Homework 2

CS51510-001, Fall 2025

Purdue University Northwest

09/19/2025

Name	**Email**
Raaj Atulkumar Patel	**Pate2682@pnw.edu**

Table of Contents

Table of Figures

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.

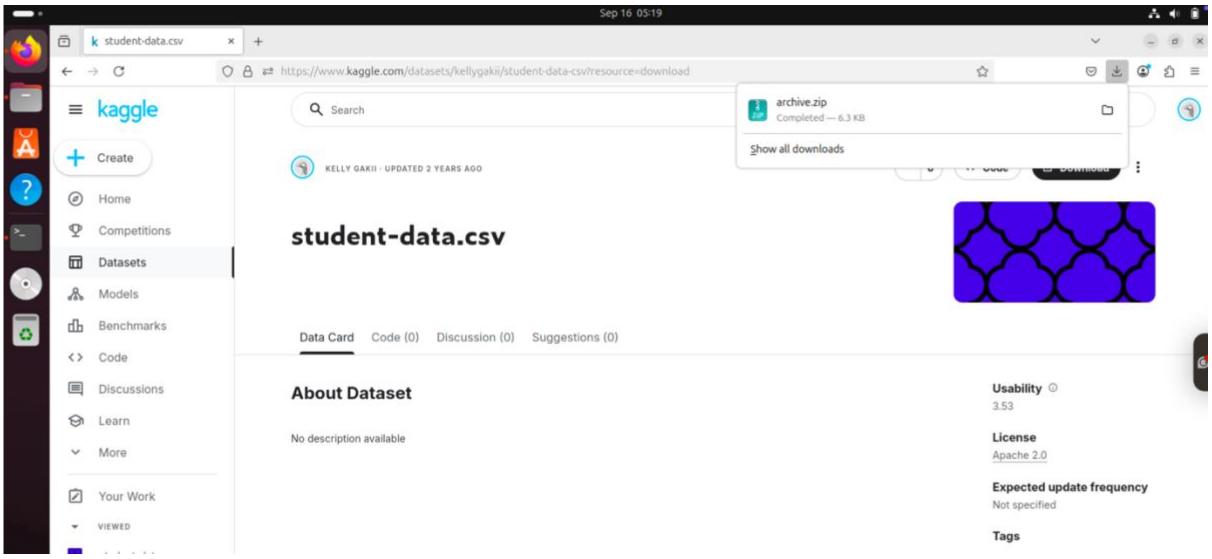


Figure 1 – Download student-data.csv file successfully

Step 2: Unzip and verify the data

- Navigated to the folder where the file was saved.

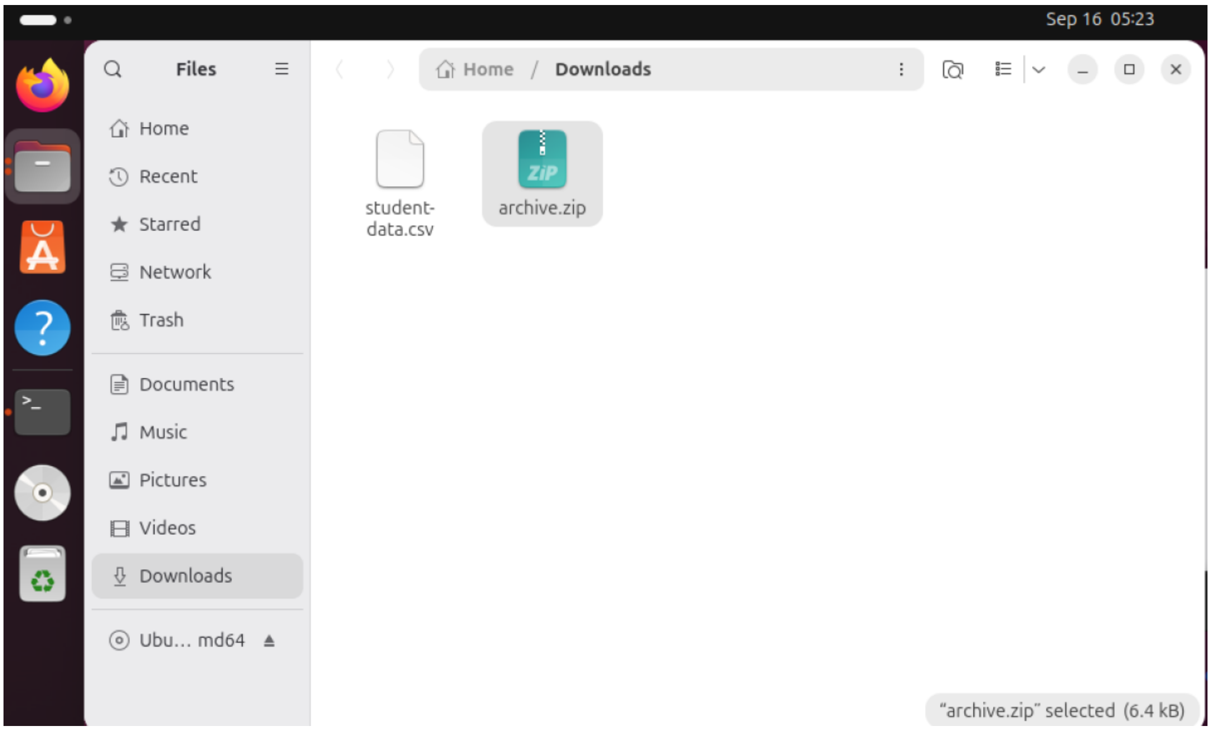
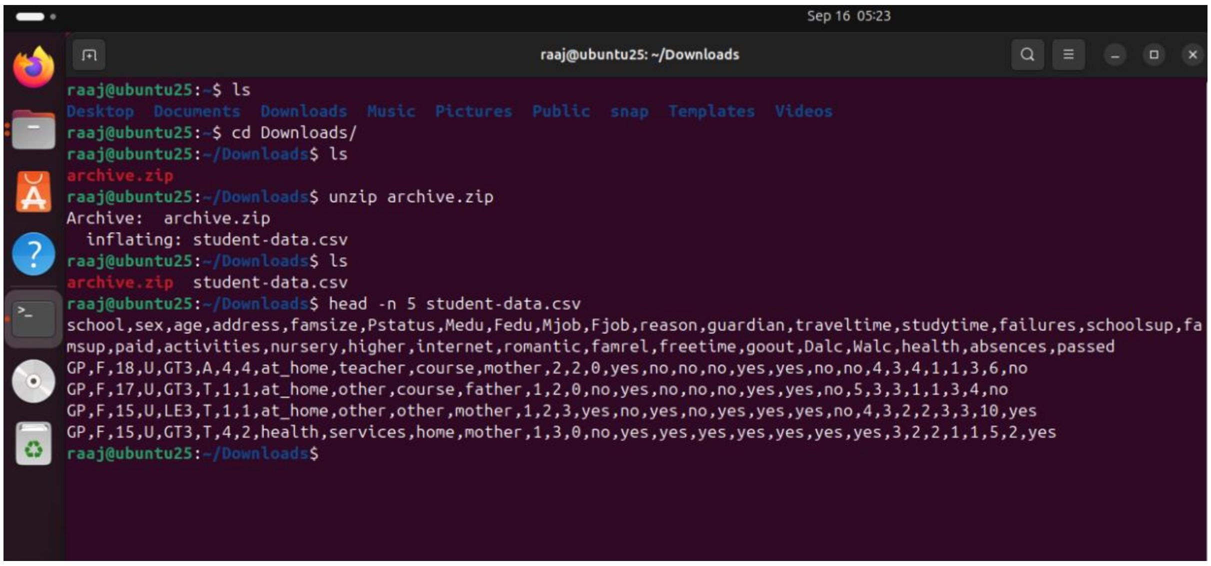


Figure 2 – Download and extra file and get data

- 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.



```
raaj@ubuntu25:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
raaj@ubuntu25:~$ cd Downloads/
raaj@ubuntu25:~/Downloads$ ls
archive.zip
raaj@ubuntu25:~/Downloads$ unzip archive.zip
Archive:  archive.zip
  inflating: student-data.csv
raaj@ubuntu25:~/Downloads$ ls
archive.zip  student-data.csv
raaj@ubuntu25:~/Downloads$ head -n 5 student-data.csv
school,sex,age,address,famsize,Pstatus,Medu,Fedu,Mjob,Fjob,reason,guardian,traveltime,studytime,failures,schoolsup,fa
msup,paid,activities,nursery,higher,internet,romantic,famrel,freetime,goout,Dalc,Walc,health,absences,passed
GP,F,18,U,GT3,A,4,4,at_home,teacher,course,mother,2,2,0,yes,no,no,no,yes,yes,no,no,4,3,4,1,1,3,6,no
GP,F,17,U,GT3,T,1,1,at_home,other,course,father,1,2,0,no,yes,no,no,no,yes,yes,no,5,3,3,1,1,3,4,no
GP,F,15,U,LE3,T,1,1,at_home,other,other,mother,1,2,3,yes,no,yes,no,yes,yes,yes,no,4,3,2,2,3,3,10,yes
GP,F,15,U,GT3,T,4,2,health,services,home,mother,1,3,0,no,yes,yes,yes,yes,yes,yes,yes,3,2,2,1,1,5,2,yes
raaj@ubuntu25:~/Downloads$
```

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

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

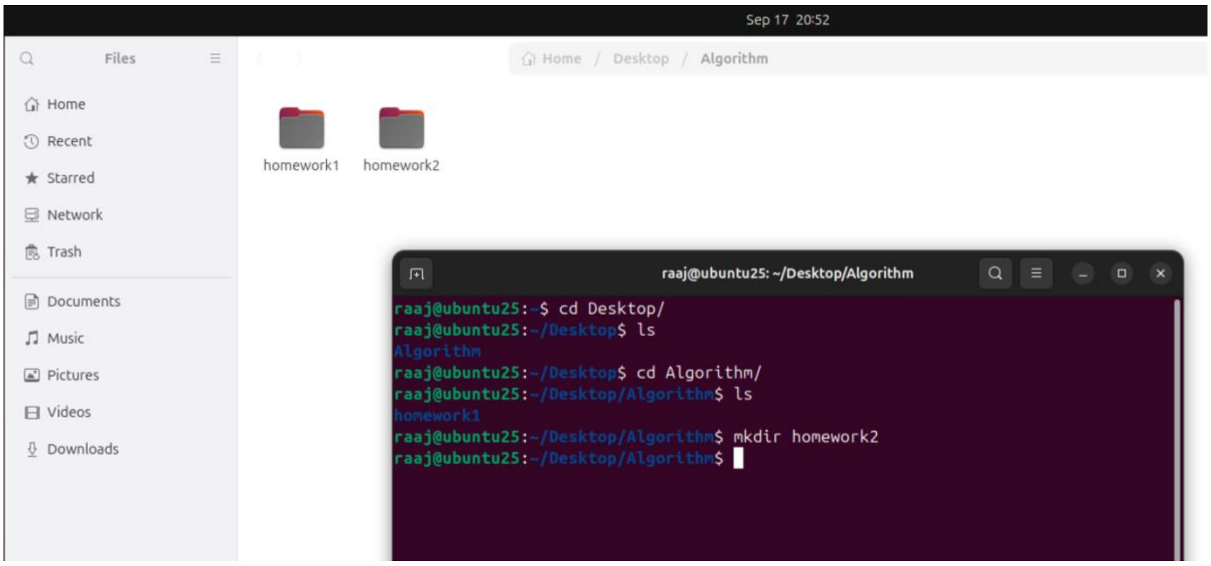
The goal of this task was to design a simple in-memory database from scratch using the concept of a singly linked list. A singly linked list is a basic data structure where each element, called a node, contains two parts: the actual data and a pointer to the next node. The list starts at a special node 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.

