Individual HW: 02

CS51510-001, fall 2025

Purdue University Northwest

09/19/2025

In-Memory Database

Name	Email
Kowshick Srinivasan	**Srini252@pnw.edu**

Table of Contents

Table of Figures

Table of Pseudo Code

Abstract
Design and implement a memory database using singly Linked List data structure, that is able to read the student data from the CSV file, adapt to any changes done in the CSV file and write it a output CSV file.

Introduction
This document is part of Algorithms course (**51510-001**), individual **homework-2**. The goal of the document is to create a memory database using Linked list that can store student details from a CSV file, Adapt to changes that are done to the CSV file, and write the output in a new CSV file.

We first download the student dataset from the given website

https://www.kaggle.com/datasets/kellygakii/student-data-csv. The implementation of this memory database has been done in two programming languages **JAVA** and **PYTHON**

The below document explains the User requirement, software design, Student class- which defines the attributes of a student (from CSV), Then it explains the singly Linked List class and it functionalities, leading us to the implementation of the memory database with the given constraints, finally we end the document with the results screenshot and sample test cases to prove the working of the memory database.

User Story
I want a lightweight **in-memory** database to store **variable** number of student records, I should be able to read and store the student records from a **CSV file** and the database to adopt to any insertions or deletion in the CSV file **automatically** and finally I should be able to **export** the database to a CSV file.

1.1 Acceptance criteria
- **Recursively** read the *student-data.csv* file to the memory database
- Adapt to any changes in the CSV file
- Export the in-memory database to a CSV file
- Should be able to run in **Ubuntu** environment
- Need the solution in **Java** and **Python**
- Provide output Screen shots to prove the working

1. Software Design
We are given with five constraints to follow while building the memory database and there are as follows,

- Use a **Singly linked List** for the in-memory database as the size of the dataset can scale in real-time
- Follow **Recursive** functions since the user specified it
- Use **file watcher** service to observe if there is any file modification change and reload the database
- Reload **only** the changes not the entire database
- Provide **Object-oriented** solution using Java and Python

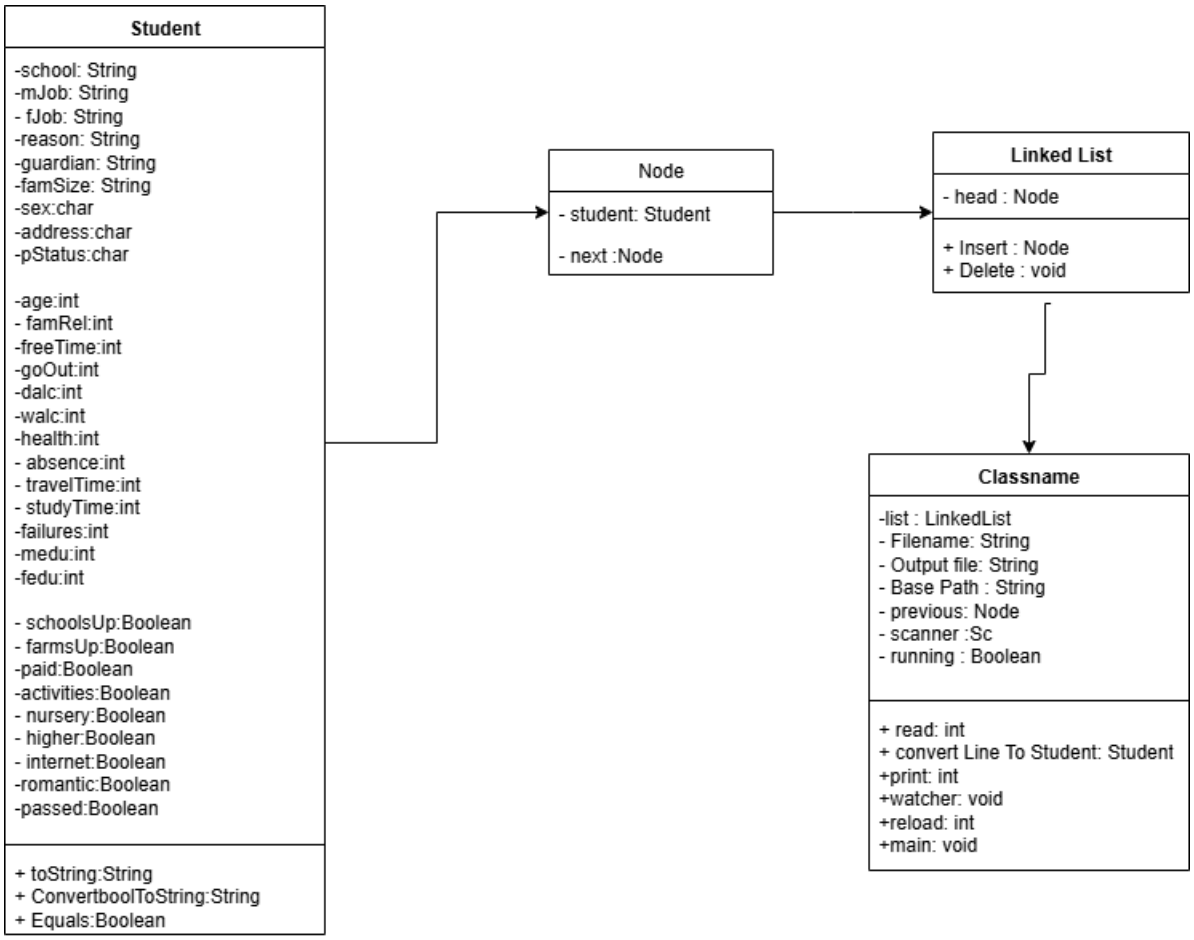


Figure 1-Class diagram

2. Student Class

this is the blueprint of the student details that are supposed to be read and stored in the Linked List. From the given CSV file we can see there are 31 features to a student which included school, age, sex and paid.

Class Student

String school;

.

```
.
Character sex;
.
.
Integer age;
.
.
Boolean paid;
String ConvertToString (BooleanVariable)
Boolean Equals(student2)
String toString(student)
```

Pseudo-code 1: Student Class

The class also contains a constructer method to convert the read CSV row (CSV reader returns a comma separated String line) to Student object. Two student attributes/methods one to convert the Student object to comma separated String, which can be written back to the CSV file and an equals methods that checks the equality of two student objects.

Note not all the attributes of the Student class are String objects, some are of primitive data types likes char, Int and Boolean. This allocation of different data types is to improve the program efficiency by reducing the memory required by String data type.

1.1 Equals

It is a overwritten method of the equals() method provided by the Object class. By default is compares two object's memory location if same return equal (shallow comparison). We over write the method to compare the member variables of the two objects and say they are equal or not (deep comparison)

```
Boolean equals(obj)
    Return school == obj.school && age == obj.age && sex =obj.sex ….. ;
```

Pseudo-code 2- Compare two objects Method

1.2 Convert Student to String

It is a overwritten method of the toString() methods provided by the Object class. By default it converts the object to string so that it can be used to print/debug/ We overwrite the method to return the Object in a specific string format (Comma separated values)

```
String toString(student)
```

```
        Return school +”,’+sex+”,”+”,’+age+”,”+address+…….”
```

Pseudo-code 3- Convert to string

Note we don’t need to specify the keyword this to specify to point to the class variable, since it is a member method

1.3 Convert Boolean to string

The user understands only the way that is specified in the input, we can convert the value and store in the memory but it is important to speak the same language of the user. The user document contains “yes” “no” instead of True/False, so the output should also be in the same language.

2. Linked List class

This class defined a singly linked list, with a static inner class which is used to define a Node.

The inner class Node define how a Linked list node should be like, Since it is a singly Node, it contains the student data and a pointer to the next Node. It also has a constructor which initializes the Node object with the student data and always the next Node address is set to null be default.

```
Static inner class Node
        Student student;
        Node next;
        Node(student)
                This.student=student
                This.next=null
```

Pseudo-code 4- Node Class

The outer class defined the structure of the Linked list on whole, initially the Linked list only contains the head Node and we can chain in any number of Nodes as and when needed. So the class only contains a head of type Node, Along with the function like insertion of a new node at the end of Linked List.

```
Class LinkedList
        Node head;
        Static inner class Node{…}
        Node insert(currentNode , student, nextNode)
        Node delete(prevNode,nextNode)
```

Pseudo-code 5- Linked List Class

2.1. Insert Method

This class method is used to insert a node as the head Node if List is empty or at the back of an existing Linked List; this accepts current last nod, the Student details object and the next Node (If there is no next Node it is null). It returns the updated last node, i.e. the current node.

```
Node insert(currentNode, student, nextNode)
        newStudent = Node(student)
        newStudent.next = nextNode
        if head==null
                head=newStudent
        else
                currentNode.next=newStudent
        return newStudent
```

Pseudo-code 6- Insert new node

2.2 Delete Method

This class method is used to delete the Node. We mean delete is we unlink the node from the linked list so that the garbage collector can remove the hanging Node. This method accepts the previous node and the next node, so that the pointer of the previous node can be set to the next node, making the current node hanging.

```
Delete (previousNode, nextNode)
        If previousNode = null     //delete the head node
                Head = nextNode
        Else
                previousNode.next = nextNode
```

Pseudo-code 7- Delete a node

3. Memory Database

This is the class that is responsible for creating and maintaining the memory database using the linkedList class. It has 4 member variables, LinkedList, inputFile, output file, a condion variable and a scanner object. It also has a constructor to instantiate the member variables (We use constructor based dependency injection) along with the following methods.

```
Class MemoryDatabase
        LinkedList() list
```

```
String input file
String output file
Scanner sc
Boolean condition
MemoryDatabase(LinkedList list, inputFile,OutputFile,Scanner sc)
        This.list =list
        This.inputFile = inputfile;
        This.outputFile= outputFile
        This.sc =sc
        This.condition = true

<Following methods>
```

Pseudo-code 8- Memory Database class

3.1. Read the CSV file

Firstly we convert the CSV file to a inbuilt Reader object, to facilitate successful reading of the file.

```
BufferedReader br = new BufferedReader(new FileReader(<Input.csv>)
```

We can now invoke the readline() method of the Reader class to read the row one by one. This method reads one row at a time and moves the pointer to the next line internally, so that we invoke the method again the next line of the file is read returning the entire comma separated row as a String.

```
Br.readLine()
```

We skip the header row/ Capture the header separately to use it while printing the output CSV.

