return
    }

    // add first header row into data
    headerRow := map[string]string{}
    for _, h := range headers {
        headerRow[h] = h
    }
    rows = append([]map[string]string{headerRow}, rows...)

    // create an empty linked list database
    db := SingleLinkedListDatabase{}
    // recursive load data
    db.load_data(rows, 0)
    // print data
    db.print_data()
    fmt.Printf("Total node: %d\n", db.size)
}
```

*Figure 17 – screenshot of go code until 3a*

Output of code after running the Go program shows how the linked list is populated and traversed. The `addData()` method is called repeatedly to insert each student record, creating a new node for every row in the dataset. Once all rows are loaded into memory, the `printData()` method starts from the head of the list and moves through each node, printing the contents until it reaches the end. This sequential traversal ensures that every record stored in memory is displayed in the terminal. As shown in Figure 18 all 396 rows from the CSV file are successfully added to the singly linked list and printed out one by one, confirming that the Go implementation works correctly for both insertion and traversal.

```
raaj@ubuntu25:~/Desktop/Algorithm/homework2/go$ go run memory_database_using_ll.go
map[Dalc:Dalc Fedu:Fedu Fjob:Fjob Medu:Medu Mjob:Mjob Pstatus:Pstatus Walc:Walc absences:absences activities:activities address:address age:age failures:failure famrel:famrel famsize:famsize fansup:fansup freetime:freetime gout:gout guardian:guardian health:health higher:higher internet:internet key:key nursery:nursery paid:paid passed:passed reason:reason romantic:romantic school:school schoolup:schoolup sex:sex studytime:studytime traveltime:traveltime]
map[Dalc1:Fedu1:Fedu1 Fjob1:Fjob1 Medu1:Medu1 Mjob1:Mjob1 Pstatus1:Pstatus1 Walc1:Walc1 absences1:absences1 activities1:activities1 address1:address1 age1:age1 failures1:failure1 famrel1:famrel1 famsize1:famsize1 fansup1:fansup1 freetime1:freetime1 gout1:gout1 guardian1:guardian1 health1:health1 higher1:higher1 internet1:internet1 key1:key1 nursery1:nursery1 paid1:paid1 passed1:passed1 reason1:reason1 romantic1:romantic1 school1:school1 schoolup1:schoolup1 sex1:sex1 studytime1:studytime1 traveltime1:traveltime1]
map[Dalc2:Fedu2:Fedu2 Fjob2:Fjob2 Medu2:Medu2 Mjob2:Mjob2 Pstatus2:Pstatus2 Walc2:Walc2 absences2:absences2 activities2:activities2 address2:address2 age2:age2 failures2:failure2 famrel2:famrel2 famsize2:famsize2 fansup2:fansup2 freetime2:freetime2 gout2:gout2 guardian2:guardian2 health2:health2 higher2:higher2 internet2:internet2 key2:key2 nursery2:nursery2 paid2:paid2 passed2:passed2 reason2:reason2 romantic2:romantic2 school2:school2 schoolup2:schoolup2 sex2:sex2 studytime2:studytime2 traveltime2:traveltime2]
map[Dalc3:Fedu3:Fedu3 Fjob3:Fjob3 Medu3:Medu3 Mjob3:Mjob3 Pstatus3:Pstatus3 Walc3:Walc3 absences3:absences3 activities3:activities3 address3:address3 age3:age3 failures3:failure3 famrel3:famrel3 famsize3:famsize3 fansup3:fansup3 freetime3:freetime3 gout3:gout3 guardian3:guardian3 health3:health3 higher3:higher3 internet3:internet3 key3:key3 nursery3:nursery3 paid3:paid3 passed3:passed3 reason3:reason3 romantic3:romantic3 school3:school3 schoolup3:schoolup3 sex3:sex3 studytime3:studytime3 traveltime3:traveltime3]
map[Dalc4:Fedu4:Fedu4 Fjob4:Fjob4 Medu4:Medu4 Mjob4:Mjob4 Pstatus4:Pstatus4 Walc4:Walc4 absences4:absences4 activities4:activities4 address4:address4 age4:age4 failures4:failure4 famrel4:famrel4 famsize4:famsize4 fansup4:fansup4 freetime4:freetime4 gout4:gout4 guardian4:guardian4 health4:health4 higher4:higher4 internet4:internet4 key4:key4 nursery4:nursery4 paid4:paid4 passed4:passed4 reason4:reason4 romantic4:romantic4 school4:school4 schoolup4:schoolup4 sex4:sex4 studytime4:studytime4 traveltime4:traveltime4]
map[Dalc5:Fedu5:Fedu5 Fjob5:Fjob5 Medu5:Medu5 Mjob5:Mjob5 Pstatus5:Pstatus5 Walc5:Walc5 absences5:absences5 activities5:activities5 address5:address5 age5:age5 failures5:failure5 famrel5:famrel5 famsize5:famsize5 fansup5:fansup5 freetime5:freetime5 gout5:gout5 guardian5:guardian5 health5:health5 higher5:higher5 internet5:internet5 key5:key5 nursery5:nursery5 paid5:paid5 passed5:passed5 reason5:reason5 romantic5:romantic5 school5:school5 schoolup5:schoolup5 sex5:sex5 studytime5:studytime5 traveltime5:traveltime5]
map[Dalc6:Fedu6:Fedu6 Fjob6:Fjob6 Medu6:Medu6 Mjob6:Mjob6 Pstatus6:Pstatus6 Walc6:Walc6 absences6:absences6 activities6:activities6 address6:address6 age6:age6 failures6:failure6 famrel6:famrel6 famsize6:famsize6 fansup6:fansup6 freetime6:freetime6 gout6:gout6 guardian6:guardian6 health6:health6 higher6:higher6 internet6:internet6 key6:key6 nursery6:nursery6 paid6:paid6 passed6:passed6 reason6:reason6 romantic6:romantic6 school6:school6 schoolup6:schoolup6 sex6:sex6 studytime6:studytime6 traveltime6:traveltime6]
map[Dalc7:Fedu7:Fedu7 Fjob7:Fjob7 Medu7:Medu7 Mjob7:Mjob7 Pstatus7:Pstatus7 Walc7:Walc7 absences7:absences7 activities7:activities7 address7:address7 age7:age7 failures7:failure7 famrel7:famrel7 famsize7:famsize7 fansup7:fansup7 freetime7:freetime7 gout7:gout7 guardian7:guardian7 health7:health7 higher7:higher7 internet7:internet7 key7:key7 nursery7:nursery7 paid7:paid7 passed7:passed7 reason7:reason7 romantic7:romantic7 school7:school7 schoolup7:schoolup7 sex7:sex7 studytime7:studytime7 traveltime7:traveltime7]
map[Dalc8:Fedu8:Fedu8 Fjob8:Fjob8 Medu8:Medu8 Mjob8:Mjob8 Pstatus8:Pstatus8 Walc8:Walc8 absences8:absences8 activities8:activities8 address8:address8 age8:age8 failures8:failure8 famrel8:famrel8 famsize8:famsize8 fansup8:fansup8 freetime8:freetime8 gout8:gout8 guardian8:guardian8 health8:health8 higher8:higher8 internet8:internet8 key8:key8 nursery8:nursery8 paid8:paid8 passed8:passed8 reason8:reason8 romantic8:romantic8 school8:school8 schoolup8:schoolup8 sex8:sex8 studytime8:studytime8 traveltime8:traveltime8]
map[Dalc9:Fedu9:Fedu9 Fjob9:Fjob9 Medu9:Medu9 Mjob9:Mjob9 Pstatus9:Pstatus9 Walc9:Walc9 absences9:absences9 activities9:activities9 address9:address9 age9:age9 failures9:failure9 famrel9:famrel9 famsize9:famsize9 fansup9:fansup9 freetime9:freetime9 gout9:gout9 guardian9:guardian9 health9:health9 higher9:higher9 internet9:internet9 key9:key9 nursery9:nursery9 paid9:paid9 passed9:passed9 reason9:reason9 romantic9:romantic9 school9:school9 schoolup9:schoolup9 sex9:sex9 studytime9:studytime9 traveltime9:traveltime9]
map[Dalc10:Fedu10:Fedu10 Fjob10:Fjob10 Medu10:Medu10 Mjob10:Mjob10 Pstatus10:Pstatus10 Walc10:Walc10 absences10:absences10 activities10:activities10 address10:address10 age10:age10 failures10:failure10 famrel10:famrel10 famsize10:famsize10 fansup10:fansup10 freetime10:freetime10 gout10:gout10 guardian10:guardian10 health10:health10 higher10:higher10 internet10:internet10 key10:key10 nursery10:nursery10 paid10:paid10 passed10:passed10 reason10:reason10 romantic10:romantic10 school10:school10 schoolup10:schoolup10 sex10:sex10 studytime10:studytime10 traveltime10:traveltime10]
map[Dalc11:Fedu11:Fedu11 Fjob11:Fjob11 Medu11:Medu11 Mjob11:Mjob11 Pstatus11:Pstatus11 Walc11:Walc11 absences11:absences11 activities11:activities11 address11:address11 age11:age11 failures11:failure11 famrel11:famrel11 famsize11:famsize11 fansup11:fansup11 freetime11:freetime11 gout11:gout11 guardian11:guardian11 health11:health11 higher11:higher11 internet11:internet11 key11:key11 nursery11:nursery11 paid11:paid11 passed11:passed11 reason11:reason11 romantic11:romantic11 school11:school11 schoolup11:schoolup11 sex11:sex11 studytime11:studytime11 traveltime11:traveltime11]
map[Dalc12:Fedu12:Fedu12 Fjob12:Fjob12 Medu12:Medu12 Mjob12:Mjob12 Pstatus12:Pstatus12 Walc12:Walc12 absences12:absences12 activities12:activities12 address12:address12 age12:age12 failures12:failure12 famrel12:famrel12 famsize12:famsize12 fansup12:fansup12 freetime12:freetime12 gout12:gout12 guardian12:guardian12 health12:health12 higher12:higher12 internet12:internet12 key12:key12 nursery12:nursery12 paid12:paid12 passed12:passed12 reason12:reason12 romantic12:romantic12 school12:school12 schoolup12:schoolup12 sex12:sex12 studytime12:studytime12 traveltime12:traveltime12]
map[Dalc13:Fedu13:Fedu13 Fjob13:Fjob13 Medu13:Medu13 Mjob13:Mjob13 Pstatus13:Pstatus13 Walc13:Walc13 absences13:absences13 activities13:activities13 address13:address13 age13:age13 failures13:failure13 famrel13:famrel13 famsize13:famsize13 fansup13:fansup13 freetime13:freetime13 gout13:gout13 guardian13:guardian13 health13:health13 higher13:higher13 internet13:internet13 key13:key13 nursery13:nursery13 paid13:paid13 passed13:passed13 reason13:reason13 romantic13:romantic13 school13:school13 schoolup13:schoolup13 sex13:sex13 studytime13:studytime13 traveltime13:traveltime13]