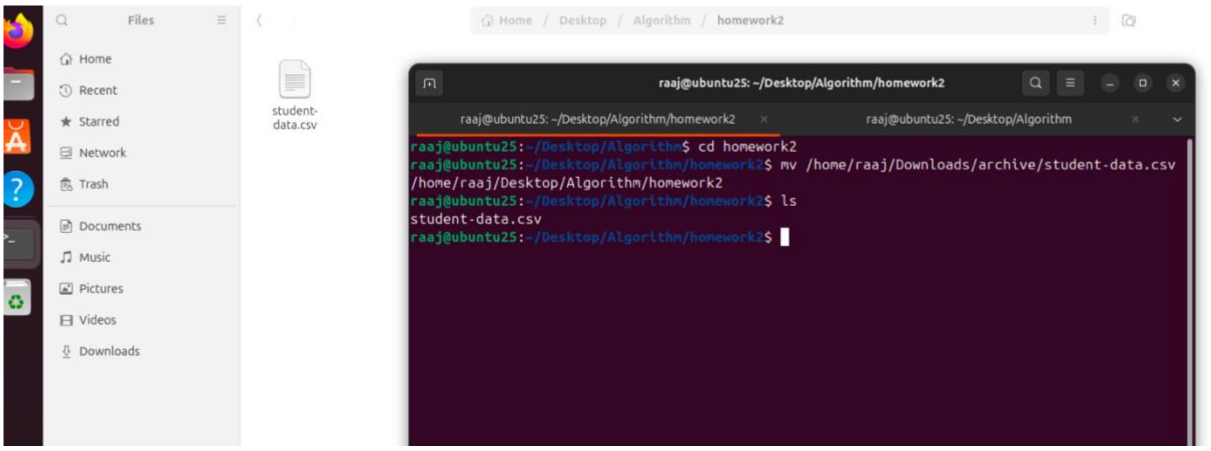
- Create a folder name “homework 2” at algorithm.



```
raaj@ubuntu25:~$ cd Desktop/
raaj@ubuntu25:~/Desktop$ ls
Algorithm
raaj@ubuntu25:~/Desktop$ cd Algorithm/
raaj@ubuntu25:~/Desktop/Algorithm$ ls
homework1
raaj@ubuntu25:~/Desktop/Algorithm$ mkdir homework2
raaj@ubuntu25:~/Desktop/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.

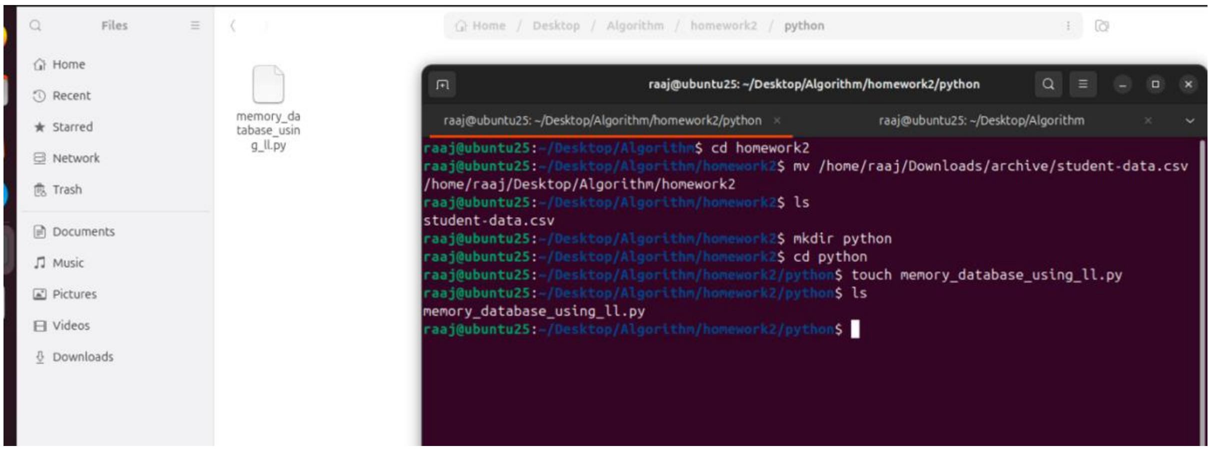


```
raaj@ubuntu25:~/Desktop/Algorithm$ cd homework2
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ mv /home/raaj/Downloads/archive/student-data.csv
/home/raaj/Desktop/Algorithm/homework2
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ ls
student-data.csv
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ 
```

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”



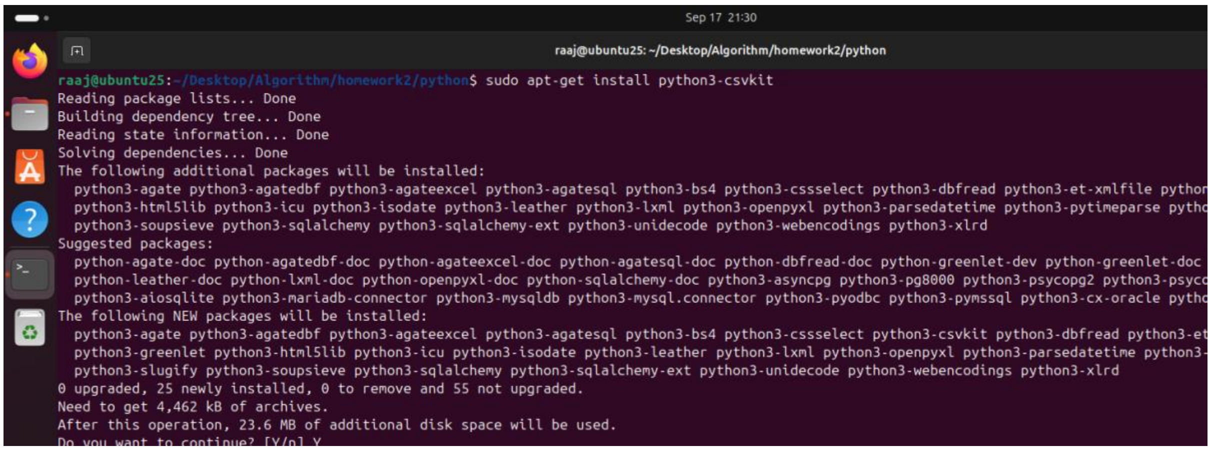
```
raaj@ubuntu25:~/Desktop/Algorithm$ cd homework2
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ mv /home/raaj/Downloads/archive/student-data.csv
/home/raaj/Desktop/Algorithm/homework2
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ ls
student-data.csv
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ mkdir python
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ cd python
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ touch memory_database_using_ll.py
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ ls
memory_database_using_ll.py
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$
```

Files
Home
Recent
Starred
Network
Trash
Documents
Music
Pictures
Videos
Downloads
Home / Desktop / Algorithm / homework2 / python
memory_database_using_ll.py
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/python
raaj@ubuntu25: ~/Desktop/Algorithm

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 pythor
  python3-html5lib python3-icu python3-isodate python3-leather python3-lxml python3-openpyxl python3-parsedatetime python3-pytimeparse pytho
  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-asyncpg python3-pg8000 python3-psycopg2 python3-psycc
  python3-aiosqlite python3-mariadb-connector python3-mysqldb python3-mysql.connector python3-pyodbc python3-pymssql python3-cx-oracle pytho
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
  python3-greenlet python3-html5lib python3-icu python3-isodate python3-leather python3-lxml python3-openpyxl python3-parsedatetime python3-
  python3-slugify 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.

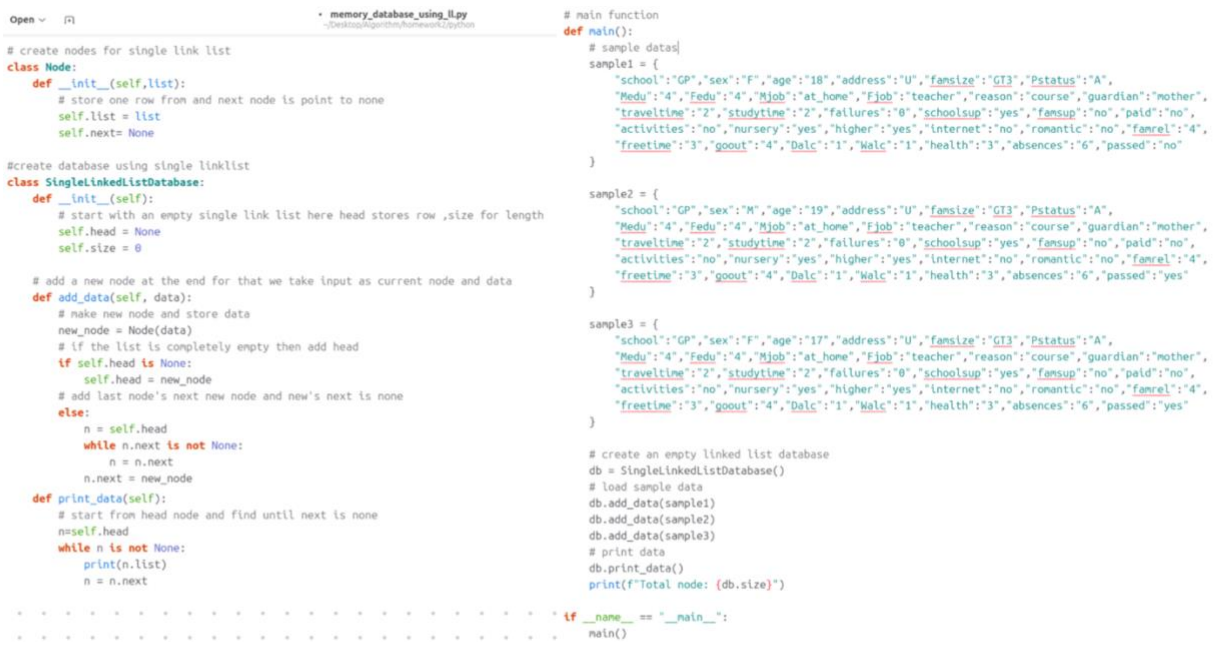
2. SingleLinkedListDatabase Class

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.



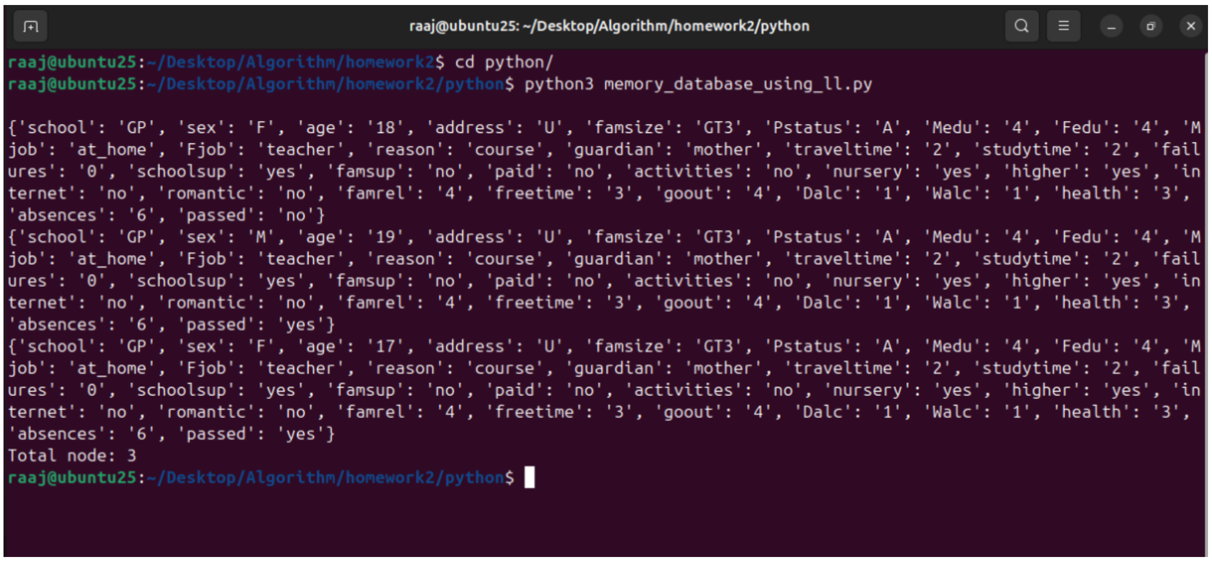
Open ⌄ memory_database_using_ll.py ~/Desktop/Algorithm/homework2/python

```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
```

```python
# main function
def main():
    # sample datas
    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","goout":"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","goout":"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","goout":"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.