We can have a dedicated method to recursively read the CSV file and add the student details at the end of the Linked List.

3.2. Read method

This method recursively calling readLine() method to read the Reader object until the end of file is reached and calls the linkedList.insert() method to insert a new node. Accepts the Reader object and the last node of the Linked List (Initially, Reader and head node) It return status code 0 as a completion reading the file to EOF.

The readLine() method return a comma separated String which is first split into a string array using regular exp “,” before befing converted into student class object. We again convert some String into int,char and Boolean to match with the student class data types.

Once we have the student, we call the LinkedList.insert() list to insert in the linked list, which return the last node of the Linked list.

We recursively call the read() method with the Reader object (read pointer is managed by the object itself) and the updated last nod of the Linked List.

```
Int read(Reader,lastNode)
        String row = reader.readline()
        If row == null      //If EOF
                Return 0;
        String[] line = row.split(“,”)
        Student student = convertStringToStudent(line)
        Node node = LinkedList.insert(lastNode,student)
        Return read(Reader,node)
```

Pseudo-code 9- Read from CSV

3.3. File Watcher

A separate thread that is responsible to notify if there is any modifications in the given CSV file. This monitors the file continuously until there is a keyboard interrupts to stop the watching process. When there is any change in the file it triggers a utility method to detect what and where the change happened in the CSV file and make changes to the Linked list accordingly. Accepts and return void.

3.4. Reload Method

This utility method recursively checks the File using the file Reader object for any changes, the check happens by comparing the file lines with the Node of the Linked list, The check can be of three type,

3.4.1. Both file and the Linked list has not reached the end

If there is any mismatch is the row and the corresponding Node of the linked list, then there is a flag of something is changed, The key idea to find if something is inserted or deleted is if something need to be deleted from the linked list then the next node of the linked list will be equal to the current row of the linked list or null else it is insertion operation.

```
    If current.next is not null and reader is not null
```

```
If currentRow is not currentNode.student
        If curentRow is equal to nextNode.student
                Delete current note
        Else
                Insert current row in the Linked list
```

3.4.2 File has reached the end and not the Linked List

If there are extra node in the Linked list and not the file then those node needs to be deleted from the linked list

```
If current.next is not null and reader is null
    Delete current node
```

Pseudo-code 11- Update the end of linked list when EOF

3.4.3 Linked list has reached the end and not the File

If there are extra rows in the File and not the linked list then those rows needs to be added at the end of the linked list.

```
If current.next is null and reader is not null
    Insert current row in the Linked list
```

Pseudo-code 12- Update at end of linked list when not EOF

3.4.4 Both file and Linked list has reached the end

Both the linked list and the File is in Sync and we can safely return back to the calling function

3.5. Prepare for printing

Open the Output CSV file in writer mode using the in build writer object.

```
PrintWriter printWriter = new PrintWriter(new FileWriter(<Output.csv>);
```

We can now use the `println()` method which is part of printWriter to write our content in the newline.

Also write the header line which was read and stored separately during the read step

```
printWriter.prinln(header);
```

Now we can call the dedicated method that traverses the Linked List and print the student details in the CSV recursively.

Note print() method writes in the same line and println() write in a new line, and the printer writer class keep track of the last line written and points to the next available line.

3.6. Print Linked List

This method is used to traverse and print the linked list node by node recursively in the output CSV file. This method accepts the Writer object and the current Node (Initially passing the head node). And returns the completion status when reached the last node of the Linked list.

Get the student object from the current node and convert it to comma separated String. Also use the println() method of the Writer object to write the comma separated string in the CSV file. Finally check if this is the last node of the Lisnked List if yes return the completion status else recursively call the print() method with the updated currentNode.next and the writer object.

```
print(writer,currentNode)
        Row = currentNode.student.toString()
        Writer.println(row)
        If currentNode.next == Null
                Return 0
        Return print(writer,currentNode.next)
```

Pseudo-code 13- Output in CSV

3.7. Prepare to execute

Get the input CSV file and the output file location from the user as input parameters

1.Instantiate the memory Database class by passing the input filename, output file name and the Linked list class object as constructor parameters

2. Open the input file using the Reader object and the output file using Writer object.

3. Read the header line before calling the read() method to recursively read the rest of file

4. Create a new thread watcherService() and load it with the watcher method

5. Start the watcher thread, which pause the current/main thread

6. Wait until the watcher thread completes and joins with the main thread

7. Print the header into the output CSV file before calling the print() to recursively print all the node of linked list to the output.csv recursively.

Step 2 to Step 7 might throw different exception including file not found, so enclose all in a try block and handle all the exception collectively using the catch block.

4. Executing the Program

Make sure all the source program and the required data have been downloaded an in place. The folder structure is given as follows,

Downloads

|

AlgoIndHw2-main

|

IndHw2

|

---- Java

|

----LinkedList.java

----Student.java

----MemoryDatabase.java

-----Python

|

----LinkedList.java

----Student.java

----MemoryDatabase.java

-----Data

|

----student-data.csv

Also make sure to change the student-data.csv file after starting the executions of the program, to verify the adoption of Linked List with any changes respect to the CSV file, you can **refer to one of the sample test cases below**.

1. Using Java

To execute,

Open terminal where we have the python programs **Downloads>AlgoIndHw2-main>IndHw2>Java**

"***javac *.java***" – this compiles the all the java files in the current directory to the .class file

"***java MemoryDatabase***" – this executes the MemoryDatabase.class file which has the main method and that is used for executing a java program, The rest two classes are dependents so those are executed when the program initializes them internally.

Note the class file name is the same as the java class name given in "*MemoryDatabase*.java". We can also run the java file in one command "*java MemoryDatabase.java*" but it is not recommended for large scale applications. Also it is worthy to mention running the java MemoryDatabase only triggers the **main** and the Memory database class and it methods needs to be initialized inside the main method (Static and not static methods)