map[Dalc14:Fedu14:Fedu14 Fjob14:Fjob14 Medu14:Medu14 Mjob14:Mjob14 Pstatus14:Pstatus14 Walc14:Walc14 absences14:absences14 activities14:activities14 address14:address14 age14:age14 failures14:failure14 famrel14:famrel14 famsize14:famsize14 fansup14:fansup14 freetime14:freetime14 gout14:gout14 guardian14:guardian14 health14:health14 higher14:higher14 internet14:internet14 key14:key14 nursery14:nursery14 paid14:paid14 passed14:passed14 reason14:reason14 romantic14:romantic14 school14:school14 schoolup14:schoolup14 sex14:sex14 studytime14:studytime14 traveltime14:traveltime14]
map[Dalc15:Fedu15:Fedu15 Fjob15:Fjob15 Medu15:Medu15 Mjob15:Mjob15 Pstatus15:Pstatus15 Walc15:Walc15 absences15:absences15 activities15:activities15 address15:address15 age15:age15 failures15:failure15 famrel15:famrel15 famsize15:famsize15 fansup15:fansup15 freetime15:freetime15 gout15:gout15 guardian15:guardian15 health15:health15 higher15:higher15 internet15:internet15 key15:key15 nursery15:nursery15 paid15:paid15 passed15:passed15 reason15:reason15 romantic15:romantic15 school15:school15 schoolup15:schoolup15 sex15:sex15 studytime15:studytime15 traveltime15:traveltime15]
map[Dalc16:Fedu16:Fedu16 Fjob16:Fjob16 Medu16:Medu16 Mjob16:Mjob16 Pstatus16:Pstatus16 Walc16:Walc16 absences16:absences16 activities16:activities16 address16:address16 age16:age16 failures16:failure16 famrel16:famrel16 famsize16:famsize16 fansup16:fansup16 freetime16:freetime16 gout16:gout16 guardian16:guardian16 health16:health16 higher16:higher16 internet16:internet16 key16:key16 nursery16:nursery16 paid16:paid16 passed16:passed16 reason16:reason16 romantic16:romantic16 school16:school16 schoolup16:schoolup16 sex16:sex16 studytime16:studytime16 traveltime16:traveltime16]
map[Dalc17:Fedu17:Fedu17 Fjob17:Fjob17 Medu17:Medu17 Mjob17:Mjob17 Pstatus17:Pstatus17 Walc17:Walc17 absences17:absences17 activities17:activities17 address17:address17 age17:age17 failures17:failure17 famrel17:famrel17 famsize17:famsize17 fansup17:fansup17 freetime17:freetime17 gout17:gout17 guardian17:guardian17 health17:health17 higher17:higher17 internet17:internet17 key17:key17 nursery17:nursery17 paid17:paid17 passed17:passed17 reason17:reason17 romantic17:romantic17 school17:school17 schoolup17:schoolup17 sex17:sex17 studytime17:studytime17 traveltime17:traveltime17]
map[Dalc18:Fedu18:Fedu18 Fjob18:Fjob18 Medu18:Medu18 Mjob18:Mjob18 Pstatus18:Pstatus18 Walc18:Walc18 absences18:absences18 activities18:activities18 address18:address18 age18:age18 failures18:failure18 famrel18:famrel18 famsize18:famsize18 fansup18:fansup18 freetime18:freetime18 gout18:gout18 guardian18:guardian18 health18:health18 higher18:higher18 internet18:internet18 key18:key18 nursery18:nursery18 paid18:paid18 passed18:passed18 reason18:reason18 romantic18:romantic18 school18:school18 schoolup18:schoolup18 sex18:sex18 studytime18:studytime18 traveltime18:traveltime18]
map[Dalc19:Fedu19:Fedu19 Fjob19:Fjob19 Medu19:Medu19 Mjob19:Mjob19 Pstatus19:Pstatus19 Walc19:Walc19 absences19:absences19 activities19:activities19 address19:address19 age19:age19 failures19:failure19 famrel19:famrel19 famsize19:famsize19 fansup19:fansup19 freetime19:freetime19 gout19:gout19 guardian19:guardian19 health19:health19 higher19:higher19 internet19:internet19 key19:key19 nursery19:nursery19 paid19:paid19 passed19:passed19 reason19:reason19 romantic19:romantic19 school19:school19 schoolup19:schoolup19 sex19:sex19 studytime19:studytime19 traveltime19:traveltime19]
map[Dalc20:Fedu20:Fedu20 Fjob20:Fjob20 Medu20:Medu20 Mjob20:Mjob20 Pstatus20:Pstatus20 Walc20:Walc20 absences20:absences20 activities20:activities20 address20:address20 age20:age20 failures20:failure20 famrel20:famrel20 famsize20:famsize20 fansup20:fansup20 freetime20:freetime20 gout20:gout20 guardian20:guardian20 health20:health20 higher20:higher20 internet20:internet20 key20:key20 nursery20:nursery20 paid20:paid20 passed20:passed20 reason20:reason20 romantic20:romantic20 school20:school20 schoolup20:schoolup20 sex20:sex20 studytime20:studytime20 traveltime20:traveltime20]
map[Dalc21:Fedu21:Fedu21 Fjob21:Fjob21 Medu21:Medu21 Mjob21:Mjob21 Pstatus21:Pstatus21 Walc21:Walc21 absences21:absences21 activities21:activities21 address21:address21 age21:age21 failures21:failure21 famrel21:famrel21 famsize21:famsize21 fansup21:fansup21 freetime21:freetime21 gout21:gout21 guardian21:guardian21 health21:health21 higher21:higher21 internet21:internet21 key21:key21 nursery21:nursery21 paid21:paid21 passed21:passed21 reason21:reason21 romantic21:romantic21 school21:school21 schoolup21:schoolup21 sex21:sex21 studytime21:studytime21 traveltime21:traveltime21]
map[Dalc22:Fedu22:Fedu22 Fjob22:Fjob22 Medu22:Medu22 Mjob22:Mjob22 Pstatus22:Pstatus22 Walc22:Walc22 absences22:absences22 activities22:activities22 address22:address22 age22:age22 failures22:failure22 famrel22:famrel22 famsize22:famsize22 fansup22:fansup22 freetime22:freetime22 gout22:gout22 guardian22:guardian22 health22:health22 higher22:higher22 internet22:internet22 key22:key22 nursery22:nursery22 paid22:paid22 passed22:passed22 reason22:reason22 romantic22:romantic22 school22:school22 schoolup22:schoolup22 sex22:sex22 studytime22:studytime22 traveltime22:traveltime22]
map[Dalc23:Fedu23:Fedu23 Fjob23:Fjob23 Medu23:Medu23 Mjob23:Mjob23 Pstatus23:Pstatus23 Walc23:Walc23 absences23:absences23 activities23:activities23 address23:address23 age23:age23 failures23:failure23 famrel23:famrel23 famsize23:famsize23 fansup23:fansup23 freetime23:freetime23 gout23:gout23 guardian23:guardian23 health23:health23 higher23:higher23 internet23:internet23 key23:key23 nursery23:nursery23 paid23:paid23 passed23:passed23 reason23:reason23 romantic23:romantic23 school23:school23 schoolup23:schoolup23 sex23:sex23 studytime23:studytime23 traveltime23:traveltime23]
map[Dalc24:Fedu24:Fedu24 Fjob24:Fjob24 Medu24:Medu24 Mjob24:Mjob24 Pstatus24:Pstatus24 Walc24:Walc24 absences24:absences24 activities24:activities24 address24:address24 age24:age24 failures24:failure24 famrel24:famrel24 famsize24:famsize24 fansup24:fansup24 freetime24:freetime24 gout24:gout24 guardian24:guardian24 health24:health24 higher24:higher24 internet24:internet24 key24:key24 nursery24:nursery24 paid24:paid24 passed24:passed24 reason24:reason24 romantic24:romantic24 school24:school24 schoolup24:schoolup24 sex24:sex24 studytime24:studytime24 traveltime24:traveltime24]
map[Dalc25:Fedu25:Fedu25 Fjob25:Fjob25 Medu25:Medu25 Mjob25:Mjob25 Pstatus25:Pstatus25 Walc25:Walc25 absences25:absences25 activities25:activities25 address25:address25 age25:age25 failures25:failure25 famrel25:famrel25 famsize25:famsize25 fansup25:fansup25 freetime25:freetime25 gout25:gout25 guardian25:guardian25 health25:health25 higher25:higher25 internet25:internet25 key25:key25 nursery25:nursery25 paid25:paid25 passed25:passed25 reason25:reason25 romantic25:romantic25 school25:school25 schoolup25:schoolup25 sex25:sex25 studytime25:studytime25 traveltime25:traveltime25]
map[Dalc26:Fedu26:Fedu26 Fjob26:Fjob26 Medu26:Medu26 Mjob26:Mjob26 Pstatus26:Pstatus26 Walc26:Walc26 absences26:absences26 activities26:activities26 address26:address26 age26:age26 failures26:failure26 famrel26:famrel26 famsize26:famsize26 fansup26:fansup26 freetime26:freetime26 gout26:gout26 guardian26:guardian26 health26:health26 higher26:higher26 internet26:internet26 key26:key26 nursery26:nursery26 paid26:paid26 passed26:passed26 reason26:reason26 romantic26:romantic26 school26:school26 schoolup26:schoolup26 sex26:sex26 studytime26:studytime26 traveltime26:traveltime26]
map[Dalc27:Fedu27:Fedu27 Fjob27:Fjob27 Medu27:Medu27 Mjob27:Mjob27 Pstatus27:Pstatus27 Walc27:Walc27 absences27:absences27 activities27:activities27 address27:address27 age27:age27 failures27:failure27 famrel27:famrel27 famsize27:famsize27 fansup27:fansup27 freetime27:freetime27 gout27:gout27 guardian27:guardian27 health27:health27 higher27:higher27 internet27:internet27 key27:key27 nursery27:nursery27 paid27:paid27 passed27:passed27 reason27:reason27 romantic27:romantic27 school27:school27 schoolup27:schoolup27 sex27:sex27 studytime27:studytime27 traveltime27:traveltime27]
map[Dalc28:Fedu28:Fedu28 Fjob28:Fjob28 Medu28:Medu28 Mjob28:Mjob28 Pstatus28:Pstatus28 Walc28:Walc28 absences28:absences28 activities28:activities28 address28:address28 age28:age28 failures28:failure28 famrel28:famrel28 famsize28:famsize28 fansup28:fansup28 freetime28:freetime28 gout28:gout28 guardian28:guardian28 health28:health28 higher28:higher28 internet28:internet28 key28:key28 nursery28:nursery28 paid28:paid28 passed28:passed28 reason28:reason28 romantic28:romantic28 school28:school28 schoolup28:schoolup28 sex28:sex28 studytime28:studytime28 traveltime28:traveltime28]