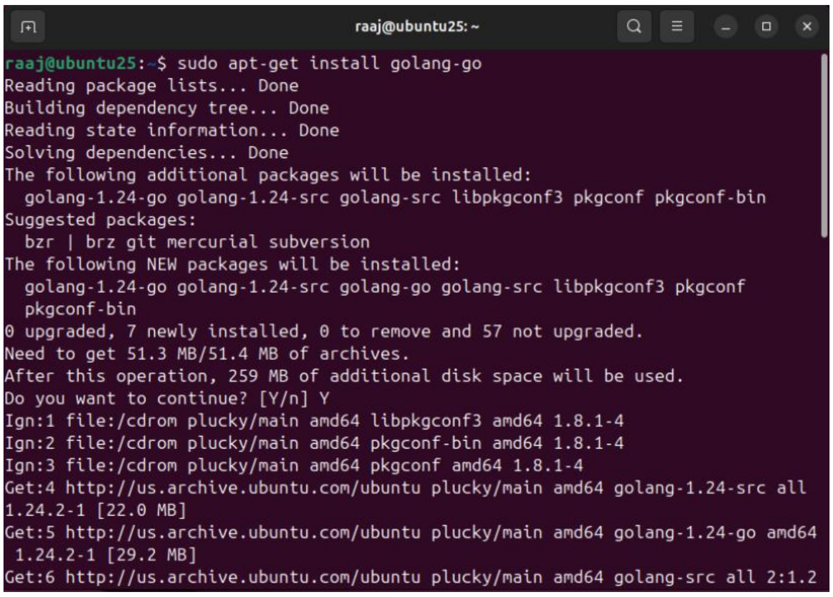
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/python

```
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', 'M
job': 'at_home', 'Fjob': 'teacher', 'reason': 'course', 'guardian': 'mother', 'traveltime': '2', 'studytime': '2', 'fail
ures': '0', 'schoolsup': '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'}
{'school': 'GP', 'sex': 'M', 'age': '19', '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', 'schoolsup': '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': 'yes'}
{'school': 'GP', 'sex': 'F', 'age': '17', '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', 'schoolsup': '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': 'yes'}
Total node: 3
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$
```

Figure 9 – Output of python code for task2

Go code

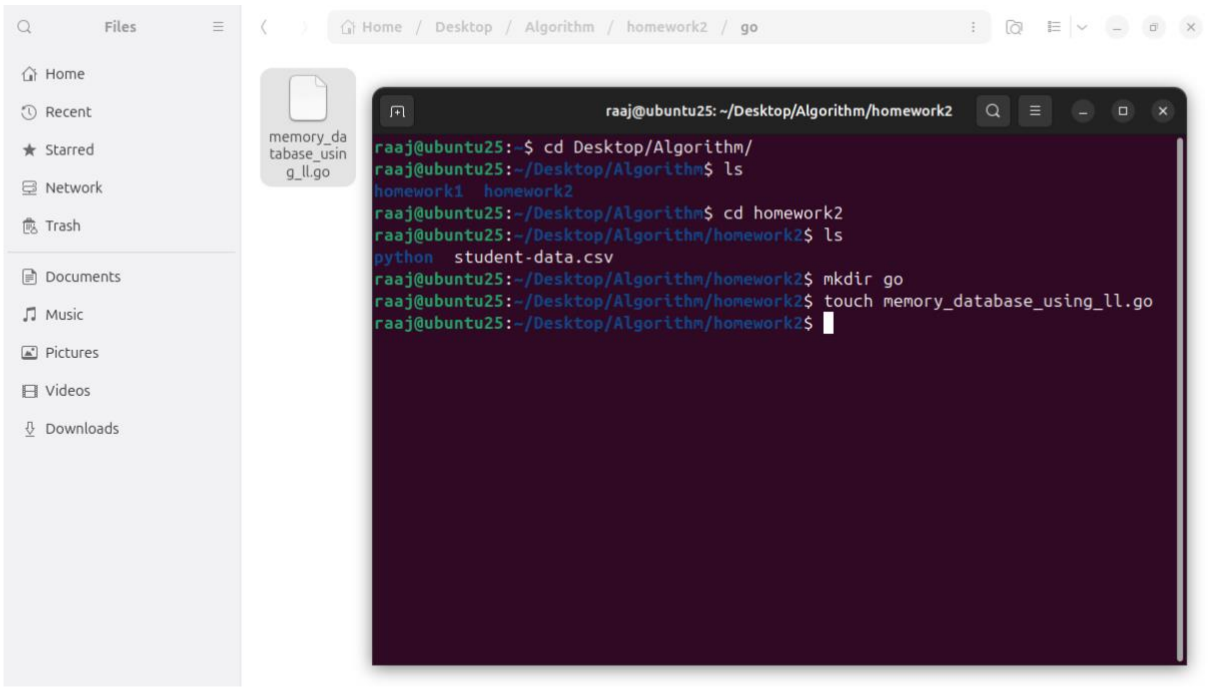
- Install go in ubuntu by using command “sudo apt-get install golang-go”.



```
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”.



```
raaj@ubuntu25:~$ cd Desktop/Algorithm/
raaj@ubuntu25:~/Desktop/Algorithm$ ls
homework1  homework2
raaj@ubuntu25:~/Desktop/Algorithm$ cd homework2
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ ls
python  student-data.csv
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ mkdir go
raaj@ubuntu25:~/Desktop/Algorithm/homework2$ touch memory_database_using_ll.go
raaj@ubuntu25:~/Desktop/Algorithm/homework2$
```

Figure 11 – Make memory_database_using_ll.go file using touch command

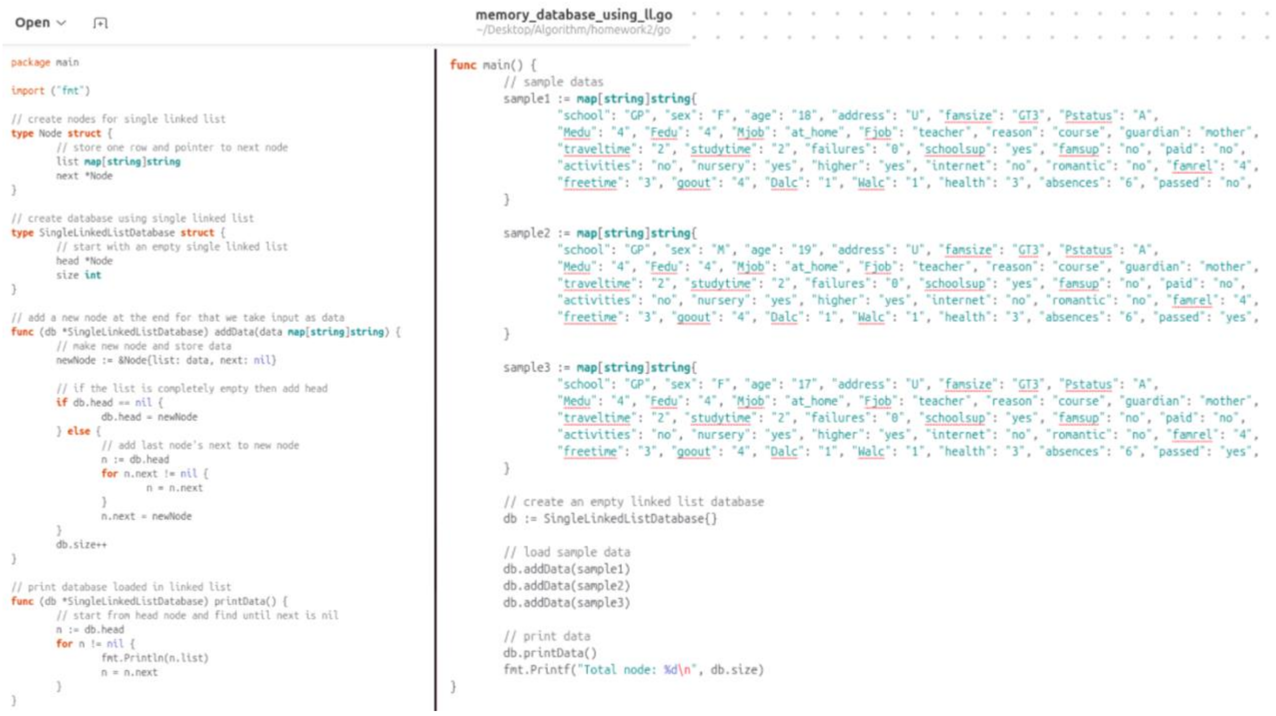
i. Install required go modules

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.



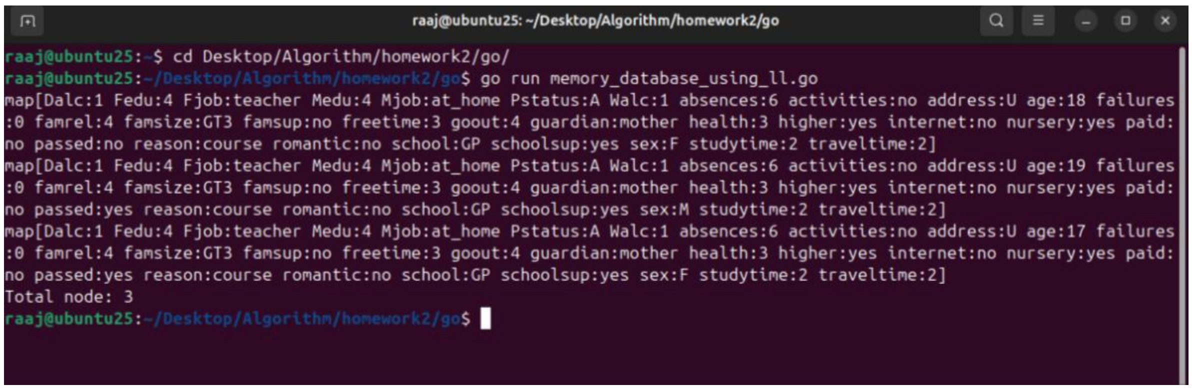
```go
package main

import ("fmt")

// create nodes for single linked list
type Node struct {
	// store one row and pointer to next node
	list map[string]string
	next *Node
}

// create database using single linked list
type SingleLinkedListDatabase struct {
	// start with an empty single linked list
	head *Node
	size int
}

// add a new node at the end for that we take input as data
func (db *SingleLinkedListDatabase) addData(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 to new node
		n := db.head
		for n.next != nil {
			n = n.next
		}
		n.next = newNode
	}
	db.size++
}

// print database loaded in linked list
func (db *SingleLinkedListDatabase) printData() {
	// start from head node and find until next is nil
	n := db.head
	for n != nil {
		fmt.Println(n.list)
		n = n.next
	}
}
```

```go
func main() {
	// sample datas
	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", "studytime": "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",
	}

	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", "studytime": "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",
	}

	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", "studytime": "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",
	}

	// create an empty linked list database
	db := SingleLinkedListDatabase{}

	// load sample data
	db.addData(sample1)
	db.addData(sample2)
	db.addData(sample3)

	// print data
	db.printData()
	fmt.Printf("Total node: %d\n", db.size)
}
```

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.



```
raaj@ubuntu25:~$ cd 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 famsup: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 traveltime: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 famsup: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 traveltime: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 famsup: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 traveltime: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.