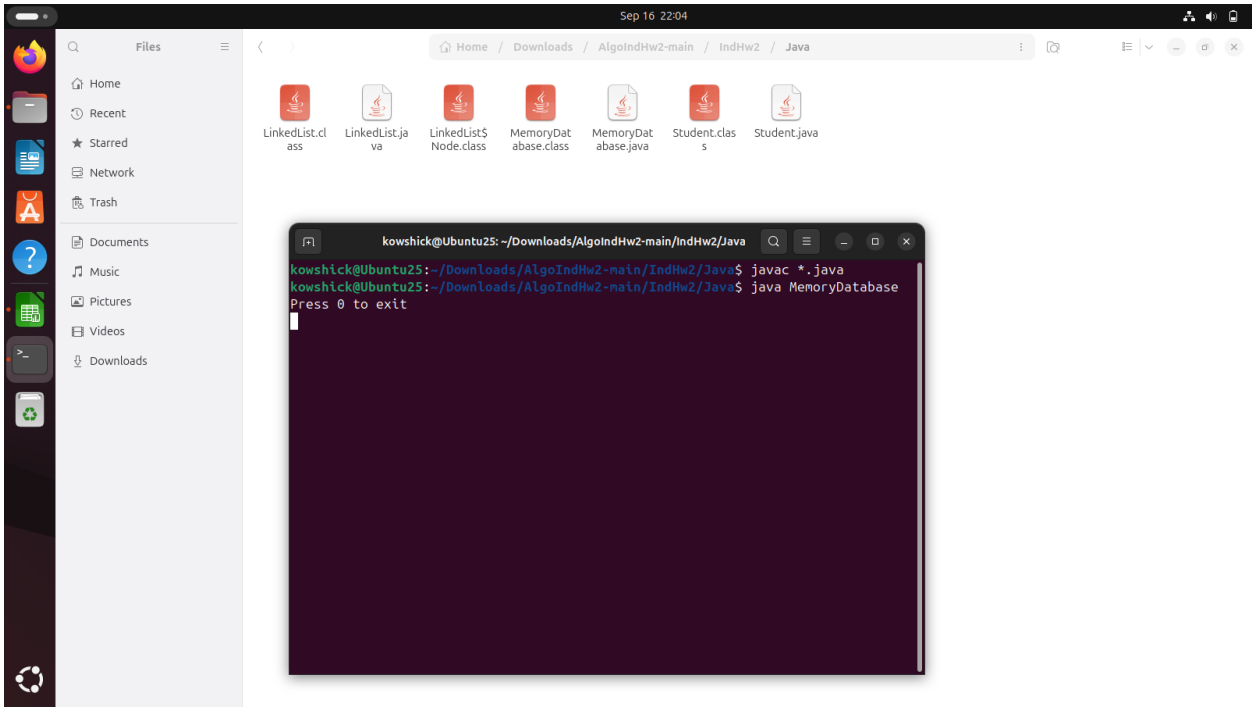


Figure 2- Compile and run the Java program

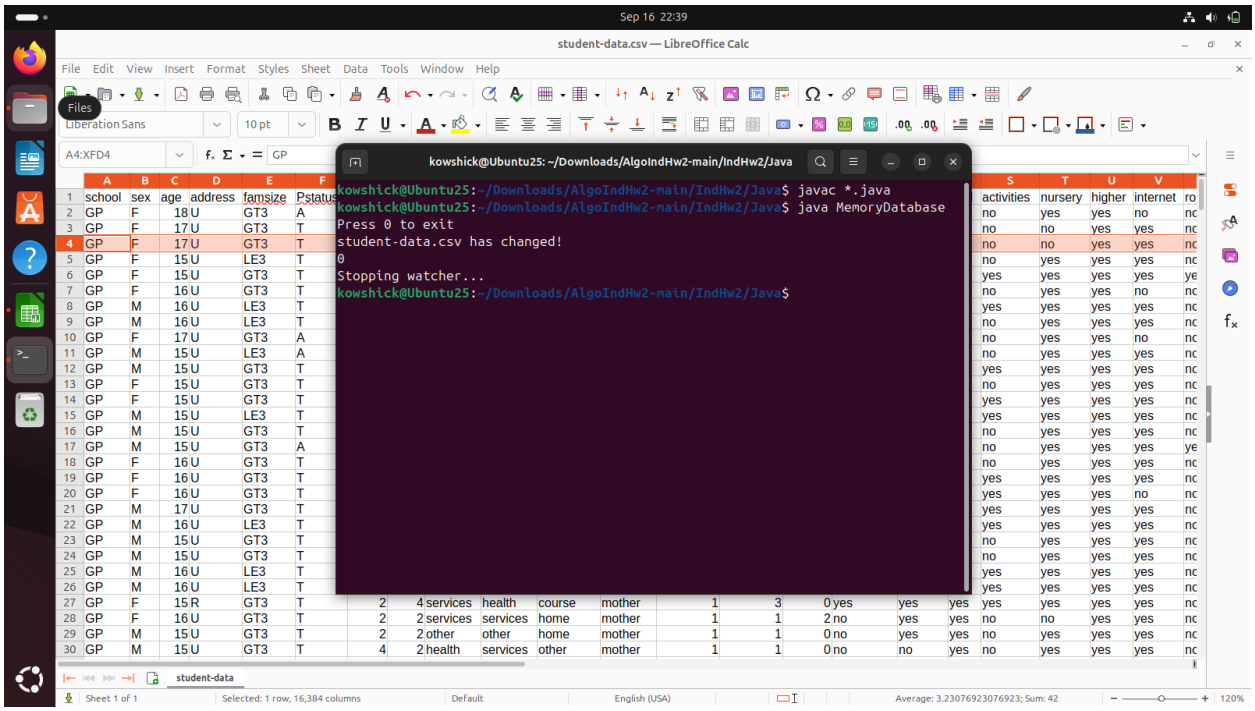


Figure 3- Do changes to CSV when program is running

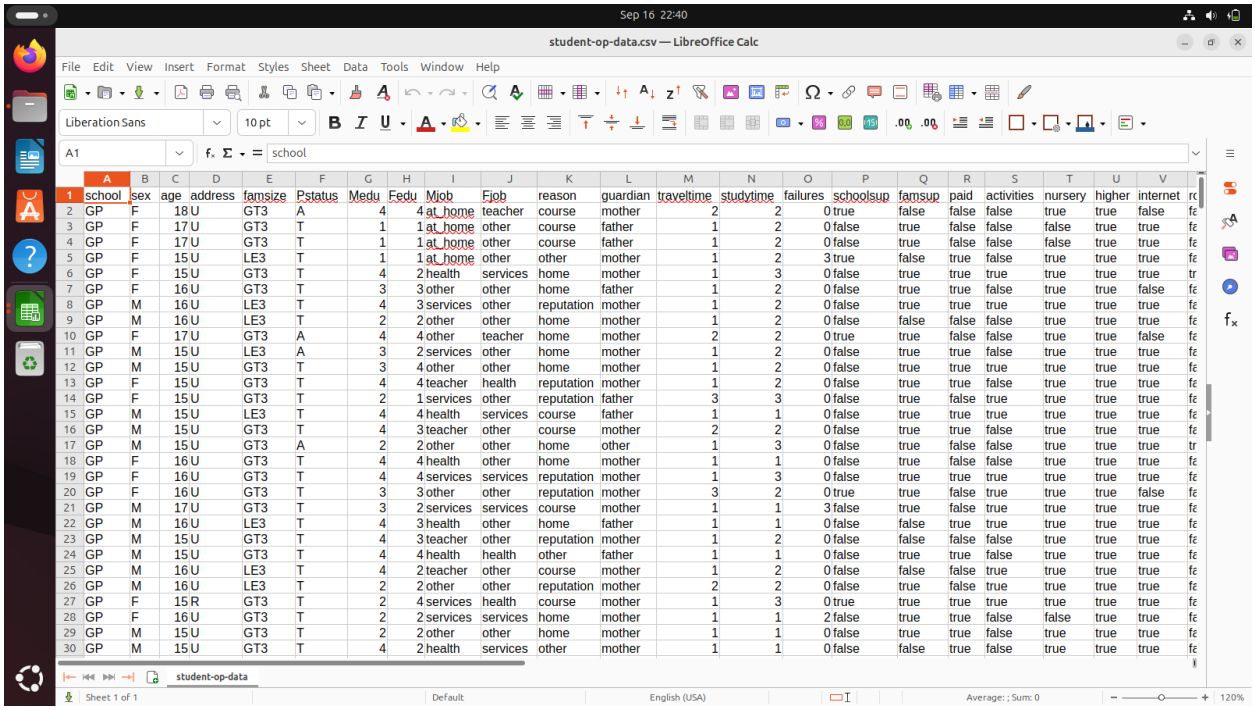


Figure 4- Updated output CSV

2. Using Python

To execute

Open terminal where we have the python programs **Downloads>AlgoIndHw2-main>IndHw2>Python**

"***python3 MemoryDatabase.py***" – command interprets and executes the code.