map[Dalc29:Fedu29:Fedu29 Fjob29:Fjob29 Medu29:Medu29 Mjob29:Mjob29 Pstatus29:Pstatus29 Walc29:Walc29 absences29:absences29 activities29:activities29 address29:address29 age29:age29 failures29:failure29 famrel29:famrel29 famsize29:famsize29 fansup29:fansup29 freetime29:freetime29 gout29:gout29 guardian29:guardian29 health29:health29 higher29:higher29 internet29:internet29 key29:key29 nursery29:nursery29 paid29:paid29 passed29:passed29 reason29:reason29 romantic29:romantic29 school29:school29 schoolup29:schoolup29 sex29:sex29 studytime29:studytime29 traveltime29:traveltime29]
map[Dalc30:Fedu30:Fedu30 Fjob30:Fjob30 Medu30:Medu30 Mjob30:Mjob30 Pstatus30:Pstatus30 Walc30:Walc30 absences30:absences30 activities30:activities30 address30:address30 age30:age30 failures30:failure30 famrel30:famrel30 famsize30:famsize30 fansup30:fansup30 freetime30:freetime30 gout30:gout30 guardian30:guardian30 health30:health30 higher30:higher30 internet30:internet30 key30:key30 nursery30:nursery30 paid30:paid30 passed30:passed30 reason30:reason30 romantic30:romantic30 school30:school30 schoolup30:schoolup30 sex30:sex30 studytime30:studytime30 traveltime30:traveltime30]
map[Dalc31:Fedu31:Fedu31 Fjob31:Fjob31 Medu31:Medu31 Mjob31:Mjob31 Pstatus31:Pstatus31 Walc31:Walc31 absences31:absences31 activities31:activities31 address31:address31 age31:age31 failures31:failure31 famrel31:famrel31 famsize31:famsize31 fansup31:fansup31 freetime31:freetime31 gout31:gout31 guardian31:guardian31 health31:health31 higher31:higher31 internet31:internet31 key31:key31 nursery31:nursery31 paid31:paid31 passed31:passed31 reason31:reason31 romantic31:romantic31 school31:school31 schoolup31:schoolup31 sex31:sex31 studytime31:studytime31 traveltime31:traveltime31]
map[Dalc32:Fedu32:Fedu32 Fjob32:Fjob32 Medu32:Medu32 Mjob32:Mjob32 Pstatus32:Pstatus32 Walc32:Walc32 absences32:absences32 activities32:activities32 address32:address32 age32:age32 failures32:failure32 famrel32:famrel32 famsize32:famsize32 fansup32:fansup32 freetime32:freetime32 gout32:gout32 guardian32:guardian32 health32:health32 higher32:higher32 internet32:internet32 key32:key32 nursery32:nursery32 paid32:paid32 passed32:passed32 reason32:reason32 romantic32:romantic32 school32:school32 schoolup32:schoolup32 sex32:sex32 studytime32:studytime32 traveltime32:traveltime32]
map[Dalc33:Fedu33:Fedu33 Fjob33:Fjob33 Medu33:Medu33 Mjob33:Mjob33 Pstatus33:Pstatus33 Walc33:Walc33 absences33:absences33 activities33:activities33 address33:address33 age33:age33 failures33:failure33 famrel33:famrel33 famsize33:famsize33 fansup33:fansup33 freetime33:freetime33 gout33:gout33 guardian33:guardian33 health33:health33 higher33:higher33 internet33:internet33 key33:key33 nursery33:nursery33 paid33:paid33 passed33:passed33 reason33:reason33 romantic33:romantic33 school33:school33 schoolup33:schoolup33 sex33:sex33 studytime33:studytime33 traveltime33:traveltime33]
map[Dalc34:Fedu34:Fedu34 Fjob34:Fjob34 Medu34:Medu34 Mjob34:Mjob34 Pstatus34:Pstatus34 Walc34:Walc34 absences34:absences34 activities34:activities34 address34:address34 age34:age34 failures34:failure34 famrel34:famrel34 famsize34:famsize34 fansup34:fansup34 freetime34:freetime34 gout34:gout34 guardian34:guardian34 health34:health34 higher34:higher34 internet34:internet34 key34:key34 nursery34:nursery34 paid34:paid34 passed34:passed34 reason34:reason34 romantic34:romantic34 school34:school34 schoolup34:schoolup34 sex34:sex34 studytime34:studytime34 traveltime34:traveltime34]
map[Dalc35:Fedu35:Fedu35 Fjob35:Fjob35 Medu35:Medu35 Mjob35:Mjob35 Pstatus35:Pstatus35 Walc35:Walc35 absences35:absences35 activities35:activities35 address35:address35 age35:age35 failures35:failure35 famrel35:famrel35 famsize35:famsize35 fansup35:fansup35 freetime35:freetime35 gout35:gout35 guardian35:guardian35 health35:health35 higher35:higher35 internet35:internet35 key35:key35 nursery35:nursery35 paid35:paid35 passed35:passed35 reason35:reason35 romantic35:romantic35 school35:school35 schoolup35:schoolup35 sex35:sex35 studytime35:studytime35 traveltime35:traveltime35]
map[Dalc36:Fedu36:Fedu36 Fjob36:Fjob36 Medu36:Medu36 Mjob36:Mjob36 Pstatus36:Pstatus36 Walc36:Walc36 absences36:absences36 activities36:activities36 address36:address36 age36:age36 failures36:failure36 famrel36:famrel36 famsize36:famsize36 fansup36:fansup36 freetime36:freetime36 gout36:gout36 guardian36:guardian36 health36:health36 higher36:higher36 internet36:internet36 key36:key36 nursery36:nursery36 paid36:paid36 passed36:passed36 reason36:reason36 romantic36:romantic36 school36:school36 schoolup36:schoolup36 sex36:sex36 studytime36:studytime36 traveltime36:traveltime36]
map[Dalc37:Fedu37:Fedu37 Fjob37:Fjob37 Medu37:Medu37 Mjob37:Mjob37 Pstatus37:Pstatus37 Walc37:Walc37 absences37:absences37 activities37:activities37 address37:address37 age37:age37 failures37:failure37 famrel37:famrel37 famsize37:famsize37 fansup37:fansup37 freetime37:freetime37 gout37:gout37 guardian37:guardian37 health37:health37 higher37:higher37 internet37:internet37 key37:key37 nursery37:nursery37 paid37:paid37 passed37:passed37 reason37:reason37 romantic37:romantic37 school37:school37 schoolup37:schoolup37 sex37:sex37 studytime37:studytime37 traveltime37:traveltime37]
map[Dalc38:Fedu38:Fedu38 Fjob38:Fjob38 Medu38:Medu38 Mjob38:Mjob38 Pstatus38:Pstatus38 Walc38:Walc38 absences38:absences38 activities38:activities38 address38:address38 age38:age38 failures38:failure38 famrel38:famrel38 famsize38:famsize38 fansup38:fansup38 freetime38:freetime38 gout38:gout38 guardian38:guardian38 health38:health38 higher38:higher38 internet38:internet38 key38:key38 nursery38:nursery38 paid38:paid38 passed38:passed38 reason38:reason38 romantic38:romantic38 school38:school38 schoolup38:schoolup38 sex38:sex38 studytime38:studytime38 traveltime38:traveltime38]
map[Dalc39:Fedu39:Fedu39 Fjob39:Fjob39 Medu39:Medu39 Mjob39:Mjob39 Pstatus39:Pstatus39 Walc39:Walc39 absences39:absences39 activities39:activities39 address39:address39 age39:age39 failures39:failure39 famrel39:famrel39 famsize39:famsize39 fansup39:fansup39 freetime39:freetime39 gout39:gout39 guardian39:guardian39 health39:health39 higher39:higher39 internet39:internet39 key39:key39 nursery39:nursery39 paid39:paid39 passed39:passed39 reason39:reason39 romantic39:romantic39 school39:school39 schoolup39:schoolup39 sex39:sex39 studytime39:studytime39 traveltime39:traveltime39]
map[Dalc40:Fedu40:Fedu40 Fjob40:Fjob40 Medu40:Medu40 Mjob40:Mjob40 Pstatus40:Pstatus40 Walc40:Walc40 absences40:absences40 activities40:activities40 address40:address40 age40:age40 failures40:failure40 famrel40:famrel40 famsize40:famsize40 fansup40:fansup40 freetime40:freetime40 gout40:gout40 guardian40:guardian40 health40:health40 higher40:higher40 internet40:internet40 key40:key40 nursery40:nursery40 paid40:paid40 passed40:passed40 reason40:reason40 romantic40:romantic40 school40:school40 schoolup40:schoolup40 sex40:sex40 studytime40:studytime40 traveltime40:traveltime40]
```

*Figure 18 – screenshot of output code that shows all data of csv file loaded into linklist*

(3b) When a row is added or removed from the CSV file, the memory database will adapt to the changes.

### i. Python Implementation

After you edit student-data.csv, the file is read again with ReadAndWriteCSV.read\_csv(path), which returns the latest rows and headers. These rows are passed to SingleLinkedListDatabase.adapt\_changes(new\_rows), which keeps the in-memory list in sync.

The method works in three passes:

#### Index new data by key

- It builds new\_map = {row[self.id]: row for row in new\_rows} so each new row can be looked up instantly by its "key".

#### Collect existing keys from the list

- A small inner function collect(n) walks the linked list (from self.head) and gathers all current keys, skipping the header "node" if present. This gives an ordered snapshot of what's already in memory.

#### Reconcile differences

- **Removals:** for any key found in memory but **not** in new\_map, call remove\_node(k) to unlink that node.
- **Updates:** for keys present in **both** places, call update\_node(new\_map[k]) to replace the node's record with the new values.
- **Insertions:** for keys that exist in new\_map but **not** in memory, call update\_node(row) which falls back to add\_data(row) when the node can't be found.

Because search\_node, remove\_node, and update\_node are written to traverse and modify the chain safely, this one routine leaves the linked list mirroring the latest CSV exactly.