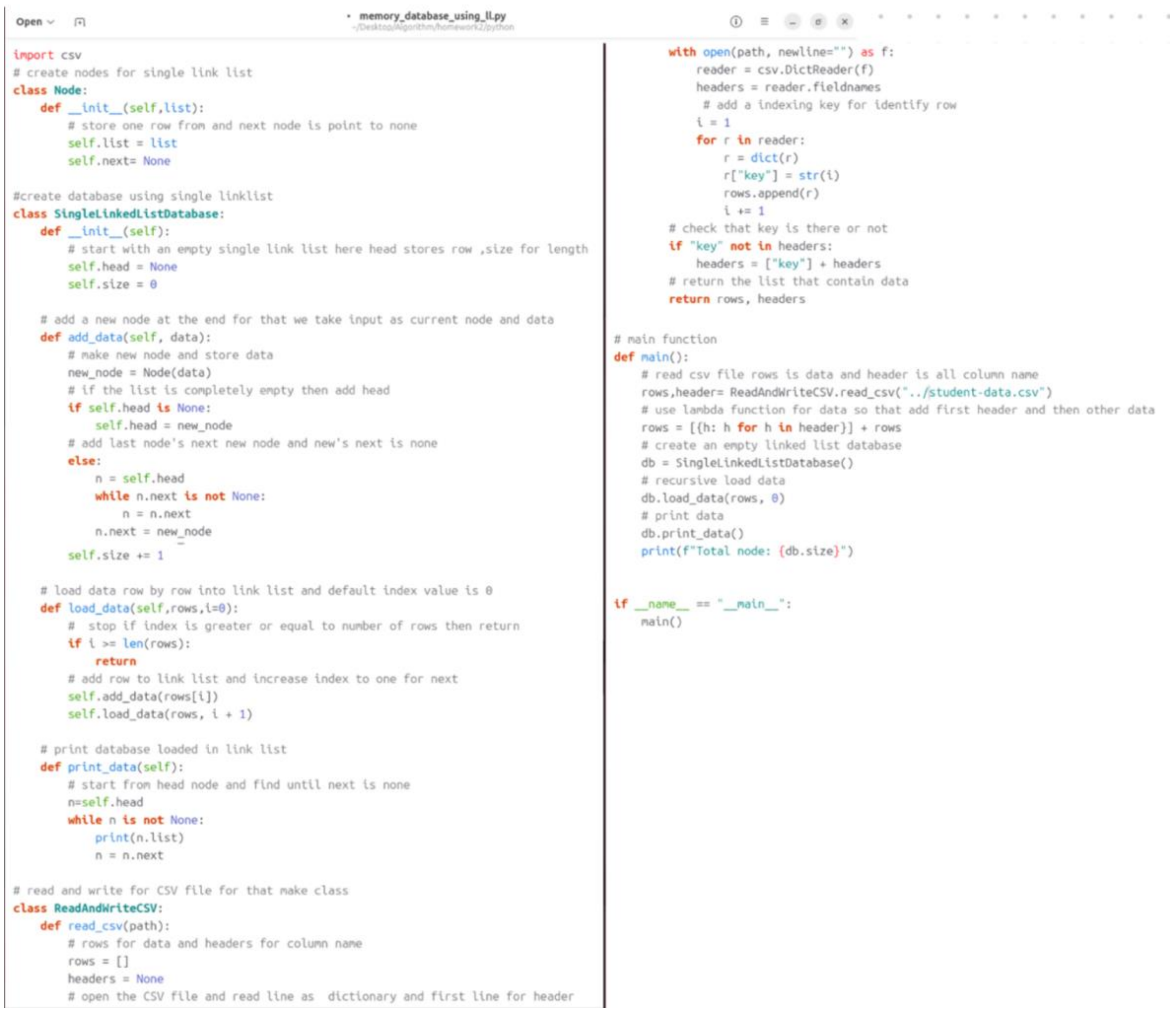
Once the rows are prepared, the SingleLinkedListDatabase.load_data(rows, i=0) method is called This is where recursion comes in. The function checks:

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.

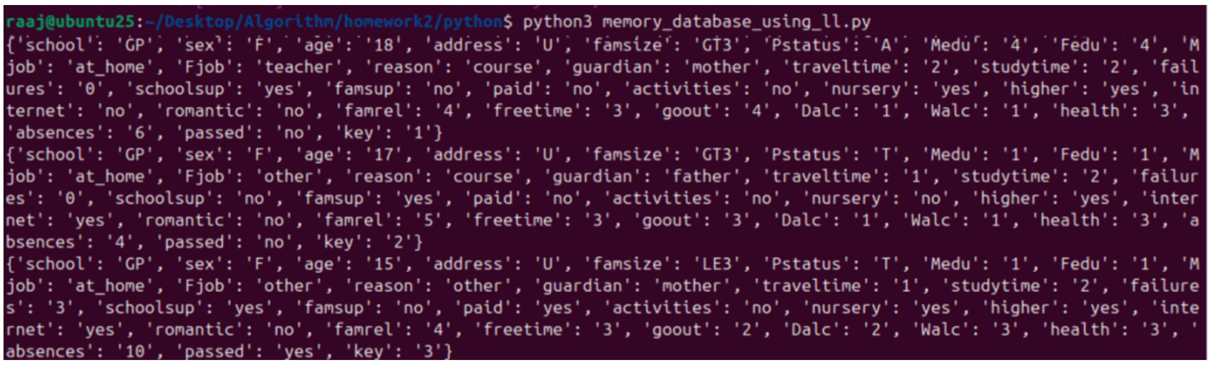
Step by step, the recursion creates one node for each student record until all rows are processed. At the end, the linked list in memory mirrors the CSV file completely.


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

```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 = 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

# 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()
```

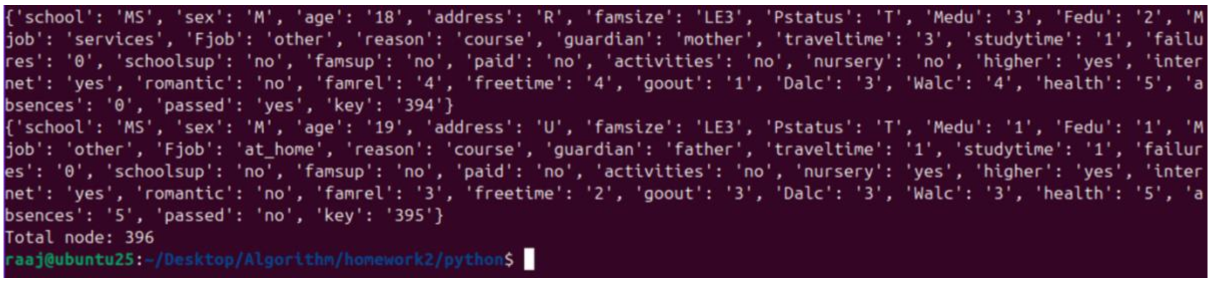
Figure 14 – Screenshot of python code until task 3a

- 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', 'schoolsup': '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', 'schoolsup': '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', 'schoolsup': 'yes', 'famsup': 'no', 'paid': 'yes', 'activities': 'no', 'nursery': 'yes', 'higher': 'yes', 'inte
rnet': 'yes', 'romantic': 'no', 'famrel': '4', 'freetime': '3', 'goout': '2', 'Dalc': '2', 'Walc': '3', 'health': '3', '
absences': '10', 'passed': 'yes', 'key': '3'}
```

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', 'schoolsup': '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', 'a
bsences': '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', 'schoolsup': '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).

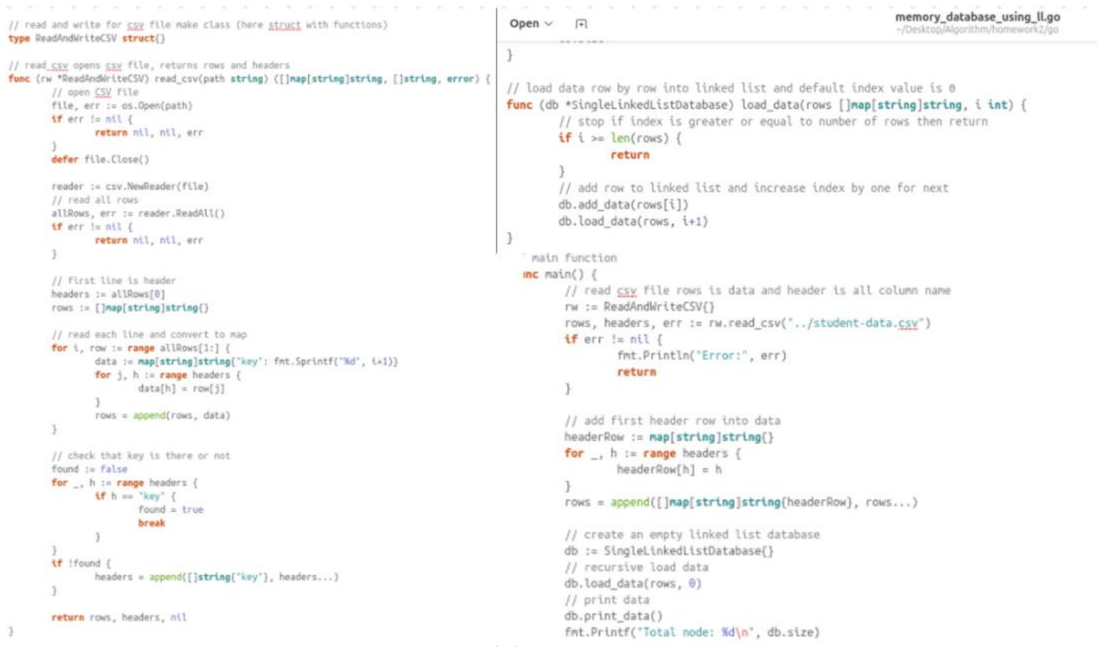
Just like in Python, it works by checking:

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.



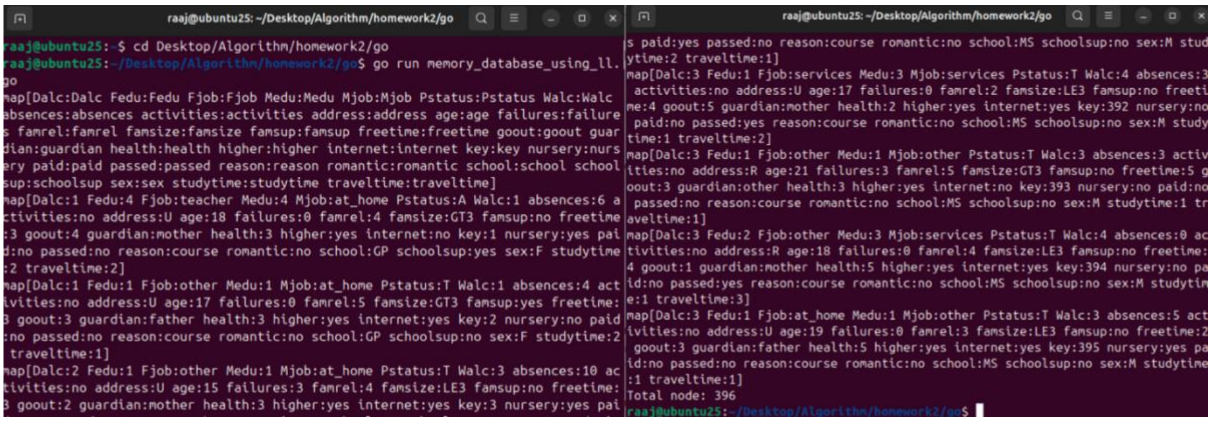
memory_database_using_ll.go
~/Desktop/Algorithm/homework2/go

```go
// 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
}
```

```go
}

// 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
inc 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

```
raaj@ubuntu25:~$ cd Desktop/Algorithm/homework2/go
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
s famrel:famrel famsize:famsize famsup:famsup freetime:freetime goout:goout guar
dian:guardian health:health higher:higher internet:internet key:key nursery:nurs
ery paid:paid passed:passed reason:reason romantic:romantic school:school school
sup:schoolsup sex:sex studytime:studytime traveltime:traveltime]
map[Dalc:1 Fedu:4 Fjob:teacher Medu:4 Mjob:at_home Pstatus:A Walc:1 absences:6 a
ctivities:no address:U age:18 failures:0 famrel:4 famsize:GT3 famsup:no freetime
:3 goout:4 guardian:mother health:3 higher:yes internet:no key:1 nursery:yes pai
d:no passed:no reason:course romantic:no school:GP schoolsup:yes sex:F studytime
:2 traveltime:2]
map[Dalc:1 Fedu:1 Fjob:other Medu:1 Mjob:at_home Pstatus:T Walc:1 absences:4 act
ivities:no address:U age:17 failures:0 famrel:5 famsize:GT3 famsup:yes freetime:
3 goout:3 guardian:father health:3 higher:yes internet:yes key:2 nursery:no paid
:no passed:no reason:course romantic:no school:GP schoolsup:no sex:F studytime:2
 traveltime:1]
map[Dalc:2 Fedu:1 Fjob:other Medu:1 Mjob:at_home Pstatus:T Walc:3 absences:10 ac
tivities:no address:U age:15 failures:3 famrel:4 famsize:LE3 famsup:no freetime:
3 goout:2 guardian:mother health:3 higher:yes internet:yes key:3 nursery:yes pai
```

raaj@ubuntu25: ~/Desktop/Algorithm/homework2/go

```
s paid:yes passed:no reason:course romantic:no school:MS schoolsup:no sex:M stud
ytime:2 traveltime:1]
map[Dalc:3 Fedu:1 Fjob:services Medu:3 Mjob:services Pstatus:T Walc:4 absences:3
 activities:no address:U age:17 failures:0 famrel:2 famsize:LE3 famsup:no freeti
me: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 study
time:1 traveltime:2]
map[Dalc:3 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:3 absences:3 activ
ities:no address:R age:21 failures:3 famrel:5 famsize:GT3 famsup:no freetime:5 g
oout: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 tr
aveltime:1]
map[Dalc:3 Fedu:2 Fjob:other Medu:3 Mjob:services Pstatus:T Walc:4 absences:0 ac
tivities: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 pa
id:no passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytim
e:1 traveltime:3]
map[Dalc:3 Fedu:1 Fjob:at_home Medu:1 Mjob:other Pstatus:T Walc:3 absences:5 act
ivities: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 pa
id:no passed:no reason:course romantic:no school:MS schoolsup:no sex:M studytime
:1 traveltime:1]
Total node: 396
raaj@ubuntu25:~/Desktop/Algorithm/homework2/go$
```

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".