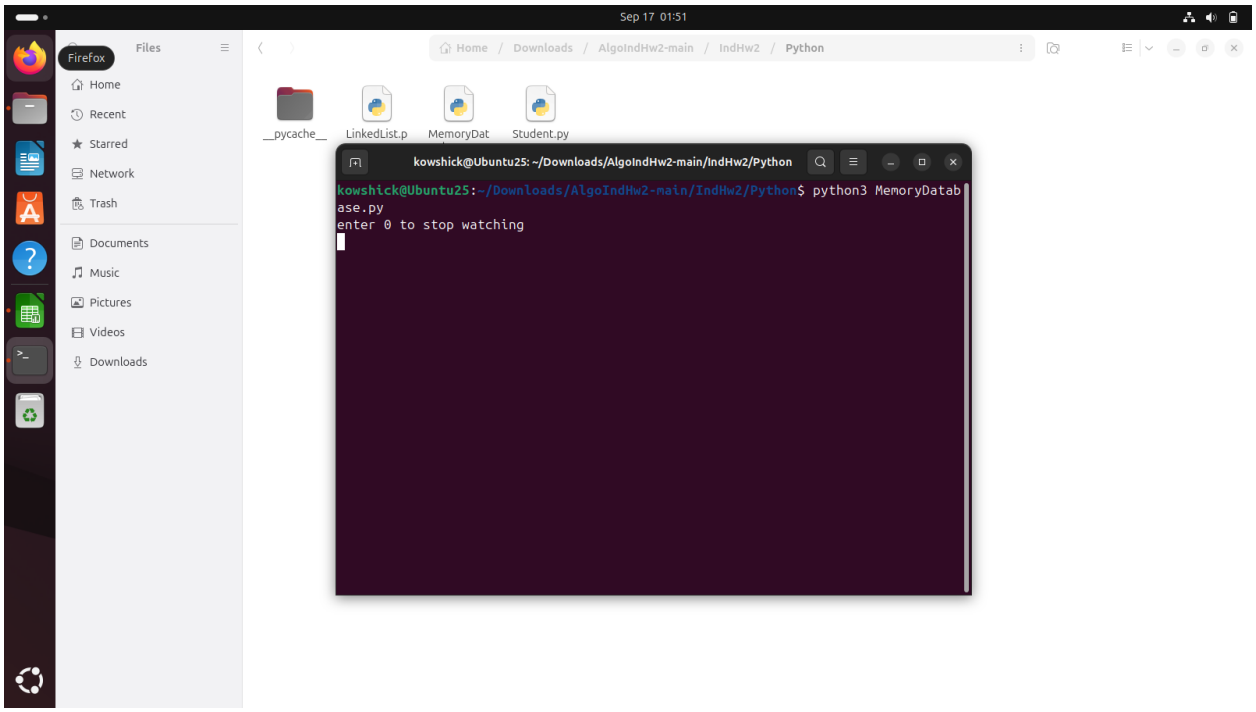


Figure 5- Run the python program

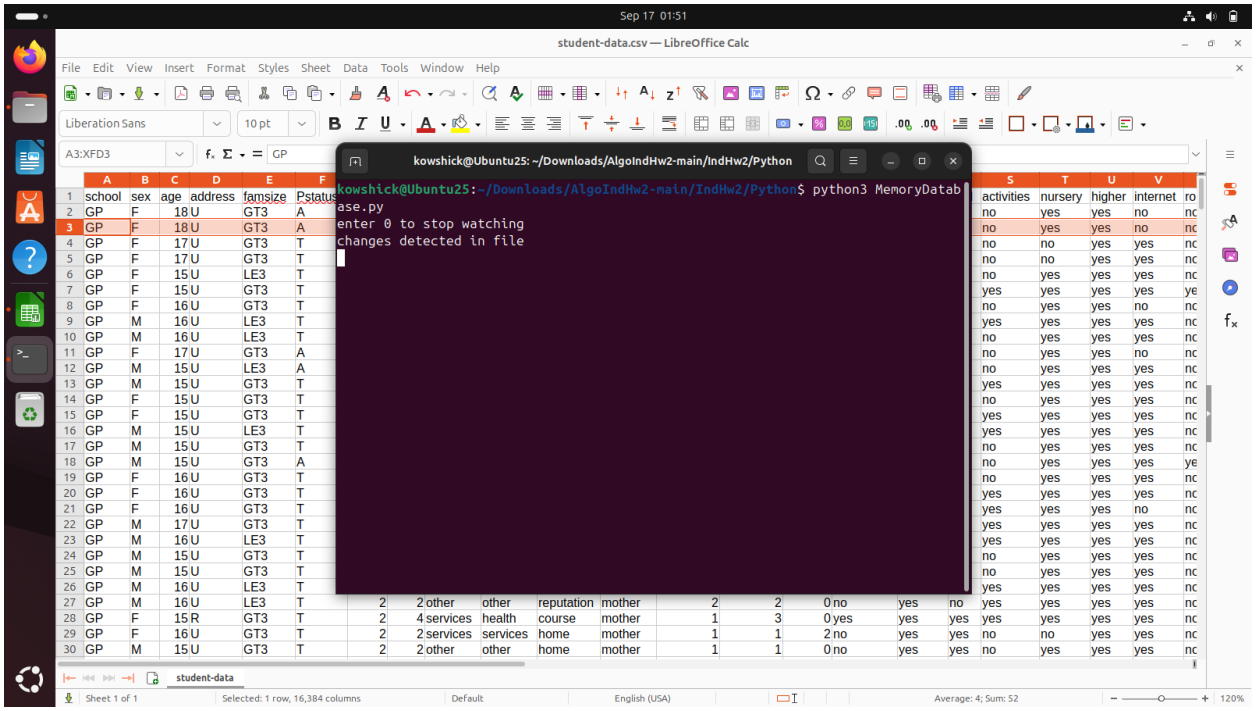


Figure 6- Do changes in CSV and detected by program

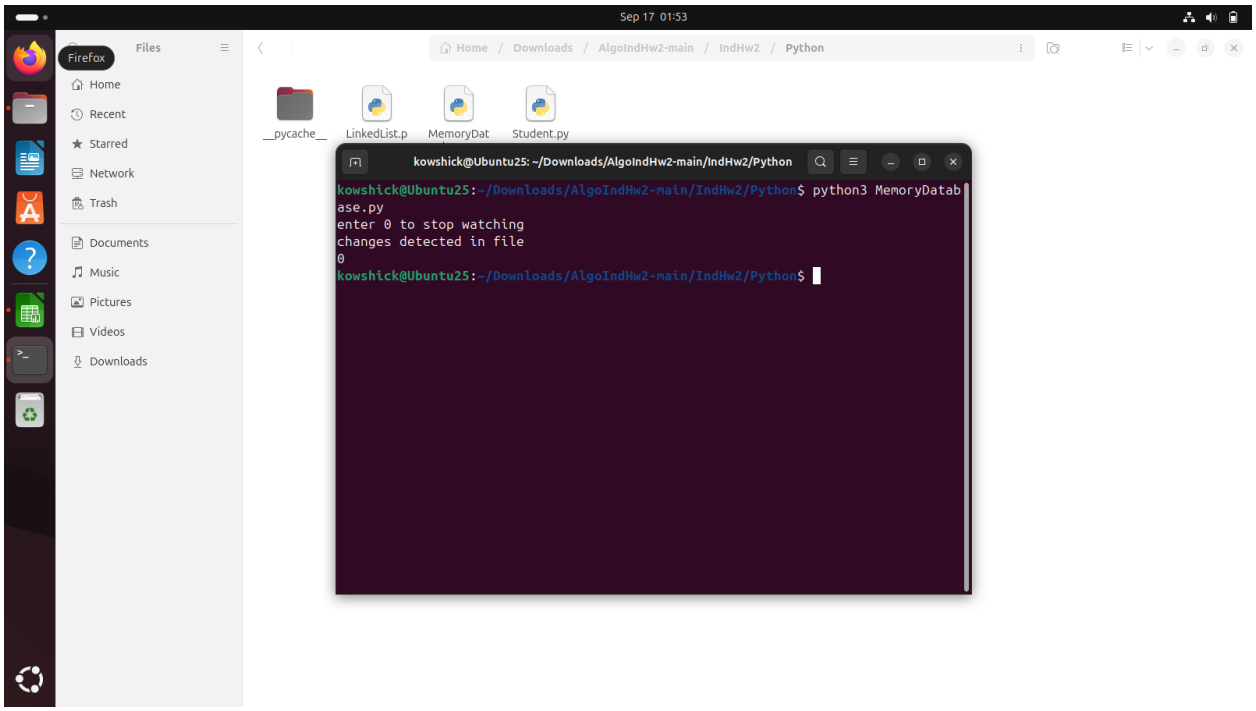
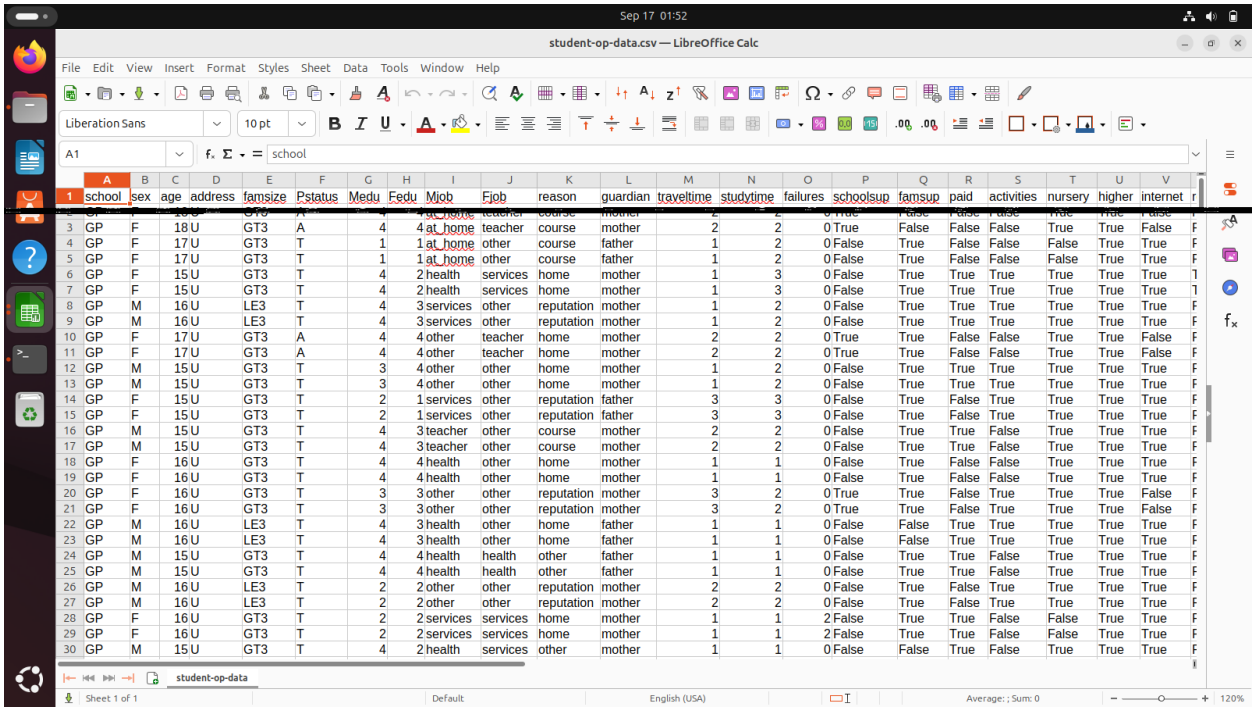


Figure 7- Stop the watcher



school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup	famsup	paid	activities	nursery	higher	internet
GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	True	False	False	False	True	True	False
GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	False	True	False	False	False	True	True
GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	False	True	False	False	False	True	True
GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	False	True	True	True	True	True	True
GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	False	True	True	True	True	True	True
GP	M	16	U	LE3	T	4	3	services	other	reputation	mother	1	2	0	False	True	True	True	True	True	True
GP	M	16	U	LE3	T	4	3	services	other	reputation	mother	1	2	0	False	True	True	True	True	True	True
GP	F	17	U	GT3	A	4	4	other	teacher	home	mother	2	2	0	True	True	False	False	True	True	False
GP	F	17	U	GT3	A	4	4	other	teacher	home	mother	2	2	0	True	True	False	False	True	True	False
GP	M	15	U	GT3	T	3	4	other	other	home	mother	1	2	0	False	True	True	True	True	True	True
GP	M	15	U	GT3	T	3	4	other	other	home	mother	1	2	0	False	True	True	True	True	True	True
GP	F	15	U	GT3	T	2	1	services	other	reputation	father	3	3	0	False	True	False	True	True	True	True
GP	F	15	U	GT3	T	2	1	services	other	reputation	father	3	3	0	False	True	False	True	True	True	True
GP	M	15	U	GT3	T	4	3	teacher	other	course	mother	2	2	0	False	True	True	False	True	True	True
GP	M	15	U	GT3	T	4	3	teacher	other	course	mother	2	2	0	False	True	True	False	True	True	True
GP	F	16	U	GT3	T	4	4	health	other	home	mother	1	1	0	False	True	False	False	True	True	True
GP	F	16	U	GT3	T	4	4	health	other	home	mother	1	1	0	False	True	False	False	True	True	True
GP	F	16	U	GT3	T	3	3	other	other	reputation	mother	3	2	0	True	True	False	True	True	True	False
GP	F	16	U	GT3	T	3	3	other	other	reputation	mother	3	2	0	True	True	False	True	True	True	False
GP	M	16	U	LE3	T	4	3	health	other	home	father	1	1	0	False	False	True	True	True	True	True
GP	M	16	U	LE3	T	4	3	health	other	home	father	1	1	0	False	False	True	True	True	True	True
GP	M	15	U	GT3	T	4	4	health	health	other	father	1	1	0	False	True	True	False	True	True	True
GP	M	15	U	GT3	T	4	4	health	health	other	father	1	1	0	False	True	True	False	True	True	True
GP	M	16	U	LE3	T	2	2	other	other	reputation	mother	2	2	0	False	True	False	True	True	True	True
GP	M	16	U	LE3	T	2	2	other	other	reputation	mother	2	2	0	False	True	True	True	True	True	True
GP	F	16	U	GT3	T	2	2	services	services	home	mother	1	1	2	False	True	True	False	False	True	True
GP	F	16	U	GT3	T	2	2	services	services	home	mother	1	1	2	False	True	True	False	False	True	True
GP	M	15	U	GT3	T	4	2	health	services	other	mother	1	1	0	False	False	True	False	True	True	True