```

Open ▾  memory_database_using_ll.py
memory_database_using_ll.py
# searching node one by one using key here that is id
def search_node(self, key):
    # It do recursively search from start to last node and next is none then return
    if n is None:
        return None
    if n.list.get(self.id, "") == key:
        return n
    # keep searching next node recursively
    return search(n.next)
return search(self.head)

# remove node by using key
def remove_node(self, key):
    def remove(prev, current):
        # If node not found then return false
        if current is None:
            return False
        # If we found the key then either remove head or unlink current and connect to next node
        if current.list.get(self.id, "") == key:
            if prev is None:
                self.head = current.next
            else:
                prev.next = current.next
            self.size -= 1
            return True
        # keep checking next node recursively
        return remove(current, current.next)
    return remove(None, self.head)

# update node by row
def update_node(self, row):
    # find row by id then add it
    key = row.get(self.id, "")
    node = self.search_node(key)
    # if node not found then add it else update value
    if node is None:
        self.add_data(row)
    else:
        node.list = row

# that change the link |list as updated
def adapt_changes(self, new_rows):
    # build dictionary of new rows from csv by key and check row is new, removed, or updated.
    new_map = {x.get(self.id, ""): x for x in new_rows}

    # collect current keys in linked list
    existing_keys = []
    def collect():
        # stop reach the end of the list
        if n is None:
            return
        # get the key for these node
        keyn.list.get(self.id, "")
        if key != "key":
            existing_keys.append(key)
        # move to the next node recursive
        collect(n.next)
    collect(self.head)

    # start from the head
    collect(self.head)

    # remove rows not in new
    for k in existing_keys:
        if k not in new_map:
            self.remove_node(k)

    # update rows that still exist
    for k in existing_keys:
        if k in new_map:
            self.update_node(new_map[k])

    # add new rows
    for k, row in new_map.items():
        if k not in existing_keys:
            self.update_node(row)

# main function
def main():
    # read csv file rows is data and header is all column name
    rows, headers = ReadAndWriteCSV.read_csv("../student-data.csv")
    # use lambda function for data so that add first header and then other data
    rows = [[h for h in header]] + rows
    # create an empty linked list database
    db = SingleLinkedListDatabase(id="key")
    # recursive load data
    db.load_data(rows, 0)
    # print data
    db.print_data()
    print(f"Total node: {db.size}")
    print("=> Edit 'student-data.csv'")
    input("=>Press Enter when change is done. ")
    new_rows, header = ReadAndWriteCSV.read_csv("./student-data.csv")
    db.adapt_changes(new_rows)
    db.print_data()
    print(f"Total node : {db.size}")

if __name__ == "__main__":
    main()

```

*Figure 19 – Screenshot of python code until task 3b*

After running the Python program, the linked list is first built with all rows from the CSV file. The terminal shows every student record being printed out, followed by a message asking the user to edit student-data.csv. At this point, the file can be modified—such as adding a new row, removing an existing one, or updating values. Once the user presses Enter, the program re-reads the CSV and calls the `adapt_changes()` method. This function compares the updated rows with the current nodes and performs three actions: it removes nodes for rows that no longer exist, updates values in nodes whose records have changed, and inserts nodes for any new rows. The terminal output then displays the refreshed linked list, showing the newly synchronized records. The total node count is updated at the end, confirming that the in-memory list now exactly matches the CSV file.

Figure 20 – output of the code that shows that linklist updated as per csv file for python

ii Go Implementation

In Go, the flow is the same. After re-reading the file via `ReadWriteCSV_read_csv(path)`, the program invokes `SingleLinkedListDatabase adapt_changes(new_rows)`.

1. **Build a lookup of fresh rows** [SEP] newMap = map[string]map[string]string{} is filled so each "key" points to its new row.
  2. **Collect existing keys** [SEP] A recursive helper collect(n \*Node) starts at db head and pushes each node's key onto existingKeys, skipping the header node's key if needed.
  3. **Apply removals, updates, insertions**
    - **Remove:** if an existing key isn't in newMap, call db remove\_node(k) to relink pointers and shrink db size.
    - **Update:** if it exists in both, call db update\_node(row) to replace that node's list map.
    - **Insert:** if a key is only in newMap, db update\_node(row) will add it via db add\_data(row) when no node is found.

The end state is identical to Python: the in-memory list exactly reflects the file on disk, with all pointer changes handled cleanly.

```

// search node by key
func (db *singleLinkedListDatabase) search_node(key string) *Node {
    var search func(*Node) *Node
    search = func(n *Node) *Node {
        if n == nil {
            return nil
        }
        // found the key
        if n.list[db.id] == key {
            return n
        }
        // keep searching next node
        return search(n.next)
    }
    return search(db.head)
}

// remove node by key
func (db *singleLinkedListDatabase) remove_node(key string) bool {
    var remove func(prev, current *Node) bool
    remove = func(prev, current *Node) bool {
        // base case: reached end
        if current == nil {
            return false
        }
        // found the key
        if current.list[db.id] == key {
            if prev == nil { // removing head
                db.head = current.next
            } else { // within current
                prev.next = current.next
            }
            db.size--
            return true
        }
        // keep checking next node
        return remove(current, current.next)
    }
    return remove(nil, db.head)
}

// update node with row data
func (db *singleLinkedListDatabase) update_node(row map[string]string) {
    key := row[db.id]
    node := db.search_node(key)
    if node == nil {
        // not found - append new node
        db.add_data(row)
    } else {
        // found - update values
        node.list = row
    }
}

// adapt database when rows added/removed
func (db *singleLinkedListDatabase) adapt_changes(new_rows []map[string]string) {
    // build dictionary of new rows by key
    newMap := make(map[string]map[string]string)
    for _, x := range new_rows {
        newMap[x[db.id]] = x
    }

    // collect current keys in linked list
    existingKeys := []string{}
    var collect func(*Node)
    collect = func(n *Node) {
        if n == nil {
            return
        }
        existingKeys = append(existingKeys, n.list[db.id])
        collect(n.next)
    }
    collect(db.head)