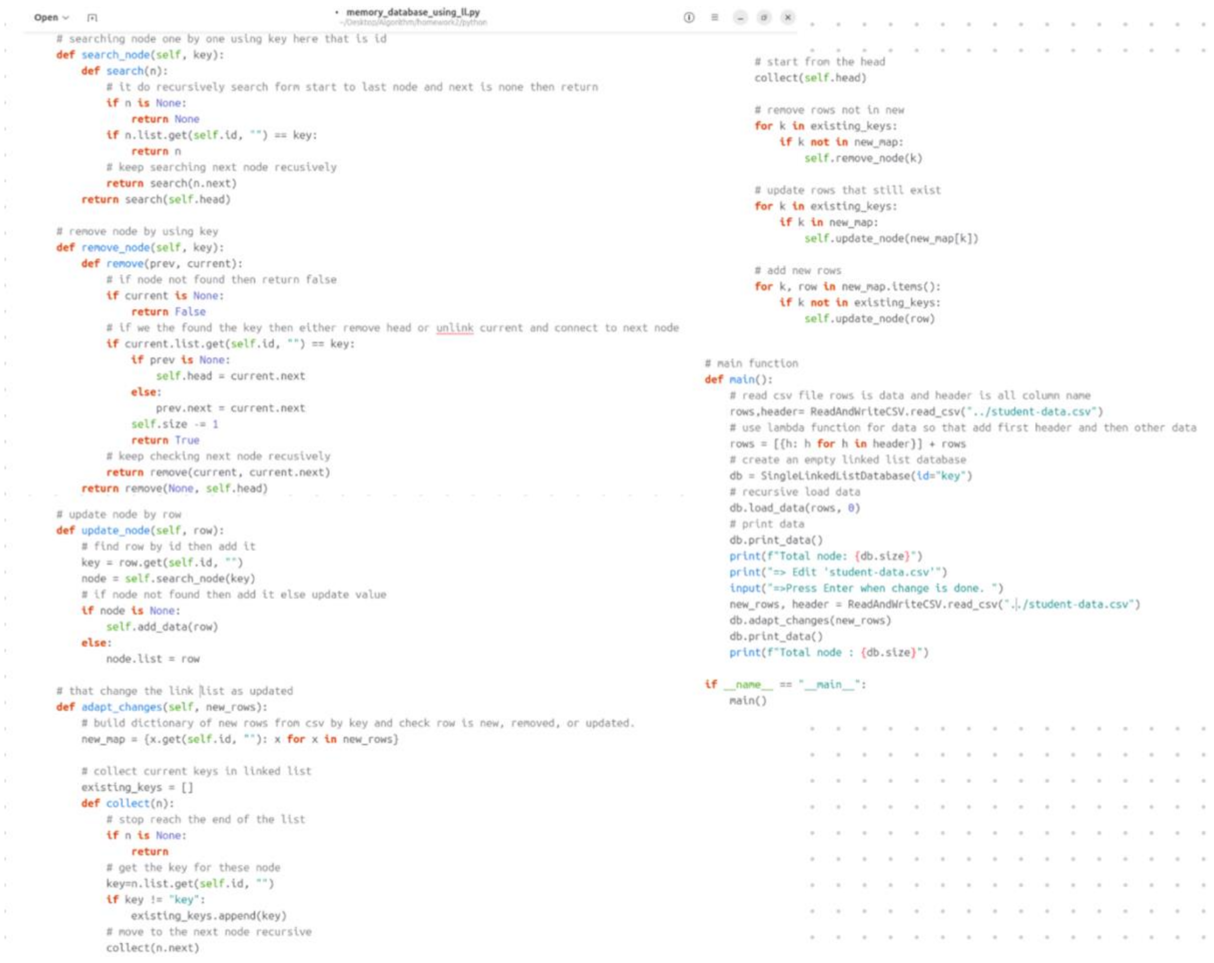
Collect existing keys from the list

- 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.

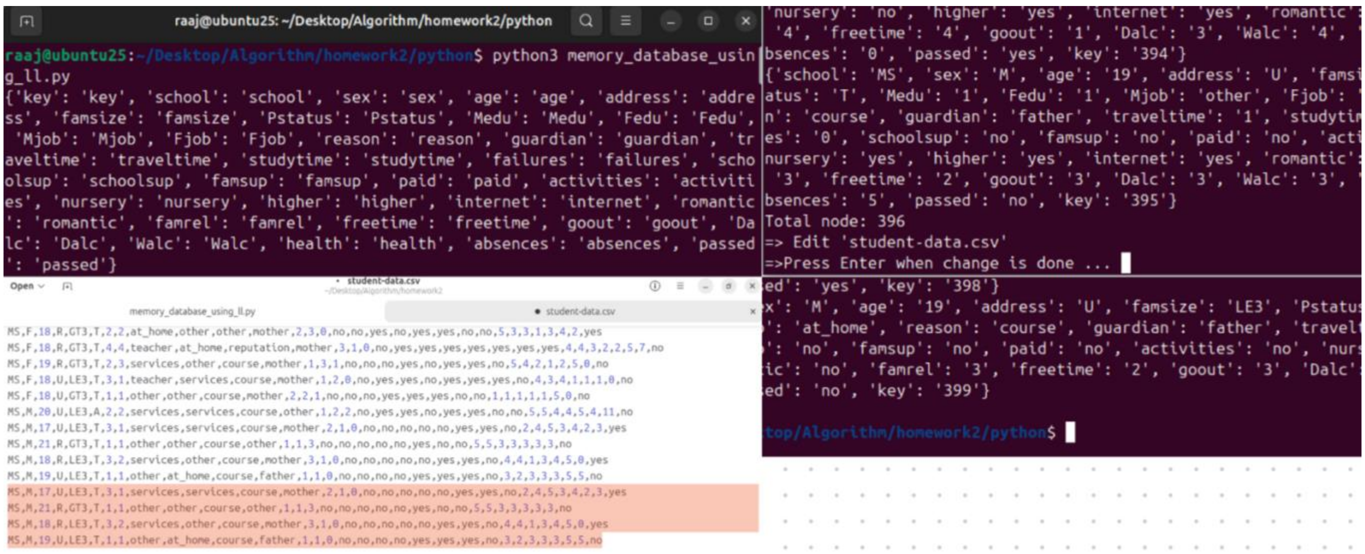


memory_database_using_ll.py

```python
    # 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 recusively
            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(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)

# 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}")

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.



```
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/python
raaj@ubuntu25:~/Desktop/Algorithm/homework2/python$ python3 memory_database_usin
g_ll.py
{'key': 'key', 'school': 'school', 'sex': 'sex', 'age': 'age', 'address': 'addre
ss', 'famsize': 'famsize', 'Pstatus': 'Pstatus', 'Medu': 'Medu', 'Fedu': 'Fedu',
 'Mjob': 'Mjob', 'Fjob': 'Fjob', 'reason': 'reason', 'guardian': 'guardian', 'tr
aveltime': 'traveltime', 'studytime': 'studytime', 'failures': 'failures', 'scho
olsup': 'schoolsup', 'famsup': 'famsup', 'paid': 'paid', 'activities': 'activiti
es', 'nursery': 'nursery', 'higher': 'higher', 'internet': 'internet', 'romantic
': 'romantic', 'famrel': 'famrel', 'freetime': 'freetime', 'goout': 'goout', 'Da
lc': 'Dalc', 'Walc': 'Walc', 'health': 'health', 'absences': 'absences', 'passed
': 'passed'}
```

```
'nursery': 'no', 'higher': 'yes', 'internet': 'yes', 'romantic':
 '4', 'freetime': '4', 'goout': '1', 'Dalc': '3', 'Walc': '4', '
bsences': '0', 'passed': 'yes', 'key': '394'}
{'school': 'MS', 'sex': 'M', 'age': '19', 'address': 'U', 'famsi
atus': 'T', 'Medu': '1', 'Fedu': '1', 'Mjob': 'other', 'Fjob': '
n': 'course', 'guardian': 'father', 'traveltime': '1', 'studyti
es': '0', 'schoolsup': 'no', 'famsup': 'no', 'paid': 'no', 'acti
nursery': 'yes', 'higher': 'yes', 'internet': 'yes', 'romantic':
 '3', 'freetime': '2', 'goout': '3', 'Dalc': '3', 'Walc': '3', '
bsences': '5', 'passed': 'no', 'key': '395'}
Total node: 396
=> Edit 'student-data.csv'
=>Press Enter when change is done ...
```

```
ed': 'yes', 'key': '398'}
x': 'M', 'age': '19', 'address': 'U', 'famsize': 'LE3', 'Pstatus
': 'at_home', 'reason': 'course', 'guardian': 'father', 'travelt
': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nurs
ic': 'no', 'famrel': '3', 'freetime': '2', 'goout': '3', 'Dalc':
ed': 'no', 'key': '399'}

top/Algorithm/homework2/python$
```

student-data.csv
~/Desktop/Algorithm/homework2

memory_database_using_ll.py | student-data.csv

```
MS,F,18,R,GT3,T,2,2,at_home,other,other,mother,2,3,0,no,no,yes,no,yes,yes,no,no,5,3,3,1,3,4,2,yes
MS,F,18,R,GT3,T,4,4,teacher,at_home,reputation,mother,3,1,0,no,yes,yes,yes,yes,yes,yes,yes,4,4,3,2,2,5,7,no
MS,F,19,R,GT3,T,2,3,services,other,course,mother,1,3,1,no,no,no,yes,no,yes,yes,no,5,4,2,1,2,5,0,no
MS,F,18,U,LE3,T,3,1,teacher,services,course,mother,1,2,0,no,yes,yes,no,yes,yes,yes,no,4,3,4,1,1,1,0,no
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
MS,M,20,U,LE3,A,2,2,services,services,course,other,1,2,2,no,yes,yes,no,yes,yes,no,no,5,5,4,4,5,4,11,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
```

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

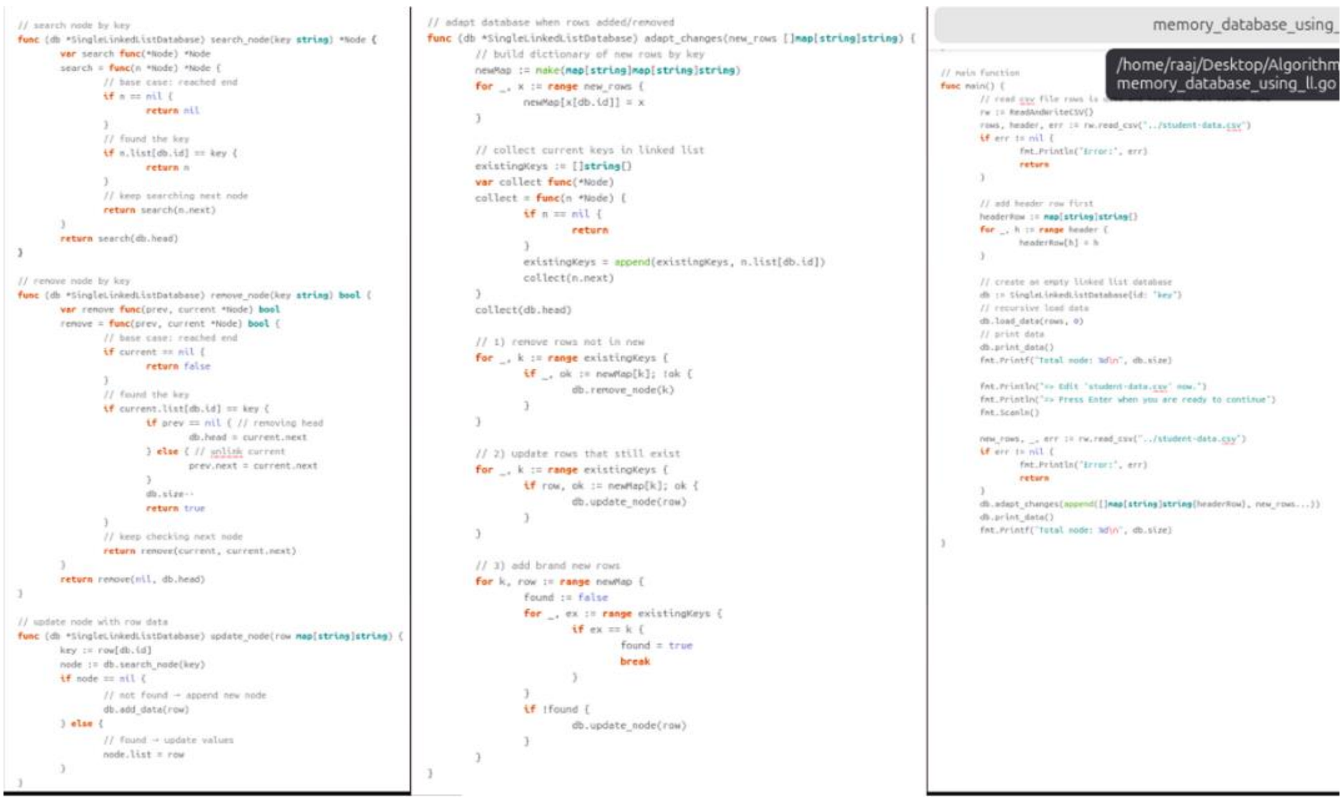
ii Go Implementation

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

1. **Build a lookup of fresh rows** newMap = map[string]map[string]string{} is filled so each "key" points to its new row.
2. **Collect existing keys** 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.