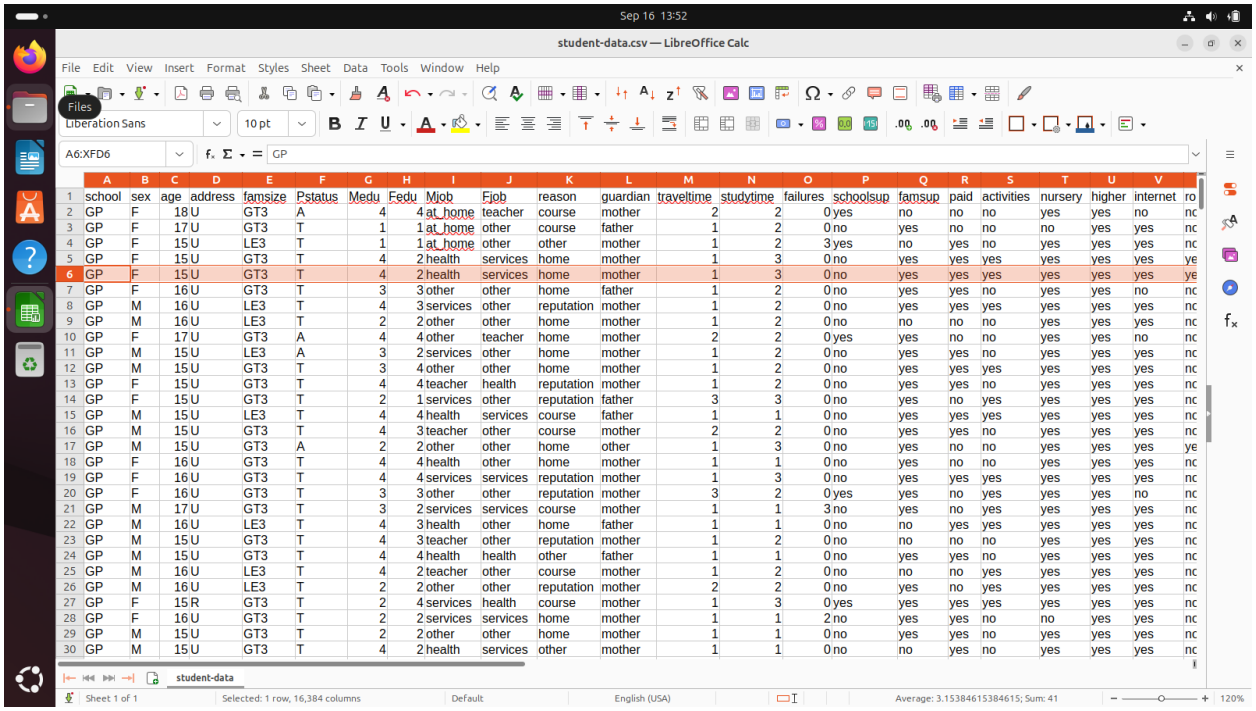
Figure 8- Output reflecting the changes

5. Sample Test Cases

5.1. Base cases

5.1.1 Insert at middle

Insert a student record somewhere in the middle of the existing CSV as shown



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
1	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup	famsup	paid	activities	nursery	higher	internet	ro
2	GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	yes	no	no	no	yes	yes	no	nc
3	GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	no	yes	no	no	no	yes	yes	nc
4	GP	F	15	U	LE3	T	1	1	at_home	other	other	mother	1	2	3	yes	no	yes	no	yes	yes	yes	nc
5	GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	no	yes	yes	yes	yes	yes	yes	ye
6	GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	no	yes	yes	yes	yes	yes	yes	ye
7	GP	F	16	U	GT3	T	3	3	other	other	home	father	1	2	0	no	yes	yes	no	yes	yes	no	nc
8	GP	M	16	U	LE3	T	4	3	services	other	reputation	mother	1	2	0	no	yes	yes	yes	yes	yes	yes	nc
9	GP	M	16	U	LE3	T	2	2	other	other	home	mother	1	2	0	no	no	no	no	yes	yes	yes	nc
10	GP	F	17	U	GT3	A	4	4	other	teacher	home	mother	2	2	0	yes	yes	no	no	yes	yes	no	nc
11	GP	M	15	U	LE3	A	3	2	services	other	home	mother	1	2	0	no	yes	yes	no	yes	yes	yes	nc
12	GP	M	15	U	GT3	T	3	4	other	other	home	mother	1	2	0	no	yes	yes	yes	yes	yes	yes	nc
13	GP	F	15	U	GT3	T	4	4	teacher	health	reputation	mother	1	2	0	no	yes	yes	no	yes	yes	yes	nc
14	GP	F	15	U	GT3	T	2	1	services	other	reputation	father	3	3	0	no	yes	no	yes	yes	yes	yes	nc
15	GP	M	15	U	LE3	T	4	4	health	services	course	father	1	1	0	no	yes	yes	yes	yes	yes	yes	nc
16	GP	M	15	U	GT3	T	4	3	teacher	other	course	mother	2	2	0	no	yes	yes	no	yes	yes	yes	nc
17	GP	M	15	U	GT3	A	2	2	other	other	home	other	1	3	0	no	yes	no	no	yes	yes	yes	ye
18	GP	F	16	U	GT3	T	4	4	health	other	home	mother	1	1	0	no	yes	no	no	yes	yes	yes	nc
19	GP	F	16	U	GT3	T	4	4	services	services	reputation	mother	1	3	0	no	yes	yes	yes	yes	yes	yes	nc
20	GP	F	16	U	GT3	T	3	3	other	other	reputation	mother	3	2	0	yes	yes	no	yes	yes	yes	no	nc
21	GP	M	17	U	GT3	T	3	2	services	services	course	mother	1	1	3	no	yes	no	yes	yes	yes	yes	nc
22	GP	M	16	U	LE3	T	4	3	health	other	home	father	1	1	0	no	no	yes	yes	yes	yes	yes	nc
23	GP	M	15	U	GT3	T	4	3	teacher	other	reputation	mother	1	2	0	no	no	no	no	yes	yes	yes	nc
24	GP	M	15	U	GT3	T	4	4	health	health	other	father	1	1	0	no	yes	yes	no	yes	yes	yes	nc
25	GP	M	16	U	LE3	T	4	2	teacher	other	course	mother	1	2	0	no	no	no	yes	yes	yes	yes	nc
26	GP	M	16	U	LE3	T	2	2	other	other	reputation	mother	2	2	0	no	yes	no	yes	yes	yes	yes	nc
27	GP	F	15	R	GT3	T	2	4	services	health	course	mother	1	3	0	yes	yes	yes	yes	yes	yes	yes	nc
28	GP	F	16	U	GT3	T	2	2	services	services	home	mother	1	1	2	no	yes	yes	no	no	yes	yes	nc
29	GP	M	15	U	GT3	T	2	2	other	other	home	mother	1	1	0	no	yes	yes	no	yes	yes	yes	nc
30	GP	M	15	U	GT3	T	4	2	health	services	other	mother	1	1	0	no	no	yes	no	yes	yes	yes	nc