    // 1) remove rows not in new
    for _, k := range existingKeys {
        if _, ok := newMap[k]; !ok {
            db.remove_node(k)
        }
    }

    // 2) update rows that still exist
    for _, k := range existingKeys {
        if row, ok := newMap[k]; ok {
            db.update_node(row)
        }
    }

    // 3) add brand new rows
    for k, row := range newMap {
        found := false
        for _, ex := range existingKeys {
            if ex == k {
                found = true
                break
            }
        }
        if !found {
            db.update_node(row)
        }
    }
}

// main function
func main() {
    // read csv file rows is ...
    rw, err := os.ReadFile("student-data.csv")
    rows, header, err := rw.ReadAllCSV("../student-data.csv")
    if err != nil {
        fmt.Println("Error:", err)
        return
    }

    // add header row first
    headerRow := map[string]string{}
    for _, h := range header {
        headerRow[h] = h
    }

    // create an empty linked list database
    db := singleLinkedListDatabase{id: "key"}
    // recursive load data
    db.load_data(rows, 0)
    // print data
    db.print_data()
    fmt.Printf("Total nodes: %d\n", db.size)

    fmt.Println("Edit 'student-data.csv' now!")
    fmt.Println("Press Enter when you are ready to continue")
    fmt.Scanln()

    new_rows, _, err := rw.ReadAllCSV("../student-data.csv")
    if err != nil {
        fmt.Println("Error:", err)
        return
    }

    db.adapt_changes(new_rows)
    db.print_data()
    fmt.Printf("Total nodes: %d\n", db.size)
}

```

Figure 21 – Screenshot of go code that shows implemented function until 3b

The Go program produces similar behaviour. When it starts, `addData()` inserts every record into the linked list, and `printData()` displays the full set of rows from the CSV. The terminal then pauses and prompts the user to edit `student-data.csv`. Any modifications—such as adding, deleting, or editing student information—are applied directly to the file. After pressing Enter, the program reloads the CSV and runs the `adapt_changes()` method. Just like in Python, this method checks for removed rows, updates changed ones, and adds new entries as needed. The output shows the adjusted list printed node by node, reflecting the edits made in the CSV file. At the bottom, the updated total node count is printed, verifying that the Go linked list remains in sync with the external file.

```

raaj@ubuntu25: ~/Desktop/Algorithm/homework2/go
activities:address:U age:17 failures:0 famrel:2 famsize:LE3 famsup: no freetime:4 gout:5 guardian:other health:2 higher:yes internet:yes key:392 nursery:no paid:passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:2]
map[Dalc:3 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:3 absences:3 activities:no address:R age:21 failures:3 famrel:5 famsize:GT3 famsup: no freetime:5 gout:3 guardian:other health:3 higher:yes internet:no key:393 nursery:no paid:passed:reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:1]
map[Dalc:3 Fedu:2 Fjob:other Medu:3 Mjob:services Pstatus:T Walc:4 absences:0 activities:no address:R age:18 failures:0 famrel:4 famsize:LE3 famsup: no freetime:4 gout:0 guardian:mother health:5 higher:yes internet:yes key:394 nursery:yes paid:no passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:3]
map[Dalc:3 Fedu:3 Fjob:at_home Medu:1 Mjob:other Pstatus:T Walc:3 absences:5 activities:address:U age:19 failures:0 famrel:3 famsize:LE3 famsup: no freetime:2 gout:3 guardian:father health:5 higher:yes internet:yes key:395 nursery:yes paid:no passed:reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:1]
Total node: 395

>>> Please edit 'student-data.csv' now (add/remove a row).
>>> Press Enter when you are ready to continue

```

student-data.csv

```

memory_database_using_ll.go student-data.csv
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/go
activities:address:U age:17 failures:0 famrel:2 famsize:LE3 famsup: no freetime:4 gout:5 guardian:other health:2 higher:yes internet:yes key:392 nursery:no paid:passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:2]
map[Dalc:3 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:3 absences:3 activities:no address:R age:21 failures:3 famrel:5 famsize:GT3 famsup: no freetime:5 gout:3 guardian:other health:3 higher:yes internet:no key:393 nursery:no paid:passed:reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:1]
map[Dalc:3 Fedu:2 Fjob:other Medu:3 Mjob:services Pstatus:T Walc:4 absences:0 activities:no address:R age:18 failures:0 famrel:4 famsize:LE3 famsup: no freetime:4 gout:0 guardian:mother health:5 higher:yes internet:yes key:394 nursery:yes paid:no passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:3]
map[Dalc:3 Fedu:3 Fjob:at_home Medu:1 Mjob:other Pstatus:T Walc:3 absences:5 activities:address:U age:19 failures:0 famrel:3 famsize:LE3 famsup: no freetime:2 gout:3 guardian:father health:5 higher:yes internet:yes key:395 nursery:yes paid:no passed:reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:1]
Total node: 395

```

Figure 22 – Output of the code that shows that linklist updated as per csv file

### (3c) Export the memory database into a csv file through recursive function

#### i. Python Implementation

Exporting is handled by SingleLinkedListDatabase.export\_to\_csv\_file(path, headers)

- Prepare headers: it drops the internal "key" column from headers to form export\_data, since that field is only for indexing in memoryss4.
- Write header row: it calls ReadAndWriteCSV.write\_csv(path, export\_data) to create/overwrite the file with the headers
- Recursive write of data rows: an inner function write\_node(n) walks the list. For each node it opens the file in append mode, writes the row values in export\_data order, then recursively calls itself on n.next until it reaches None. To avoid duplicating the header “node,” it starts from self.head.next if a header node was inserted.

The result is a clean CSV (updated\_student-data.csv) that matches what’s currently in memory.

```

# export csv file from loaded linklist
def export_to_csv_file(self, path, data):
    # drop the 'key' column that is for reference to access node
    export_data = [h for h in data if h != "key"]

    # write header first
    ReadAndWriteCSV.write_csv(path, export_data)

    # then write rows by recursively until end
    def write_node(n):
        if n is None:
            return
        # write file in append mode line by line
        with open(path, "a", newline="") as f:
            w = csv.writer(f)
            w.writerow([n.list.get(h, "") for h in export_data])
        # move to the next node in the linked list
        write_node(n.next)
    # skip the header node to avoid repetition
    start = self.head.next if self.head else None
    write_node(start)

```

```

def write_csv(path, headers):
    # open file in write mode and write line by line
    with open(path, "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(headers)
# main function:
def main():
    # read csv file rows is data and header is all column name
    rows,header= ReadAndWriteCSV.read_csv("../student-data.csv")
    # use lambda function for data so that add first header and then other data
    rows = [h: h for h in header] + rows
    # create an empty linked list database
    db = SingleLinkedListDatabase(id="key")
    # recursively load data
    db.load_data(rows, 0)
    db.print_data() # print data
    print(f"Total node: {db.size}")
    print("=>Edit 'student-data.csv'")
    input("=>Press Enter when change is done ... ")
    new_rows, header = ReadAndWriteCSV.read_csv("../student-data.csv")
    db.adapt_changes(new_rows)
    db.print_data()
    print(f"Total node : {db.size}")
    db.export_to_csv_file("updated_student-data.csv", header)
    print("Exported update csv file named \"updated_student-data.csv\"")
if __name__ == "__main__":
    main()

```

Figure 23 – Screenshot of python code until task 3c

When the Python program finishes adapting the linked list to match the current CSV file, the next step is to write the updated records back into a new CSV. The terminal output shows that the `export_to_csv_file()` method is called, which traverses the linked list node by node and writes each record in order. During this step, the "key" column used for indexing inside memory is skipped, and only the actual student details are saved. At the end of execution, a confirmation message appears in the terminal . This message verifies that the in-memory database was successfully written to a new file. Looking at the file explorer, we can see `updated_student-data.csv` created alongside the Python program. Opening the file confirms that all 390 rows were exported correctly, matching the records stored in the linked list.