```go
// search node by key
func (db *SingleLinkedListDatabase) search_node(key string) *Node {
	var search func(*Node) *Node
	search = func(n *Node) *Node {
		// base case: reached end
		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 { // unlink 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
	}
}
```

```go
// 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)
		}
	}
}
```

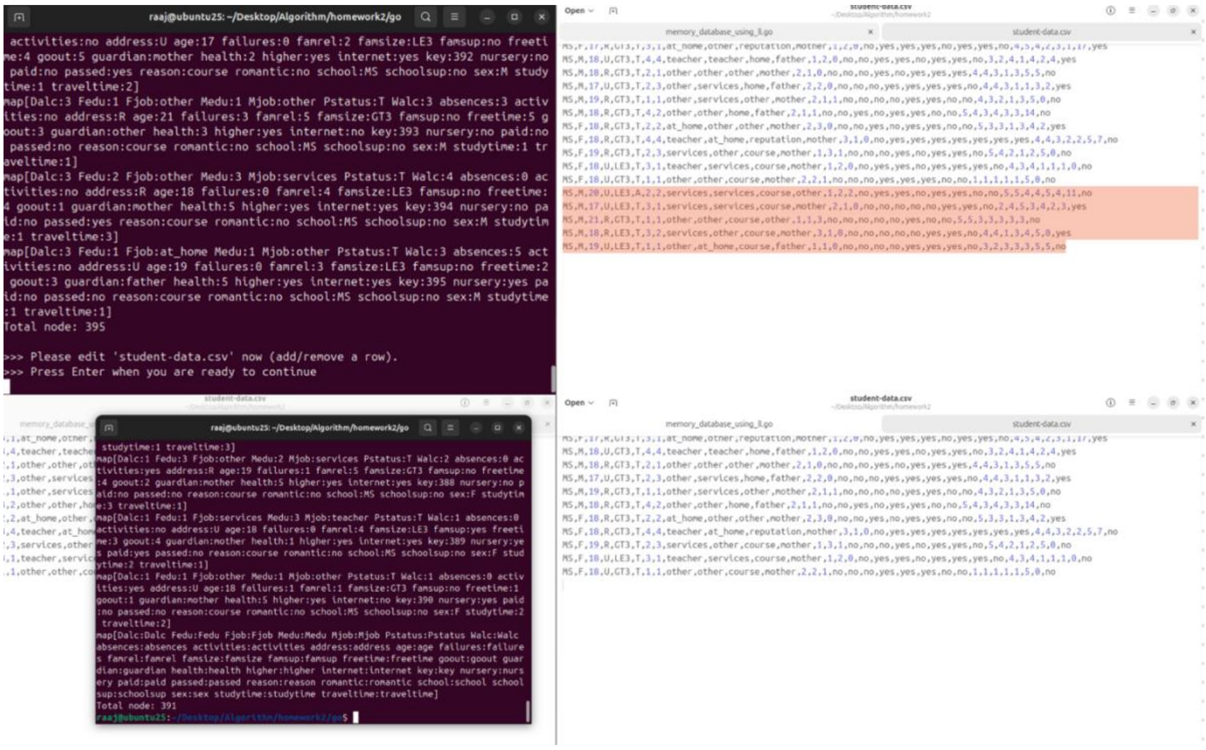
memory_database_using_

/home/raaj/Desktop/Algorithm
memory_database_using_ll.go

```go
// main function
func main() {
	// read csv file rows is
	rw := ReadAndWriteCSV()
	rows, header, err := rw.read_csv("../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 node: %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.read_csv("../student-data.csv")
	if err != nil {
		fmt.Println("Error:", err)
		return
	}
	db.adapt_changes(append([]map[string]string{headerRow}, new_rows...))
	db.print_data()
	fmt.Printf("Total node: %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:no address:U age:17 failures:0 famrel:2 famsize:LE3 famsup:no freeti
me: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 study
time:1 traveltime:2]
map[Dalc:3 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:3 absences:3 activ
ities:no address:R age:21 failures:3 famrel:5 famsize:GT3 famsup:no freetime:5 g
oout: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 tr
aveltime:1]
map[Dalc:3 Fedu:2 Fjob:other Medu:3 Mjob:services Pstatus:T Walc:4 absences:0 ac
tivities: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 pa
id:no passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytim
e:1 traveltime:3]
map[Dalc:3 Fedu:1 Fjob:at_home Medu:1 Mjob:other Pstatus:T Walc:3 absences:5 act
ivities: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 pa
id: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
```

```
student-data.csv
memory_database_using_ll.go    student-data.csv
MS,F,17,R,GT3,T,3,1,at_home,other,reputation,mother,1,2,0,no,yes,yes,yes,no,yes,yes,no,4,5,4,2,3,1,17,yes
MS,M,18,U,GT3,T,4,4,teacher,teacher,home,father,1,2,0,no,no,yes,yes,no,yes,yes,no,3,2,4,1,4,2,4,yes
MS,M,18,R,GT3,T,2,1,other,other,other,mother,2,1,0,no,no,no,yes,no,yes,yes,yes,4,4,3,1,3,5,5,no
MS,M,17,U,GT3,T,2,3,other,services,home,father,2,2,0,no,no,no,yes,yes,yes,yes,no,4,4,3,1,1,3,2,yes
MS,M,19,R,GT3,T,1,1,other,services,other,mother,2,1,1,no,no,no,no,yes,yes,no,no,4,3,2,1,3,5,0,no
MS,M,18,R,GT3,T,4,2,other,other,home,father,2,1,1,no,no,yes,no,yes,yes,no,no,5,4,3,4,3,3,14,no
MS,F,18,R,GT3,T,2,2,at_home,other,other,mother,2,3,0,no,no,yes,no,yes,yes,no,no,5,3,3,1,3,4,2,yes
MS,F,18,R,GT3,T,4,4,teacher,at_home,reputation,mother,3,1,0,no,yes,yes,yes,yes,yes,yes,yes,4,4,3,2,2,5,7,no
MS,F,19,R,GT3,T,2,3,services,other,course,mother,1,3,1,no,no,no,yes,no,yes,yes,no,5,4,2,1,2,5,0,no
MS,F,18,U,LE3,T,3,1,teacher,services,course,mother,1,2,0,no,yes,yes,no,yes,yes,yes,no,4,3,4,1,1,1,0,no
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
MS,M,20,U,LE3,A,2,2,services,services,course,other,1,2,2,no,yes,yes,no,yes,yes,no,no,5,5,4,4,5,4,11,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
```

```
raaj@ubuntu25: ~/Desktop/Algorithm/homework2/go
 studytime:1 traveltime:3]
map[Dalc:1 Fedu:3 Fjob:other Medu:2 Mjob:services Pstatus:T Walc:2 absences:0 ac
tivities:yes address:R age:19 failures:1 famrel:5 famsize:GT3 famsup:no freetime
:4 goout:2 guardian:mother health:5 higher:yes internet:yes key:388 nursery:no p
aid:no passed:no reason:course romantic:no school:MS schoolsup:no sex:F studytim
e:3 traveltime:1]
map[Dalc:1 Fedu:1 Fjob:services Medu:3 Mjob:teacher Pstatus:T Walc:1 absences:0
activities:no address:U age:18 failures:0 famrel:4 famsize:LE3 famsup:yes freeti
me:3 goout:4 guardian:mother health:1 higher:yes internet:yes key:389 nursery:ye
s paid:yes passed:no reason:course romantic:no school:MS schoolsup:no sex:F stud
ytime:2 traveltime:1]
map[Dalc:1 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:1 absences:0 activ
ities:yes address:U age:18 failures:1 famrel:1 famsize:GT3 famsup:no freetime:1
goout:1 guardian:mother health:5 higher:yes internet:no key:390 nursery:yes paid
:no passed:no reason:course romantic:no school:MS schoolsup:no sex:F studytime:2
 traveltime:2]
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
s famrel:famrel famsize:famsize famsup:famsup freetime:freetime goout:goout guar
dian:guardian health:health higher:higher internet:internet key:key nursery:nurs
ery paid:paid passed:passed reason:reason romantic:romantic school:school school
sup:schoolsup sex:sex studytime:studytime traveltime:traveltime]
Total node: 391
raaj@ubuntu25:~/Desktop/Algorithm/homework2/go$
```