Figure 9- Insert at middle

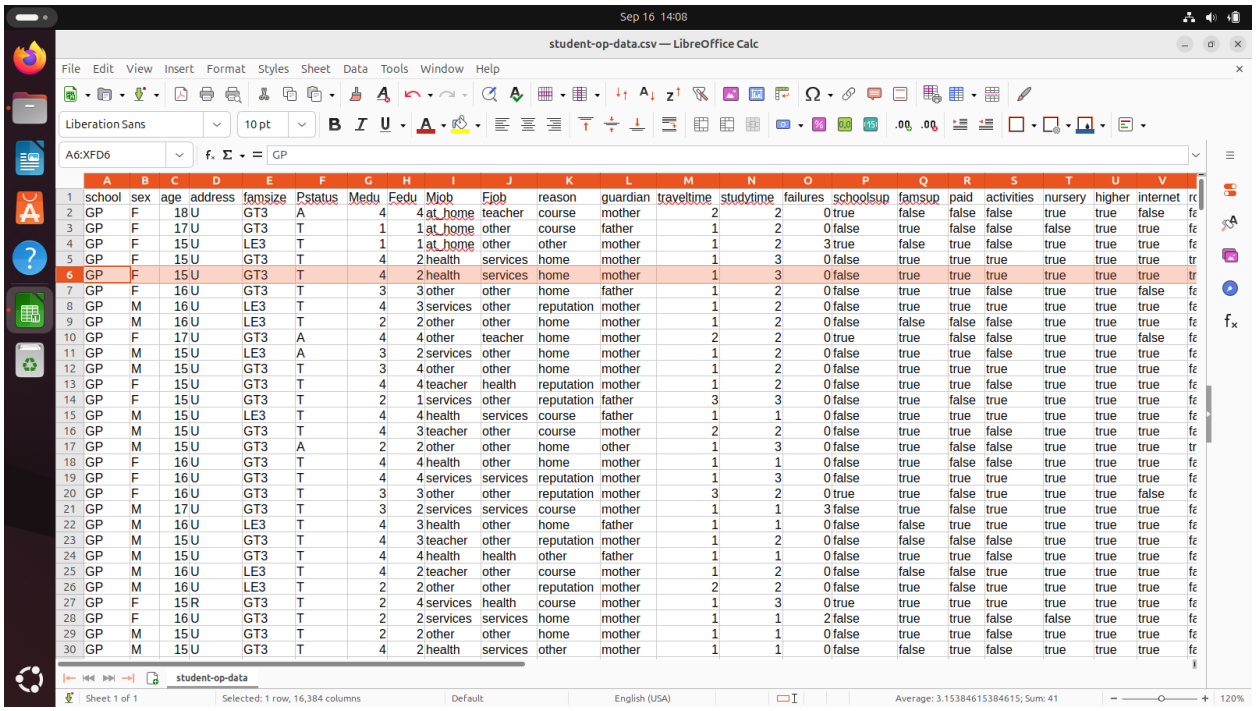


Figure 10- Output reflecting insertion in middle

5.1.2 Delete from middle

Delete a student record somewhere from the middle of the existing CSV as shown

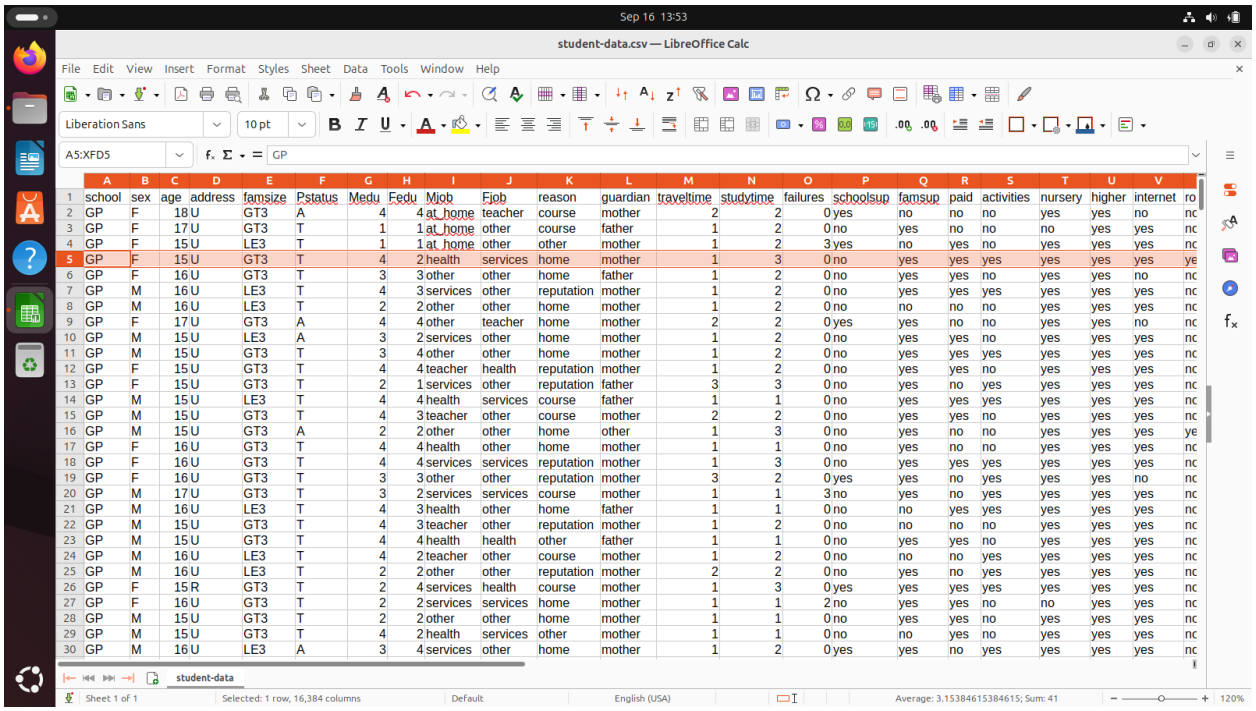


Figure 11- Delete from middle

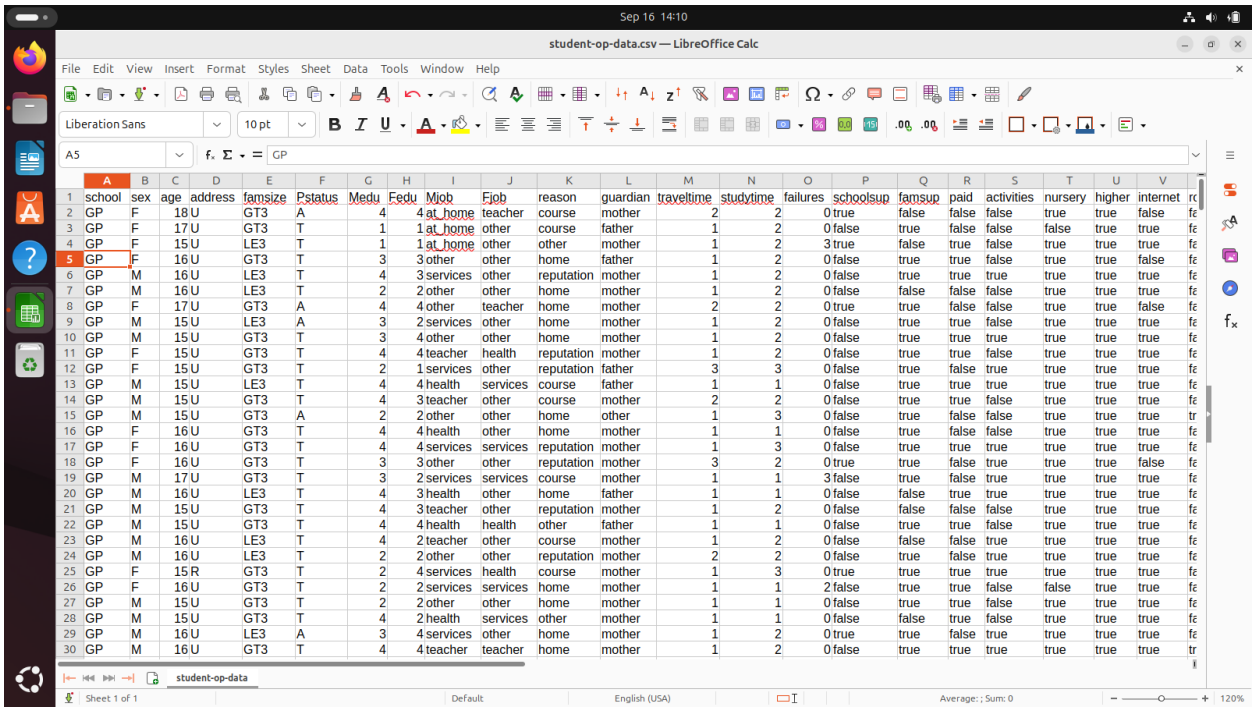


Figure 12-Output reflection deletion from middle

5.1.3 Insert at end

Insert a student record at the end of the existing CSV as shown

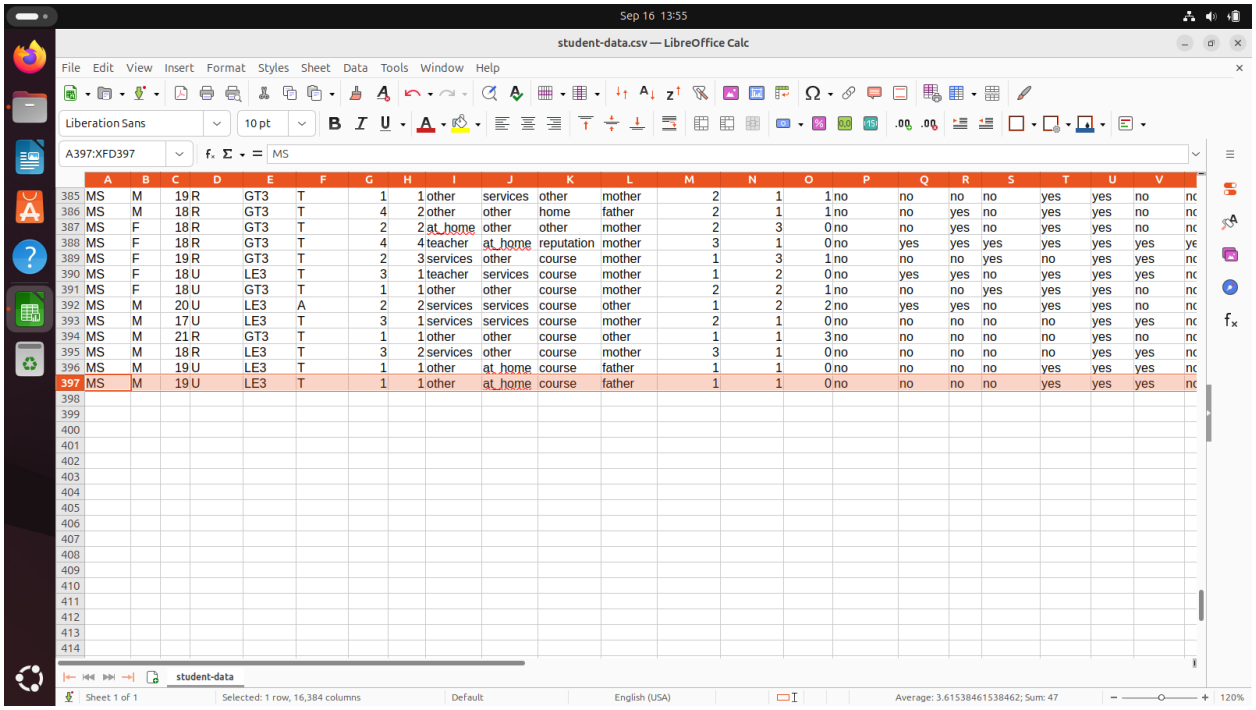