```

raaj@ubuntu25: ~/Desktop/Algorithm/homework2/python
[school]: 'MS', 'sex': 'M', 'age': '21', 'address': 'R', 'famsize': 'GT3', 'Pstatus': 'T', 'Medu': '1', 'Fedu': '1', 'Mjob': 'other', 'Fjob': 'other', 'reason': 'course', 'guardian': 'other', 'traveltime': '1', 'studytme': '1', 'failures': '3', 'schoolsups': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'no', 'higher': 'yes', 'internet': 'no', 'romantic': 'no', 'famrel': '5', 'freetime': '5', 'gout': '3', 'Dalc': '3', 'Walc': '3', 'health': '3', 'absences': '3', 'passed': 'no', 'key': '393'}
[school]: 'MS', 'sex': 'M', 'age': '18', 'address': 'R', 'famsize': 'LE3', 'Pstatus': 'T', 'Medu': '3', 'Fedu': '2', 'Mjob': 'services', 'Fjob': 'other', 'reas on': 'course', 'guardian': 'mother', 'traveltime': '3', 'studytme': '1', 'failures': '0', 'schoolsups': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'no', 'higher': 'yes', 'internet': 'yes', 'romantic': 'no', 'famrel': '4', 'freetime': '4', 'gout': '1', 'Dalc': '3', 'Walc': '4', 'health': '5', 'absences': '0', 'passed': 'yes', 'key': '394'}
[school]: 'MS', 'sex': 'M', 'age': '19', 'address': 'U', 'famsize': 'LE3', 'Pst atus': 'T', 'Medu': '1', 'Fedu': '1', 'Mjob': 'other', 'Fjob': 'at_home', 'reas on': 'course', 'guardian': 'father', 'traveltime': '1', 'studytme': '1', 'failures': '0', 'schoolsups': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'internet': 'yes', 'romantic': 'no', 'famrel': '4', 'freetime': '2', 'gout': '3', 'Dalc': '3', 'Walc': '3', 'health': '5', 'absences': '5', 'passed': 'no', 'key': '395'}
Total node: 396
=> Edit 'student-data.csv'
=>Press Enter when change is done ...
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/python
[school]: 'MS', 'sex': 'F', 'age': '18', 'address': 'R', 'famsize': 'GT3', 'Pst atus': 'T', 'Medu': '4', 'Fedu': '4', 'Mjob': 'teacher', 'Fjob': 'at_home', 'reas on': 'reputation', 'guardian': 'mother', 'traveltime': '3', 'studytme': '1', 'failures': '0', 'schoolsups': 'no', 'famsup': 'yes', 'paid': 'yes', 'activities': 'yes', 'nursery': 'yes', 'higher': 'yes', 'internet': 'yes', 'romantic': 'yes', 'famrel': '4', 'freetime': '4', 'gout': '3', 'Dalc': '2', 'Walc': '2', 'health': '5', 'absences': '7', 'passed': 'no', 'key': '387'}
[school]: 'MS', 'sex': 'F', 'age': '19', 'address': 'R', 'famsize': 'GT3', 'Pst atus': 'T', 'Medu': '2', 'Fedu': '3', 'Mjob': 'services', 'Fjob': 'other', 'reas on': 'course', 'guardian': 'mother', 'traveltime': '1', 'studytme': '3', 'failures': '1', 'schoolsups': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'yes', 'nursery': 'no', 'higher': 'yes', 'internet': 'yes', 'romantic': 'no', 'famrel': '5', 'freetime': '4', 'gout': '2', 'Dalc': '1', 'Walc': '2', 'health': '5', 'absences': '0', 'passed': 'no', 'key': '388'}
[school]: 'MS', 'sex': 'F', 'age': '18', 'address': 'U', 'famsize': 'LE3', 'Pst atus': 'T', 'Medu': '3', 'Fedu': '1', 'Mjob': 'teacher', 'Fjob': 'services', 'reas on': 'course', 'guardian': 'mother', 'traveltime': '1', 'studytme': '2', 'failures': '0', 'schoolsups': 'no', 'famsup': 'yes', 'paid': 'yes', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'internet': 'yes', 'romantic': 'no', 'famrel': '4', 'freetime': '3', 'gout': '4', 'Dalc': '1', 'Walc': '1', 'health': '1', 'absences': '0', 'passed': 'no', 'key': '389'}
Total node: 390
Exported update csv file named "updated_student-data.csv"
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/python

```

Figure 24 – Output that shows line removed and new file created and that is updated

## ii. Go Implementation

Go mirrors the same structure with `SingleLinkedListDatabase.export_to_csv_file(path string, headers []string)`

- Trim headers: it filters out "key" and writes the header once by calling `ReadAndWriteCSV_write_csv(path, exportHeaders)ss4.`
- Recursive row writer: a nested function `write_node(n *Node)` opens the file in append mode, writes one CSV record for the node in header order, flushes, closes, and then recurses to `n.next` until nil. As in Python, it skips the header "node" by starting from `db.head.next` when presentss4.

When the recursion finishes, you get `updated_student-data.csv` that you can quickly verify with a preview command. The exported file proves that the list was traversed correctly and that each node became exactly one CSV row.

```

// export csv file from loaded linklist
func (db *singleLinkedListDatabase) export_to_csv_file(path string, data []string) error {
    // drop the "key" column that is for reference to access node
    exportData := []string{}
    for _, h := range data {
        if h != "key" {
            exportData = append(exportData, h)
        }
    }

    // write header first
    if err := ReadAndWriteCSV_write_csv(path, exportData); err != nil {
        return err
    }

    // then write rows by recursively until end
    var write_node func(*Node) error
    write_node = func(n *Node) error {
        if n == nil {
            return nil
        }

        // write file in append mode line by line
        f, err := os.OpenFile(path, os.O_APPEND|os.O_WRONLY, 0644)
        if err != nil {
            return err
        }

        w := csv.NewWriter(f)
        row := make([]string, len(exportData))
        for i, h := range exportData {
            row[i] = n.list[h]
        }

        if err := w.Write(row); err != nil {
            _ = f.Close()
            return err
        }

        w.Flush()
        _ = f.Close()
        // move to the next node in the linked list
        return write_node(n.next)
    }

    // skip the header node to avoid repetition
    start := (*Node)(nil)
    if db.head != nil {
        start = db.head.next
    }
    return write_node(start)
}

// read and write for csv file for that make class (here: standalone functions)

```

### ● memory\_database\_using\_ll.go

```

// write_csv writes a single header row to path (overwrites file)
func ReadAndWriteCSV_write_csv(path string, headers []string) error {
    // open file to write mode and write line by line
    f, err := os.OpenFile(path, os.O_WRONLY, 0644)
    if err != nil {
        return err
    }

    defer f.Close()

    w := csv.NewWriter(f)
    if err := w.Write(headers); err != nil {
        return err
    }

    w.Flush()
    return w.Error()
}

// main function
func main() {
    // read csv file rows ls data and header is all column name
    rows, header, err := ReadAndWriteCSV_read_csv("./student-data.csv")
    if err != nil {
        fat.Println("Error", err)
    }

    // use Lander Function for data so that add First header and then other data
    header[0] = make(string)
    for _, h := range header {
        header[0] += h
    }

    rows = append(header, rows...)
    db := SingleLinkedListDatabase{key: "key"}
    // recursive load data
    db.load(db.list, rows)
    // print data
    db.print_data()
    fat.Println("Total node: ", db.size)

    // prompt to edit CSV, then adapt changes
    fat.Println("=> Edit 'student-data.csv'")
    fat.Println("=> Press Enter when change is done")
    fat.Scanln()

    newRows, header2, err := ReadAndWriteCSV_read_csv("./student-data.csv")
    if err != nil {
        fat.Println("Error", err)
    }

    db.add(header2)
    db.replace(newRows)
    fat.Println("=> Headers already known; list header node stays as-is")
    db.adapt_changes(newRows)
    db.print_data()
    fat.Println("Total node: ", db.size)

    // export this key column and skip header node
    if err := db.export_to_csv_file("updated_student-data.csv", header); err != nil {
        fat.Println("Export error", err)
    }

    fat.Println("=> Please edit 'student-data.csv' now (add/remove a row).")
    fat.Println("=> Press Enter when you are ready to continue")
}

```

Figure 25 – Screenshot of go code until task 3c

The Go program follows the same logic. After reloading and adapting changes to the linked list, the `export_to_csv_file()` method is executed. This method recursively walks through the list, writing each node's data into a new CSV file. As in Python, the "key" field is ignored during export, leaving only the student information in the final file. The terminal output clearly shows the process completing and prints a confirmation message that the updated file has been saved. In the working directory, a file named `updated_student-data.csv` appears, proving that the Go implementation also produced a fresh CSV export. Checking its contents shows that all rows were written out in sequence, ensuring that the linked list traversal was successful.

```

raaj@ubuntu25: ~/Desktop/Algorithm/homework2/go
activities,no address:U age:17 failures:0 famrel:2 famsize:LE3 famsup:no freetime:4 goout:5 guardian:mother health:2 higher:yes internet:yes key:392 nursery:no paid:no passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:2]
map[Dalc:3 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:3 absences:3 activities,no address:R age:21 failures:3 famrel:5 famsize:GT3 famsup:no freetime:5 goout:3 guardian:other health:3 higher:yes internet:no key:393 nursery:no paid:no passed:no reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:1]
map[Dalc:3 Fedu:2 Fjob:other Medu:3 Mjob:services Pstatus:T Walc:4 absences:0 activities,no address:R age:18 failures:0 famrel:4 famsize:LE3 famsup:no freetime:4 goout:1 guardian:mother health:5 higher:yes internet:yes key:394 nursery:no paid:yes passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:3]
map[Dalc:3 Fedu:1 Fjob:at_home Medu:1 Mjob:other Pstatus:T Walc:3 absences:5 activities,no address:U age:19 failures:0 famrel:3 famsize:LE3 famsup:no freetime:2 goout:3 guardian:father health:5 higher:yes internet:yes key:395 nursery:yes paid:no passed:no reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 traveltime:1]
Total node: 395