```
student-data.csv
memory_database_using_ll.go    student-data.csv
MS,F,17,R,GT3,T,3,1,at_home,other,reputation,mother,1,2,0,no,yes,yes,yes,no,yes,yes,no,4,5,4,2,3,1,17,yes
MS,M,18,U,GT3,T,4,4,teacher,teacher,home,father,1,2,0,no,no,yes,yes,no,yes,yes,no,3,2,4,1,4,2,4,yes
MS,M,18,R,GT3,T,2,1,other,other,other,mother,2,1,0,no,no,no,yes,no,yes,yes,yes,4,4,3,1,3,5,5,no
MS,M,17,U,GT3,T,2,3,other,services,home,father,2,2,0,no,no,no,yes,yes,yes,yes,no,4,4,3,1,1,3,2,yes
MS,M,19,R,GT3,T,1,1,other,services,other,mother,2,1,1,no,no,no,no,yes,yes,no,no,4,3,2,1,3,5,0,no
MS,M,18,R,GT3,T,4,2,other,other,home,father,2,1,1,no,no,yes,no,yes,yes,no,no,5,4,3,4,3,3,14,no
MS,F,18,R,GT3,T,2,2,at_home,other,other,mother,2,3,0,no,no,yes,no,yes,yes,no,no,5,3,3,1,3,4,2,yes
MS,F,18,R,GT3,T,4,4,teacher,at_home,reputation,mother,3,1,0,no,yes,yes,yes,yes,yes,yes,yes,4,4,3,2,2,5,7,no
MS,F,19,R,GT3,T,2,3,services,other,course,mother,1,3,1,no,no,no,yes,no,yes,yes,no,5,4,2,1,2,5,0,no
MS,F,18,U,LE3,T,3,1,teacher,services,course,mother,1,2,0,no,yes,yes,no,yes,yes,yes,no,4,3,4,1,1,1,0,no
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
```

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

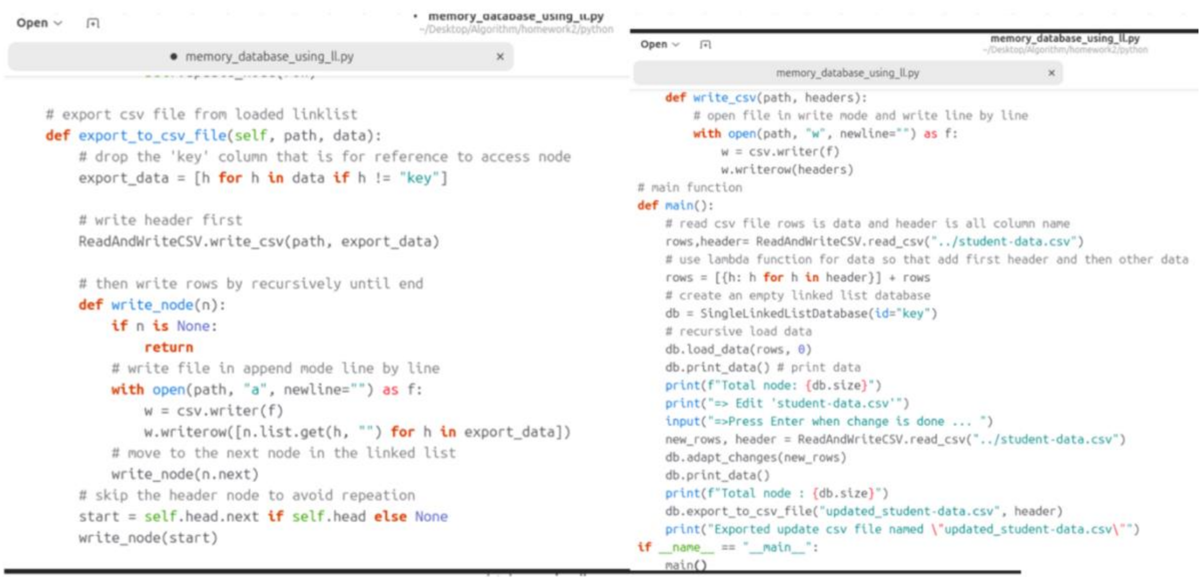
(3c) Export the memory database into a csv file through recursive function

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.

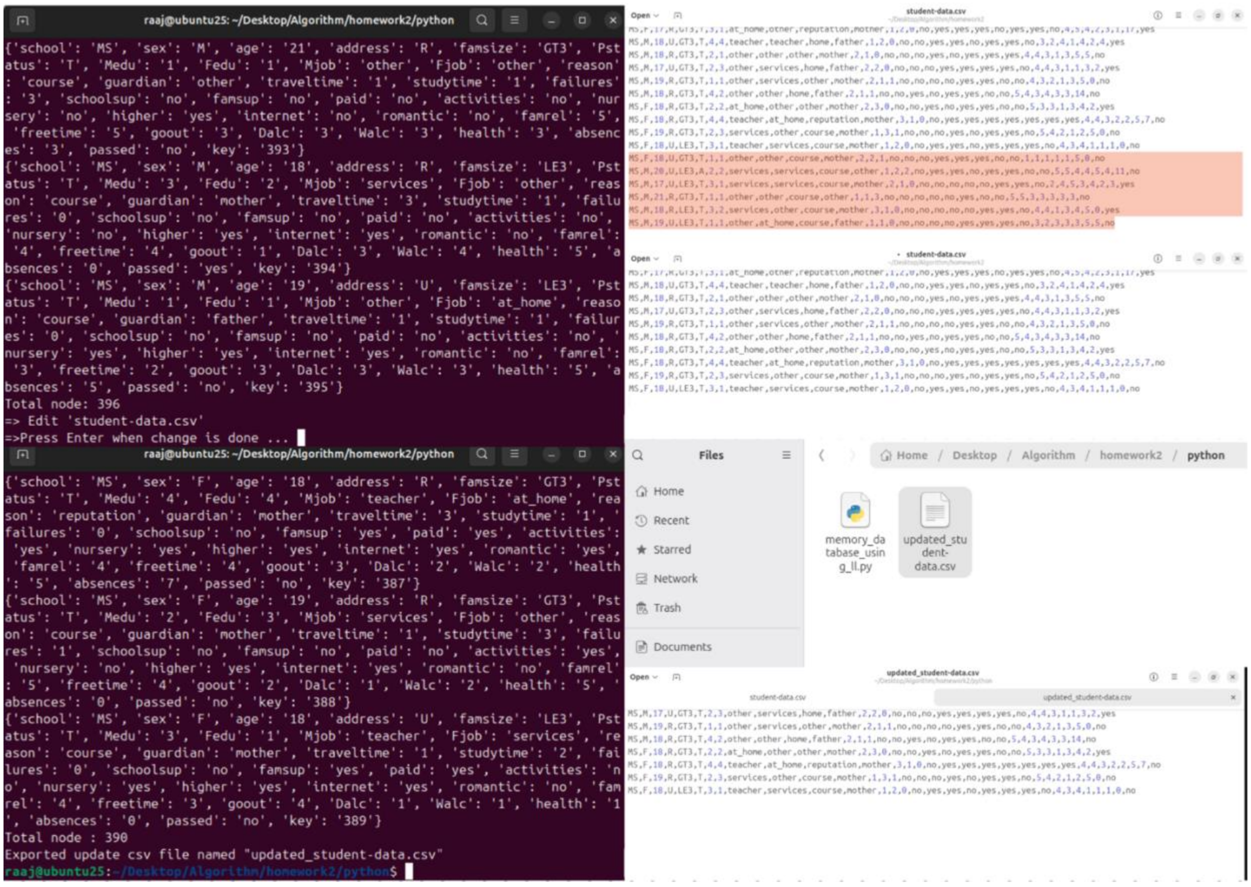


```
    # export csv file from loaded linklist
    def export_to_csv_file(self, path, data):
        # drop the 'key' column that is for reference to access node
        export_data = [h for h in data if h != "key"]

        # 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 repeation
        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")
    # recursive 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', 'Pst
atus': 'T', 'Medu': '1', 'Fedu': '1', 'Mjob': 'other', 'Fjob': 'other', 'reason'
: 'course', 'guardian': 'other', 'traveltime': '1', 'studytime': '1', 'failures'
: '3', 'schoolsup': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', 'nur
sery': 'no', 'higher': 'yes', 'internet': 'no', 'romantic': 'no', 'famrel': '5',
 'freetime': '5', 'goout': '3', 'Dalc': '3', 'Walc': '3', 'health': '3', 'absenc
es': '3', 'passed': 'no', 'key': '393'}
{'school': 'MS', 'sex': 'M', 'age': '18', 'address': 'R', 'famsize': 'LE3', 'Pst
atus': 'T', 'Medu': '3', 'Fedu': '2', 'Mjob': 'services', 'Fjob': 'other', 'reas
on': 'course', 'guardian': 'mother', 'traveltime': '3', 'studytime': '1', 'failu
res': '0', 'schoolsup': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no',
'nursery': 'no', 'higher': 'yes', 'internet': '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': '19', 'address': 'U', 'famsize': 'LE3', 'Pst
atus': 'T', 'Medu': '1', 'Fedu': '1', 'Mjob': 'other', 'Fjob': 'at_home', 'reaso
n': 'course', 'guardian': 'father', 'traveltime': '1', 'studytime': '1', 'failur
es': '0', 'schoolsup': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'no', '
nursery': 'yes', 'higher': 'yes', 'internet': '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
=> Edit 'student-data.csv'
=>Press Enter when change is done ...
```

```
student-data.csv
MS,F,17,R,GT3,T,3,1,at_home,other,reputation,mother,1,2,0,no,yes,yes,yes,no,yes,yes,no,4,5,4,2,3,1,17,yes
MS,M,18,U,GT3,T,4,4,teacher,teacher,home,father,1,2,0,no,no,yes,yes,no,yes,yes,no,3,2,4,1,4,2,4,yes
MS,M,18,R,GT3,T,2,1,other,other,other,mother,2,1,0,no,no,no,yes,no,yes,yes,yes,4,4,3,1,3,5,5,no
MS,M,17,U,GT3,T,2,3,other,services,home,father,2,2,0,no,no,no,yes,yes,yes,yes,no,4,4,3,1,1,3,2,yes
MS,M,19,R,GT3,T,1,1,other,services,other,mother,2,1,1,no,no,no,no,yes,yes,no,no,4,3,2,1,3,5,0,no
MS,M,18,R,GT3,T,4,2,other,other,home,father,2,1,1,no,no,yes,no,yes,yes,no,no,5,4,3,4,3,3,14,no
MS,F,18,R,GT3,T,2,2,at_home,other,other,mother,2,3,0,no,no,yes,no,yes,yes,no,no,5,3,3,1,3,4,2,yes
MS,F,18,R,GT3,T,4,4,teacher,at_home,reputation,mother,3,1,0,no,yes,yes,yes,yes,yes,yes,yes,4,4,3,2,2,5,7,no
MS,F,19,R,GT3,T,2,3,services,other,course,mother,1,3,1,no,no,no,yes,no,yes,yes,no,5,4,2,1,2,5,0,no
MS,F,18,U,LE3,T,3,1,teacher,services,course,mother,1,2,0,no,yes,yes,no,yes,yes,yes,no,4,3,4,1,1,1,0,no
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
MS,M,20,U,LE3,A,2,2,services,services,course,other,1,2,2,no,yes,yes,no,yes,yes,no,no,5,5,4,4,5,4,11,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
```

```
• student-data.csv
MS,F,17,R,GT3,T,3,1,at_home,other,reputation,mother,1,2,0,no,yes,yes,yes,no,yes,yes,no,4,5,4,2,3,1,17,yes
MS,M,18,U,GT3,T,4,4,teacher,teacher,home,father,1,2,0,no,no,yes,yes,no,yes,yes,no,3,2,4,1,4,2,4,yes
MS,M,18,R,GT3,T,2,1,other,other,other,mother,2,1,0,no,no,no,yes,no,yes,yes,yes,4,4,3,1,3,5,5,no
MS,M,17,U,GT3,T,2,3,other,services,home,father,2,2,0,no,no,no,yes,yes,yes,yes,no,4,4,3,1,1,3,2,yes
MS,M,19,R,GT3,T,1,1,other,services,other,mother,2,1,1,no,no,no,no,yes,yes,no,no,4,3,2,1,3,5,0,no
MS,M,18,R,GT3,T,4,2,other,other,home,father,2,1,1,no,no,yes,no,yes,yes,no,no,5,4,3,4,3,3,14,no
MS,F,18,R,GT3,T,2,2,at_home,other,other,mother,2,3,0,no,no,yes,no,yes,yes,no,no,5,3,3,1,3,4,2,yes
MS,F,18,R,GT3,T,4,4,teacher,at_home,reputation,mother,3,1,0,no,yes,yes,yes,yes,yes,yes,yes,4,4,3,2,2,5,7,no
MS,F,19,R,GT3,T,2,3,services,other,course,mother,1,3,1,no,no,no,yes,no,yes,yes,no,5,4,2,1,2,5,0,no
MS,F,18,U,LE3,T,3,1,teacher,services,course,mother,1,2,0,no,yes,yes,no,yes,yes,yes,no,4,3,4,1,1,1,0,no
```

```
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', 'rea
son': 'reputation', 'guardian': 'mother', 'traveltime': '3', 'studytime': '1', '
failures': '0', 'schoolsup': 'no', 'famsup': 'yes', 'paid': 'yes', 'activities':
 'yes', 'nursery': 'yes', 'higher': 'yes', 'internet': 'yes', 'romantic': 'yes',
 'famrel': '4', 'freetime': '4', 'goout': '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', 'studytime': '3', 'failu
res': '1', 'schoolsup': 'no', 'famsup': 'no', 'paid': 'no', 'activities': 'yes',
 'nursery': 'no', 'higher': 'yes', 'internet': 'yes', 'romantic': 'no', 'famrel'
: '5', 'freetime': '4', 'goout': '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', 're
ason': 'course', 'guardian': 'mother', 'traveltime': '1', 'studytime': '2', 'fai
lures': '0', 'schoolsup': 'no', 'famsup': 'yes', 'paid': 'yes', 'activities': 'n
o', 'nursery': 'yes', 'higher': 'yes', 'internet': 'yes', 'romantic': 'no', 'fam
rel': '4', 'freetime': '3', 'goout': '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$
```

Files
Home
Recent
Starred
Network
Trash
Documents
Home / Desktop / Algorithm / homework2 / python
memory_database_using_ll.py
updated_student-data.csv