Figure 13- Insertion at the end

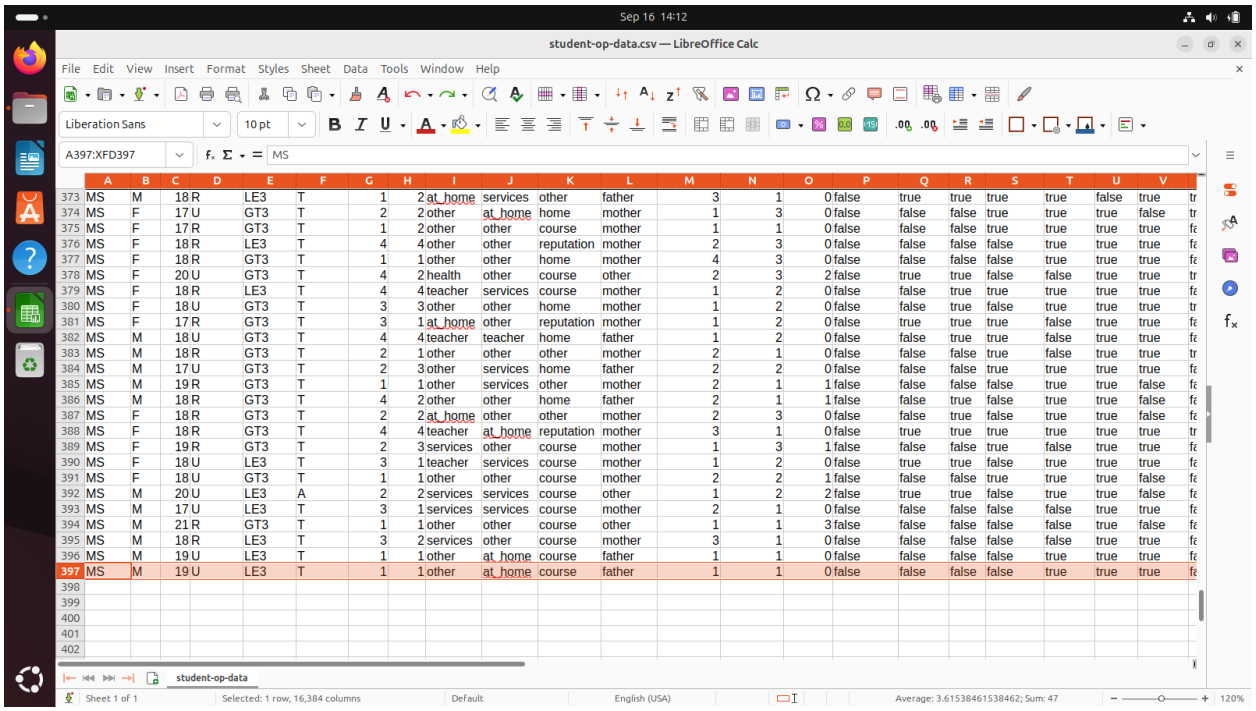


Figure 14- Output reflection insertion at the end

5.1.4 Delete at top

Delete a student record at the top of the existing CSV as shown

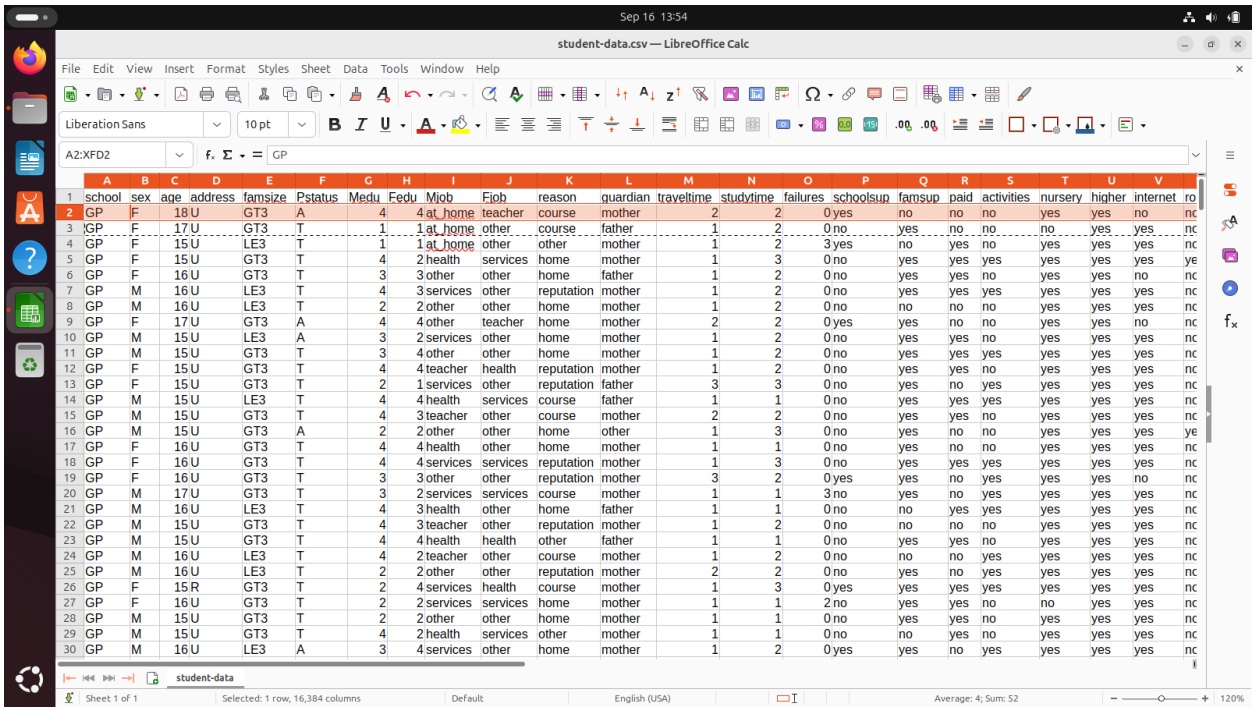


Figure 15- Delete the head

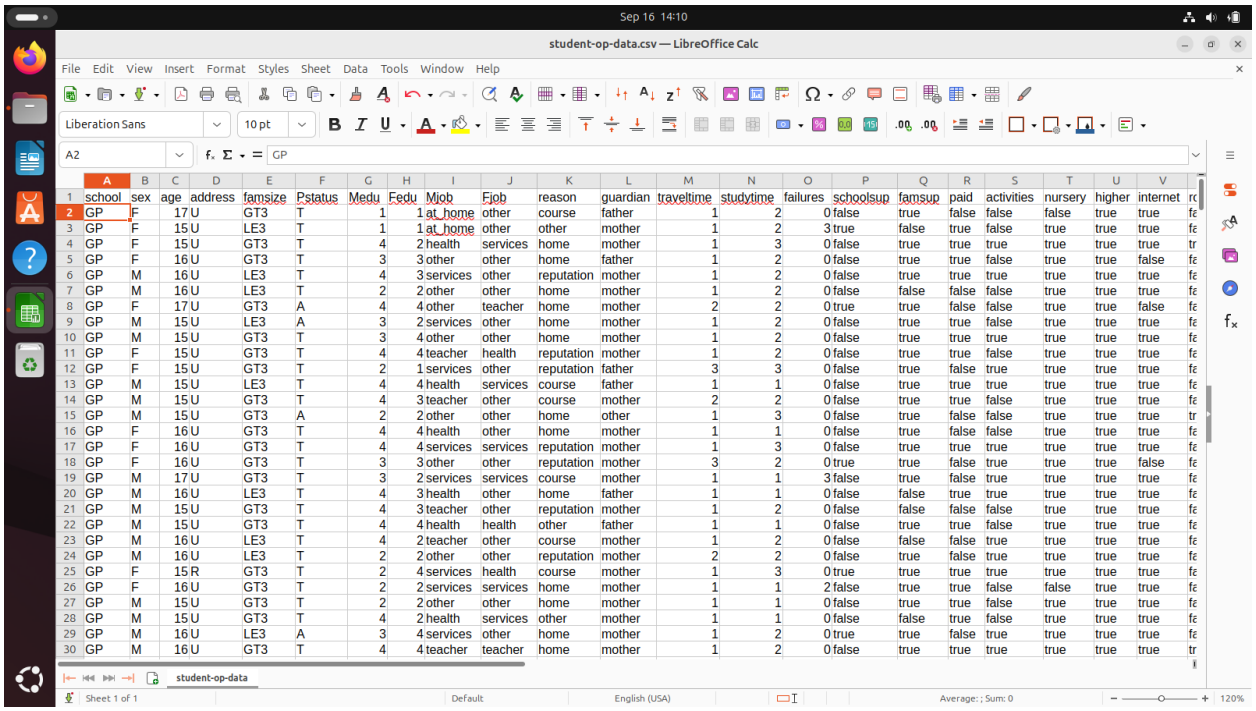


Figure 16- Output reflecting deletion of head

5.2 Edge cases

5.2.1 Insert at top

Insert a student record at the top of the existing CSV as shown

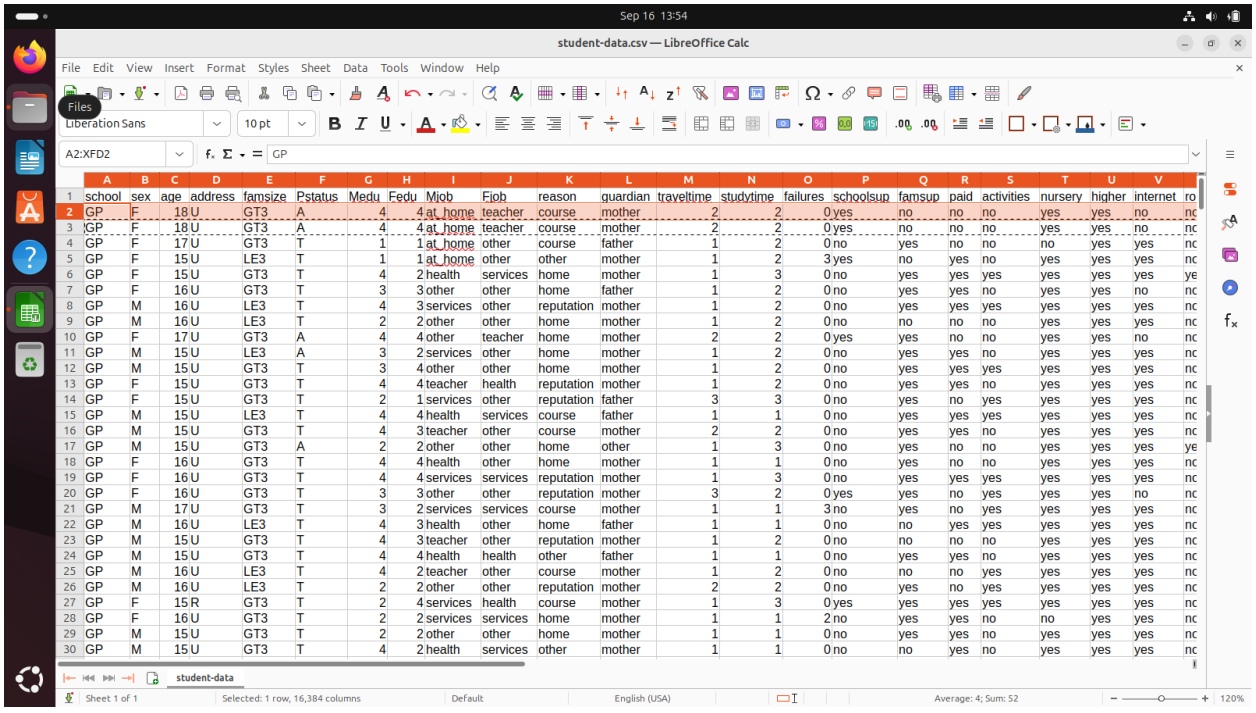