>>> Please edit 'student-data.csv' now (add/remove a row).
>>> Press Enter when you are ready to continue

```

Figure 26 – Screenshot of output where go file is executed and program run

Figure 27 – output that shows 4line added and new file created and that is updated

# **HYPERLINK TO ONLINE GDB**

The complete source code for both the Python and Go implementations has been uploaded to OnlineGDB, which allows to run the code directly in a browser. A free account may be required for smooth execution.

Python Code: <https://onlinegdb.com/MIkbA73Rh>

Go Code: <https://onlinegdb.com/i6buUufAg5>

## DISCUSSION

Working on this homework gave me a clearer understanding of how data structures operate when implemented manually, without relying on built-in database systems. By building a small memory database with a singly linked list, I was able to see how records can be represented as nodes and how pointers connect them together. This reinforced the importance of linked lists in computer science as a foundation for more advanced storage and retrieval techniques.

One of the main lessons from this assignment was the difference between working in Python and Go. In Python, the implementation felt smoother because the language is highly flexible and does not require explicit type declarations. Adding new data, traversing the list, and printing records were straightforward to implement and test. Python's built-in libraries also made file handling simple, which helped when reading data from the CSV file and writing it back after modifications.

Go, however, required me to think more carefully about data structure definitions and pointer management. Creating structs for nodes, and the linked list gave me a deeper look into how memory is handled explicitly. The strict type of system meant fewer errors at runtime, but it also made me more cautious while writing functions. Although this made development slightly more challenging compared to Python, it gave me valuable practice in handling low-level details and understanding how memory safety is enforced in Go.

Another key takeaway was how traversal works in a linked list. Whether in Python or Go, the process of walking through each node one by one shows why operations like search or update in a linked list have a time complexity of  $O(n)$ . This experience helped me appreciate the trade-offs between simplicity of implementation and efficiency of data access.

Overall, this homework highlighted the importance of learning through implementation. By coding the same logic in two different languages, I not only reinforced my understanding of linked lists but also developed confidence in adapting to different programming environments. It was a useful reminder that while syntax and tools vary between languages, the underlying concepts and theory.

## LIMITATIONS AND FUTUREWORK

This homework showed how a simple memory database can be built with a singly linked list, but it also has some clear limits. Right now, the system only supports three main actions: loading rows from a CSV, adjusting when the file changes, and exporting the list back into a CSV. While this works fine for small datasets, the design is not very efficient for larger files. Every search, update, or deletion requires walking through the list one node at a time, which can slow things down as the data grows. The program assumes the CSV is always clean and properly formatted. If the file is corrupted or missing values, errors may occur. Another limitation is that all data is stored in memory, which means the system is not suited for very large or persistent datasets.

Looking ahead, there are several ways this homework could be improved. One idea is to move from a singly linked list to a more advanced structure, such as a doubly linked list or even a tree, to make searching and updating faster. Adding indexing or hashing would also make lookups much more efficient. Since Go naturally supports concurrency, future versions could take advantage of goroutines to allow multiple operations to run in parallel. Another improvement would be to add stronger error handling, so the program can deal with unexpected input gracefully. Finally, expanding export options beyond CSV—such as JSON or SQL—would make the database more flexible and easier to connect with other applications.

## **CONCLUSION**

This assignment gave me valuable hands-on practice in creating a memory database using a singly linked list. By implementing the same concept in both Python and Go, I was able to strengthen my understanding of how data structures work and how different programming languages influence the way we write code.

Python provided a simple and flexible environment to develop the database. Its easy syntax and built-in libraries made it straightforward to load CSV data, insert new records, and display the list. This allowed me to focus more on the logic of the linked list rather than dealing with strict language rules.

Go, in contrast, required a more detailed approach with explicit struct definitions and pointer handling. While this added complexity, it also gave me a deeper appreciation of memory management and type safety. The experience of working with Go showed me how lower-level control can lead to more predictable and secure programs, even if the development process is less forgiving than in Python.

Overall, this homework connected theory with practice. It showed me how a fundamental data structure like a linked list can be applied to real tasks such as storing and managing student records from a CSV file. More importantly, it taught me how the same algorithmic idea can be implemented differently depending on the programming language. This homework reinforced my understanding of data structures while improving my ability to adapt algorithms across different programming languages.

## **ACKNOWLEDGE**

I would like to express my sincere gratitude to my professor for assigning this homework and guiding me through the process. This homework not only gave me the opportunity to strengthen my understanding of linked lists and memory management but also allowed me to gain practical experience in working with both Python and Go. The support and encouragement provided during the course helped me stay motivated and approach the task with confidence. The homework helped me connect theoretical concepts with real-world applications, and I truly appreciate the effort my professor put into designing homework that challenge us to think critically and learn by doing. Overall, this assignment has been a valuable experience, and I deeply thank my professor for providing me with the chance to explore data structures in such a meaningful and practical way.

## REFERENCES

- Kaggle. (n.d.). Student Data CSV Dataset by Kelly Gakii. Retrieved September 19, 2025, from <https://www.kaggle.com/datasets/kellygakii/student-data-csv>
- Python Software Foundation. (2025). CSV File Reading and Writing — Python 3 Documentation from <https://docs.python.org/3/library/csv.html>
- Go Documentation. (2025). CSV Reader – Package encoding/csv. The Go Authors. Retrieved September 19, 2025, from <https://pkg.go.dev/encoding/csv>
- Ubuntu. (2025). Download Ubuntu Desktop 25.04. Canonical Ltd. Retrieved September 5, 2025, from <https://ubuntu.com/download/desktop>
- Oracle. (2025). VirtualBox Downloads. Oracle Corporation. Retrieved September 19, 2025, from <https://www.virtualbox.org/wiki/Downloads>
- Ubuntu Documentation. (2025). Install Python on Ubuntu. Canonical Ltd. Retrieved September 19, 2025, from <https://ubuntu.com/server/docs/installing-python>
- Ubuntu Documentation. (2025). Install Go on Ubuntu. Canonical Ltd. Retrieved September 19, 2025, from <https://ubuntu.com/tutorials/install-go>
- Programiz. (n.d.). Linked List Data Structure. Retrieved September 10, 2025, from <https://www.programiz.com/dsa/linked-list>
- OnlineGDB. (2025) *Online IDE for C, C++, Python, Go, and Java*. Retrieved September 19, 2025, from <https://www.onlinegdb.com>
- OnlineGDB. (2025). *Python Linked List Memory Database Code*. Uploaded on September 19, 2025, from <https://www.onlinegdb.com/MIkbA73Rh>
- OnlineGDB. (2025). *Go Linked List Memory Database Code* . Uploaded on September 19, 2025, from <https://www.onlinegdb.com/i6buUufAg5>

# APPENDIX

## 1. Python code

```
import csv

# create nodes for single linked list

class Node:

    def __init__(self,list):
        # store one row from and next node is point to none
        self.list = list
        self.next= None

#create database using single linklist

class SingleLinkedListDatabase:

    def __init__(self,id):
        # start with an empty single linklist here head stores row ,size for length
        self.head = None
        self.size = 0

        # this column is treated as unique id for search/update/remove
        self.id = id

    # add a new node at the end for that we take input as current node and data

    def add_data(self, data):
        # make new node and store data
        new_node = Node(data)

        # if the list is completely empty then add head
        if self.head is None:
            self.head = new_node

        # add last node's next new node and new's next is none
        else:
            n = self.head
            while n.next is not None:
                n = n.next
            n.next = new_node
            self.size += 1

    # load data row by row into linklist and defaultt index value is 0

    def load_data(self,rows,i=0):
```

```

# stop if index is greater or equal to number of rows then return
if i >= len(rows):
    return

# add row to linklist and increase index to one for next
self.add_data(rows[i])

self.load_data(rows, i + 1)

# print database loaded in linked list

def print_data(self):
    # start from head node and find until next is none
    n=self.head

    while n is not None:
        print(n.list)

        n = n.next

    # searching node one by one using key here that is id

def search_node(self, key):
    def search(n):
        # it do recursively search form start to last node and next is none then return
        if n is None:
            return None

        if n.list.get(self.id, "") == key:
            return n

        # keep searching next node recusively
        return search(n.next)

    return search(self.head)

# remove node by using key

def remove_node(self, key):
    def remove(prev, current):
        # if node not found then return false
        if current is None:
            return False

        # if we the found the key then either remove head or unlink current and connect to next node
        if current.list.get(self.id, "") == key:
            if prev is None:
                self.head = current.next
            else:
                prev.next = current.next

            self.size -= 1

```

```

        return True

    # keep checking next node recursively
    return remove(current, current.next)

    return remove(None, self.head)

# update node by row

def update_node(self, row):
    # find row by id then add it

    key = row.get(self.id, "")
    node = self.search_node(key)

    # if node not found then add it else update value

    if node is None:
        self.add_data(row)
    else:
        node.list = row

    # that change the linklist as updated

def adapt_changes(self, new_rows):
    # build dictionary of new rows from csv by key and check row is new, removed, or updated.

    new_map = {x.get(self.id, ""): x for x in new_rows}

    # collect current keys in linked list

    existing_keys = []

    def collect(n):
        # stop reach the end of the list

        if n is None:
            return

        # get the key for these node
        key=n.list.get(self.id, "")

        if key != "key":
            existing_keys.append(key)

        # move to the next node recursive
        collect(n.next)

    # start from the head
    collect(self.head)

    # remove rows not in new
    for k in existing_keys:
        if k not in new_map:
            self.remove_node(k)

    # update rows that still exist

```

```

for k in existing_keys:
    if k in new_map:
        self.update_node(new_map[k])
    # add new rows
    for k, row in new_map.items():
        if k not in existing_keys:
            self.update_node(row)
# export csv file from loaded linklist
def export_to_csv_file(self, path, data):
    # drop the 'key' column that is for reference to access node
    export_data = [h for h in data if h != "key"]
    # write header first
    ReadAndWriteCSV.write_csv(path, export_data)
    # then write rows by recursively until end
    def write_node(n):
        if n is None:
            return
        # write file in append mode line by line
        with open(path, "a", newline="") as f:
            w = csv.writer(f)
            w.writerow([n.list.get(h, "") for h in export_data])
        # move to the next node in the linked list
        write_node(n.next)
        # skip the header node to avoid repetition
        start = self.head.next if self.head else None
        write_node(start)
# read and write for csv file for that make class
class ReadAndWriteCSV:
    def read_csv(path):
        # rows for data and headers for column name
        rows = []
        headers = None
        # open the CSV file and read line as dictionary and first line for header
        with open(path, newline="") as f:
            reader = csv.DictReader(f)
            headers = reader.fieldnames
            # add a indexing key for identify row

```