```
updated_student-data.csv
student-data.csv    updated_student-data.csv
MS,M,17,U,GT3,T,2,3,other,services,home,father,2,2,0,no,no,no,yes,yes,yes,yes,no,4,4,3,1,1,3,2,yes
MS,M,19,R,GT3,T,1,1,other,services,other,mother,2,1,1,no,no,no,no,yes,yes,no,no,4,3,2,1,3,5,0,no
MS,M,18,R,GT3,T,4,2,other,other,home,father,2,1,1,no,no,yes,no,yes,yes,no,no,5,4,3,4,3,3,14,no
MS,F,18,R,GT3,T,2,2,at_home,other,other,mother,2,3,0,no,no,yes,no,yes,yes,no,no,5,3,3,1,3,4,2,yes
MS,F,18,R,GT3,T,4,4,teacher,at_home,reputation,mother,3,1,0,no,yes,yes,yes,yes,yes,yes,yes,4,4,3,2,2,5,7,no
MS,F,19,R,GT3,T,2,3,services,other,course,mother,1,3,1,no,no,no,yes,no,yes,yes,no,5,4,2,1,2,5,0,no
MS,F,18,U,LE3,T,3,1,teacher,services,course,mother,1,2,0,no,yes,yes,no,yes,yes,yes,no,4,3,4,1,1,1,0,no
```

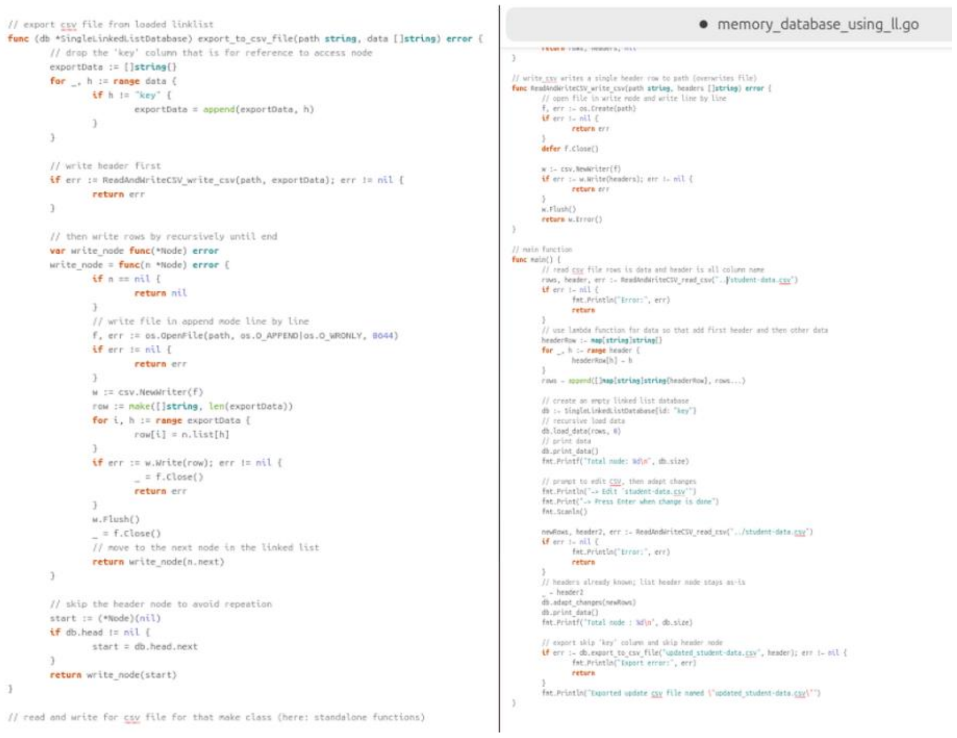
Figure 24 – Output that shows line removed and new file created and that is updated

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.



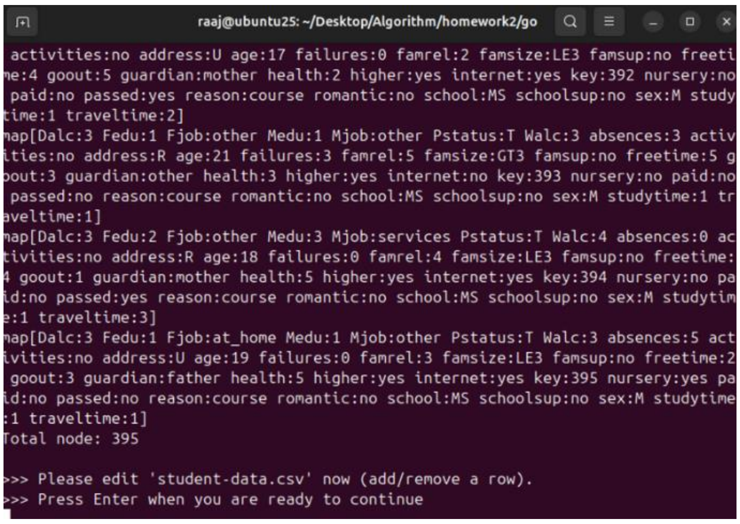
● memory_database_using_ll.go

```go
// 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 repeation
	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)
```

```go
		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\"")
}
```

Figure 25 – Screenshot of go code until task 3c

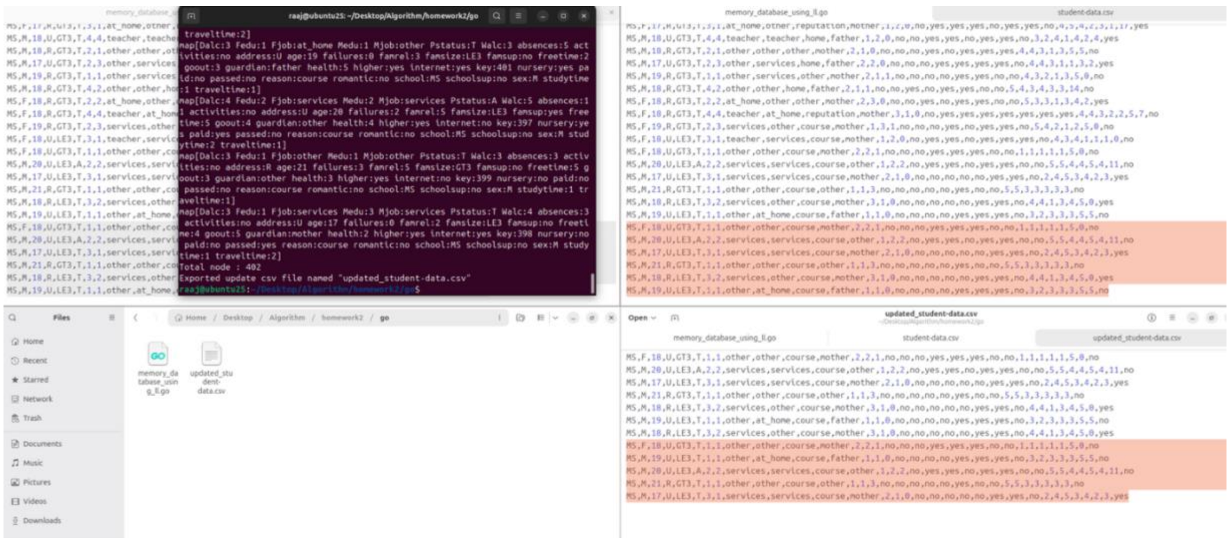
The Go program follows the same logic. After reloading and adapting changes to the linked list, the export_to_csv_file() method is executed. This method recursively walks through the list, writing each node's data into a new CSV file. As in Python, the "key" field is ignored during export, leaving only 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 freeti
me: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 study
time:1 traveltime:2]
map[Dalc:3 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:3 absences:3 activ
ities:no address:R age:21 failures:3 famrel:5 famsize:GT3 famsup:no freetime:5 g
oout: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 tr
aveltime:1]
map[Dalc:3 Fedu:2 Fjob:other Medu:3 Mjob:services Pstatus:T Walc:4 absences:0 ac
tivities: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 pa
id:no passed:yes reason:course romantic:no school:MS schoolsup:no sex:M studytim
e:1 traveltime:3]
map[Dalc:3 Fedu:1 Fjob:at_home Medu:1 Mjob:other Pstatus:T Walc:3 absences:5 act
ivities: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 pa
id: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



```
 traveltime:2]
map[Dalc:3 Fedu:1 Fjob:at_home Medu:1 Mjob:other Pstatus:T Walc:3 absences:5 act
ivities: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:401 nursery:yes pa
id:no passed:no reason:course romantic:no school:MS schoolsup:no sex:M studytime
:1 traveltime:1]
map[Dalc:4 Fedu:2 Fjob:services Medu:2 Mjob:services Pstatus:A Walc:5 absences:1
1 activities:no address:U age:20 failures:2 famrel:5 famsize:LE3 famsup:yes free
time:5 goout:4 guardian:other health:4 higher:yes internet:no key:397 nursery:ye
s paid:yes passed:no reason:course romantic:no school:MS schoolsup:no sex:M stud
ytime:2 traveltime:1]
map[Dalc:3 Fedu:1 Fjob:other Medu:1 Mjob:other Pstatus:T Walc:3 absences:3 activ
ities:no address:R age:21 failures:3 famrel:5 famsize:GT3 famsup:no freetime:5 g
oout:3 guardian:other health:3 higher:yes internet:no key:399 nursery:no paid:no
 passed:no reason:course romantic:no school:MS schoolsup:no sex:M studytime:1 tr
aveltime:1]
map[Dalc:3 Fedu:1 Fjob:services Medu:3 Mjob:services Pstatus:T Walc:4 absences:3
 activities:no address:U age:17 failures:0 famrel:2 famsize:LE3 famsup:no freeti
me:4 goout:5 guardian:mother health:2 higher:yes internet:yes key:398 nursery:no
 paid:no passed:yes reason:course romantic:no school:MS schoolsup:no sex:M study
time:1 traveltime:2]
Total node : 402
Exported update csv file named "updated_student-data.csv"
raaj@ubuntu25:~/Desktop/Algorithm/homework2/go$
```

```
MS,F,17,R,GT3,T,3,1,at_home,other,reputation,mother,1,2,0,no,yes,yes,yes,no,yes,yes,no,4,5,4,2,3,1,17,yes
MS,M,18,U,GT3,T,4,4,teacher,teacher,home,father,1,2,0,no,no,yes,yes,no,yes,yes,no,3,2,4,1,4,2,4,yes
MS,M,18,R,GT3,T,2,1,other,other,other,mother,2,1,0,no,no,no,yes,no,yes,yes,yes,4,4,3,1,3,5,5,no
MS,M,17,U,GT3,T,2,3,other,services,home,father,2,2,0,no,no,no,yes,yes,yes,yes,no,4,4,3,1,1,3,2,yes
MS,M,19,R,GT3,T,1,1,other,services,other,mother,2,1,1,no,no,no,no,yes,yes,no,no,4,3,2,1,3,5,0,no
MS,M,18,R,GT3,T,4,2,other,other,home,father,2,1,1,no,no,yes,no,yes,yes,no,no,5,4,3,4,3,3,14,no
MS,F,18,R,GT3,T,2,2,at_home,other,other,mother,2,3,0,no,no,yes,no,yes,yes,no,no,5,3,3,1,3,4,2,yes
MS,F,18,R,GT3,T,4,4,teacher,at_home,reputation,mother,3,1,0,no,yes,yes,yes,yes,yes,yes,yes,4,4,3,2,2,5,7,no
MS,F,19,R,GT3,T,2,3,services,other,course,mother,1,3,1,no,no,no,yes,no,yes,yes,no,5,4,2,1,2,5,0,no
MS,F,18,U,LE3,T,3,1,teacher,services,course,mother,1,2,0,no,yes,yes,no,yes,yes,yes,no,4,3,4,1,1,1,0,no
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
MS,M,20,U,LE3,A,2,2,services,services,course,other,1,2,2,no,yes,yes,no,yes,yes,no,no,5,5,4,4,5,4,11,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
MS,M,20,U,LE3,A,2,2,services,services,course,other,1,2,2,no,yes,yes,no,yes,yes,no,no,5,5,4,4,5,4,11,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
```

```
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
MS,M,20,U,LE3,A,2,2,services,services,course,other,1,2,2,no,yes,yes,no,yes,yes,no,no,5,5,4,4,5,4,11,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
MS,M,18,R,LE3,T,3,2,services,other,course,mother,3,1,0,no,no,no,no,no,yes,yes,no,4,4,1,3,4,5,0,yes
MS,F,18,U,GT3,T,1,1,other,other,course,mother,2,2,1,no,no,no,yes,yes,yes,no,no,1,1,1,1,1,5,0,no
MS,M,19,U,LE3,T,1,1,other,at_home,course,father,1,1,0,no,no,no,no,yes,yes,yes,no,3,2,3,3,3,5,5,no
MS,M,20,U,LE3,A,2,2,services,services,course,other,1,2,2,no,yes,yes,no,yes,yes,no,no,5,5,4,4,5,4,11,no
MS,M,21,R,GT3,T,1,1,other,other,course,other,1,1,3,no,no,no,no,no,yes,no,no,5,5,3,3,3,3,3,no
MS,M,17,U,LE3,T,3,1,services,services,course,mother,2,1,0,no,no,no,no,no,yes,yes,no,2,4,5,3,4,2,3,yes
```

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. It's 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 defaullt 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 recusively
        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 repeation
        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 defaullt 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 recusively
                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 recusively
				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 repeation
```

```
	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\"")
}
```