```

i = 1

for r in reader:

    r = dict(r)

    r["key"] = str(i)

    rows.append(r)

    i += 1

# check that key is there or not

if "key" not in headers:

    headers = ["key"] + headers

# return the list that contain data

return rows, headers

def write_csv(path, headers):

    # open file in write mode and write line by line

    with open(path, "w", newline="") as f:

        w = csv.writer(f)

        w.writerow(headers)

# main function

def main():

    # read csv file rows is data and header is all column name

    rows,header= ReadAndWriteCSV.read_csv("./student-data.csv")

    # use lambda function for data so that add first header and then other data

    rows = [ {h: h for h in header} ] + rows

    # create an empty linked list database

    db = SingleLinkedListDatabase(id="key")

    # recursive load data

    db.load_data(rows, 0)

    # print data

    db.print_data()

    print(f"Total node: {db.size}")

    print("=> Edit 'student-data.csv'")

    input("=>Press Enter when change is done ... ")

    new_rows, header = ReadAndWriteCSV.read_csv("./student-data.csv")

    db.adapt_changes(new_rows)

    db.print_data()

    print(f"Total node : {db.size}")

    db.export_to_csv_file("updated_student-data.csv", header)

```

```

print("Exported update csv file named \"updated_student-data.csv\"")

if __name__ == "__main__":
    main()

```

## 2. Go code

```

package main

import (
    "encoding/csv"
    "fmt"
    "os"
)

// create nodes for single linked list

type Node struct {
    // store one row from and next node is point to nil
    list map[string]string
    next *Node}
}

// create database using single linklist

type SingleLinkedListDatabase struct {
    // start with an empty single linklist here head stores row ,size for length
    head *Node
    size int
    // this column is treated as unique id for search/update/remove
    id string}

// add a new node at the end for that we take input as current node and data

func (db *SingleLinkedListDatabase) add_data(data map[string]string) {
    // make new node and store data
    newNode := &Node{list: data, next: nil}
    // if the list is completely empty then add head
    if db.head == nil {
        db.head = newNode} else {
        // add last node's next new node and new's next is nil
        n := db.head
        for n.next != nil {
            n = n.next}
        n.next = newNode}
    db.size++}

// load data row by row into linklist and defaultt index value is 0

func (db *SingleLinkedListDatabase) load_data(rows []map[string]string, i int) {

```

```

// stop if index is greater or equal to number of rows then return
if i >= len(rows) {return}

// add row to linklist and increase index to one for next
db.add_data(rows[i])

db.load_data(rows, i+1)}

// print database loaded in linked list

func (db *SingleLinkedListDatabase) print_data() {

    // start from head node and find until next is nil

    n := db.head

    for n != nil {

        fmt.Println(n.list)

        n = n.next}}
}

// searching node one by one using key here that is id

func (db *SingleLinkedListDatabase) search_node(key string) *Node {

    var search func(*Node) *Node

    search = func(n *Node) *Node {

        // it do recursively search form start to last node and next is nil then return

        if n == nil {return nil}

        if n.list[db.id] == key {return n}

        // keep searching next node recursively

        return search(n.next)}
    }

    return search(db.head)
}

// remove node by using key

func (db *SingleLinkedListDatabase) remove_node(key string) bool {

    var remove func(prev, current *Node) bool

    remove = func(prev, current *Node) bool {

        // if node not found then return false

        if current == nil {return false}

        // if we the found the key then either remove head or unlink current and connect to next node

        if current.list[db.id] == key {

            if prev == nil {

                db.head = current.next

            } else {

                prev.next = current.next

            }

            db.size--}}
}

```

```

        return true
    }

    // keep checking next node recursively
    return remove(current, current.next)
}

return remove(nil, db.head)
}

// update node by row

func (db *SingleLinkedListDatabase) update_node(row map[string]string) {
    // find row by id then add it
    key := row[db.id]
    node := db.search_node(key)
    // if node not found then add it else update value
    if node == nil {
        db.add_data(row)
    } else {
        node.list = row
    }
}

// that change the linklist as updated

func (db *SingleLinkedListDatabase) adapt_changes(new_rows []map[string]string) {
    // build dictionary of new rows from csv by key and check row is new, removed, or updated.
    newMap := make(map[string]map[string]string)
    for _, x := range new_rows {
        newMap[x[db.id]] = x
    }

    // collect current keys in linked list
    existingKeys := []string{}
    var collect func(*Node)
    collect = func(n *Node) {
        // stop reach the end of the list
        if n == nil {
            return
        }
        // get the key for these node
        key := n.list[db.id]

```

```

        if key != "key" {

            existingKeys = append(existingKeys, key)

        }

        // move to the next node recursive

        collect(n.next)

    }

    // start from the head

    collect(db.head)

    // remove rows not in new

    for _, k := range existingKeys {

        if _, ok := newMap[k]; !ok {

            db.remove_node(k)

        }

    }

    // update rows that still exist

    for _, k := range existingKeys {

        if row, ok := newMap[k]; ok {

            db.update_node(row)

        }

    }

    // add new rows

    for k, row := range newMap {

        found := false

        for _, e := range existingKeys {

            if e == k {

                found = true

                break

            }

        }

        if !found {

            db.update_node(row)

        }

    }

}

// export csv file from loaded linklist

func (db *SingleLinkedListDatabase) export_to_csv_file(path string, data []string) error {

```

```

// drop the 'key' column that is for reference to access node

exportData := []string{}

for _, h := range data {
    if h != "key" {
        exportData = append(exportData, h)
    }
}

// write header first

if err := ReadAndWriteCSV_write_csv(path, exportData); err != nil {
    return err
}

// then write rows by recursively until end

var write_node func(*Node) error

write_node = func(n *Node) error {
    if n == nil {
        return nil
    }

    // write file in append mode line by line

    f, err := os.OpenFile(path, os.O_APPEND|os.O_WRONLY, 0644)
    if err != nil {
        return err
    }

    w := csv.NewWriter(f)

    row := make([]string, len(exportData))

    for i, h := range exportData {
        row[i] = n.list[h]
    }

    if err := w.Write(row); err != nil {
        _ = f.Close()
        return err
    }

    w.Flush()
    _ = f.Close()

    // move to the next node in the linked list

    return write_node(n.next)
}

// skip the header node to avoid repetition

```

```

start := (*Node)(nil)

if db.head != nil {
    start = db.head.next
}

return write_node(start)
}

// read and write for csv file for that make class read_csv opens CSV and returns (rows, headers)

func ReadAndWriteCSV_read_csv(path string) ([]map[string]string, []string, error) {

    // rows for data and headers for column name

    rows := []map[string]string{}

    var headers []string

    // open the CSV file and read line as dictionary and first line for header

    f, err := os.Open(path)

    if err != nil {
        return nil, nil, err
    }

    defer f.Close()

    r := csv.NewReader(f)

    all, err := r.ReadAll()

    if err != nil {
        return nil, nil, err
    }

    if len(all) == 0 {
        return rows, headers, nil
    }

    headers = all[0]

    // add a indexing key for identify row

    for i, rec := range all[1:] {
        row := map[string]string{"key": fmt.Sprintf("%d", i+1)}

        for j, h := range headers {
            if j < len(rec) {
                row[h] = rec[j]
            } else {
                row[h] = ""
            }
        }

        rows = append(rows, row)
    }
}

```

```

    }

// check that key is there or not

found := false

for _, h := range headers {
    if h == "key" {
        found = true
        break
    }
}

if !found {
    headers = append([]string{"key"}, headers...)
}

// return the list that contain data

return rows, headers, nil
}

// write_csv writes a single header row to path (overwrites file)

func ReadAndWriteCSV_write_csv(path string, headers []string) error {
    // open file in write mode and write line by line

    f, err := os.Create(path)

    if err != nil {
        return err
    }

    defer f.Close()

    w := csv.NewWriter(f)

    if err := w.Write(headers); err != nil {
        return err
    }

    w.Flush()

    return w.Error()
}

// main function

func main() {
    // read csv file rows is data and header is all column name

    rows, header, err := ReadAndWriteCSV_read_csv("./student-data.csv")

    if err != nil {
        fmt.Println("Error:", err)
        return
    }

    // use lambda function for data so that add first header and then other data
}

```

```

headerRow := map[string]string{}

for _, h := range header {
    headerRow[h] = h
}

rows = append([]map[string]string{headerRow}, rows...)

// create an empty linked list database

db := SingleLinkedListDatabase{id: "key"}

// recursive load data

db.load_data(rows, 0)

// print data

db.print_data()

fmt.Printf("Total node: %d\n", db.size)

// prompt to edit CSV, then adapt changes

fmt.Println("=> Edit 'student-data.csv'")

fmt.Print("=> Press Enter when change is done")

fmt.Scanln()

newRows, header2, err := ReadAndWriteCSV_read_csv("../student-data.csv")

if err != nil {

    fmt.Println("Error:", err)

    return
}

// headers already known; list header node stays as-is

_ = header2

db.adapt_changes(newRows)

db.print_data()

fmt.Printf("Total node : %d\n", db.size)

// export skip 'key' column and skip header node

if err := db.export_to_csv_file("updated_student-data.csv", header); err != nil {

    fmt.Println("Export error:", err)

    return
}

fmt.Println("Exported update csv file named \"updated_student-data.csv\"")

}

```