Figure 17- Insert a new head

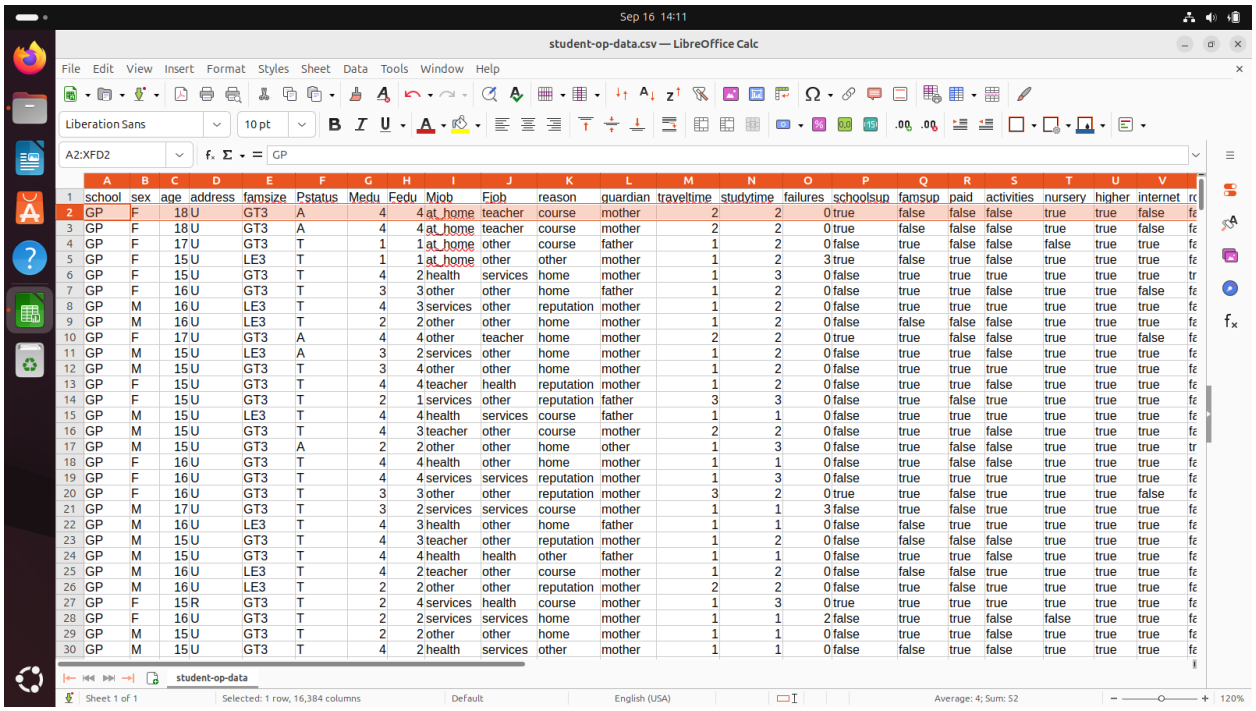


Figure 18- Output reflecting insertion of new head

5.2.2 Delete at end

Delete a student record at the bottom of the existing CSV as shown

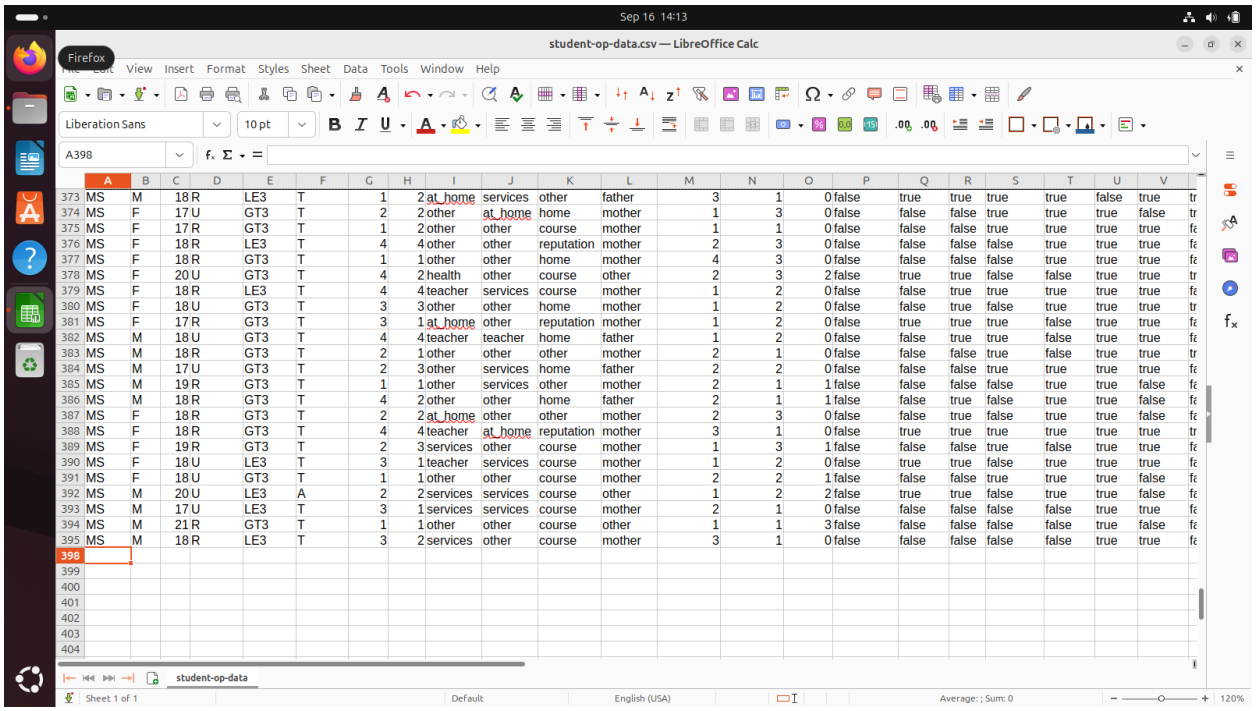


Figure 19- Delete the last node

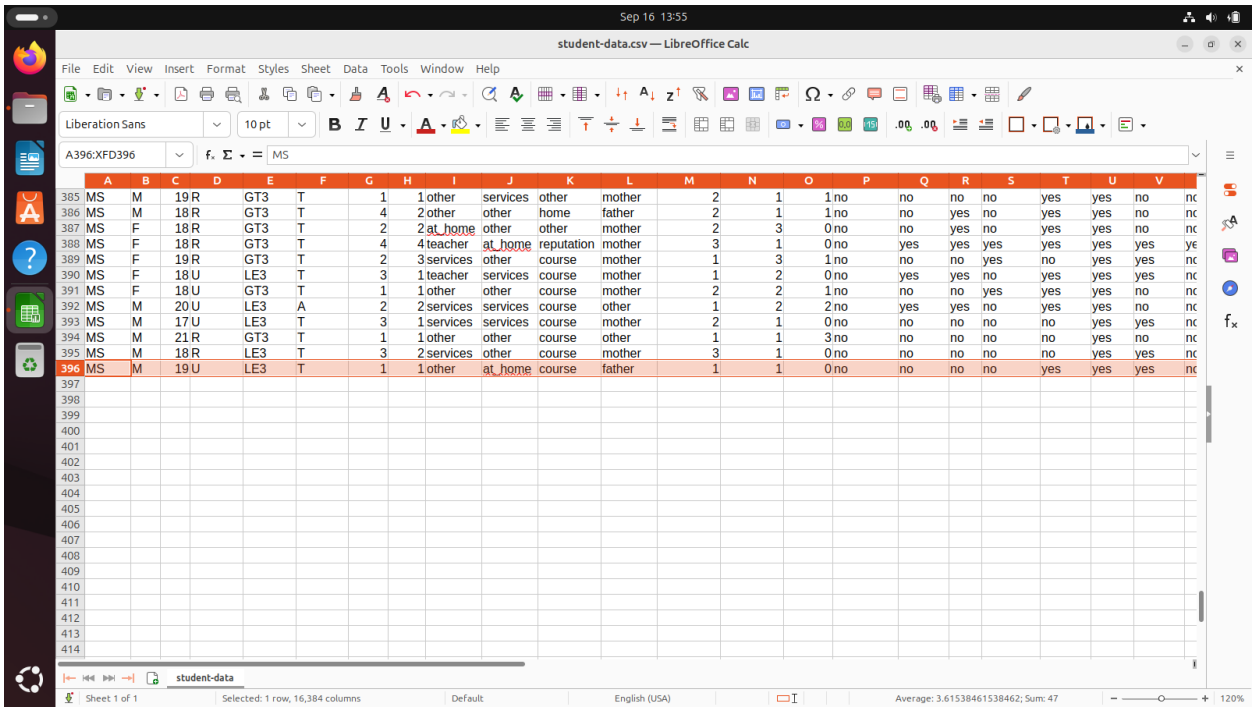


Figure 20- Output reflecting deletion of last node

5.2.3 No Insertion or deletion

It is not specified is the requirement document that there will always be insertions or deletion

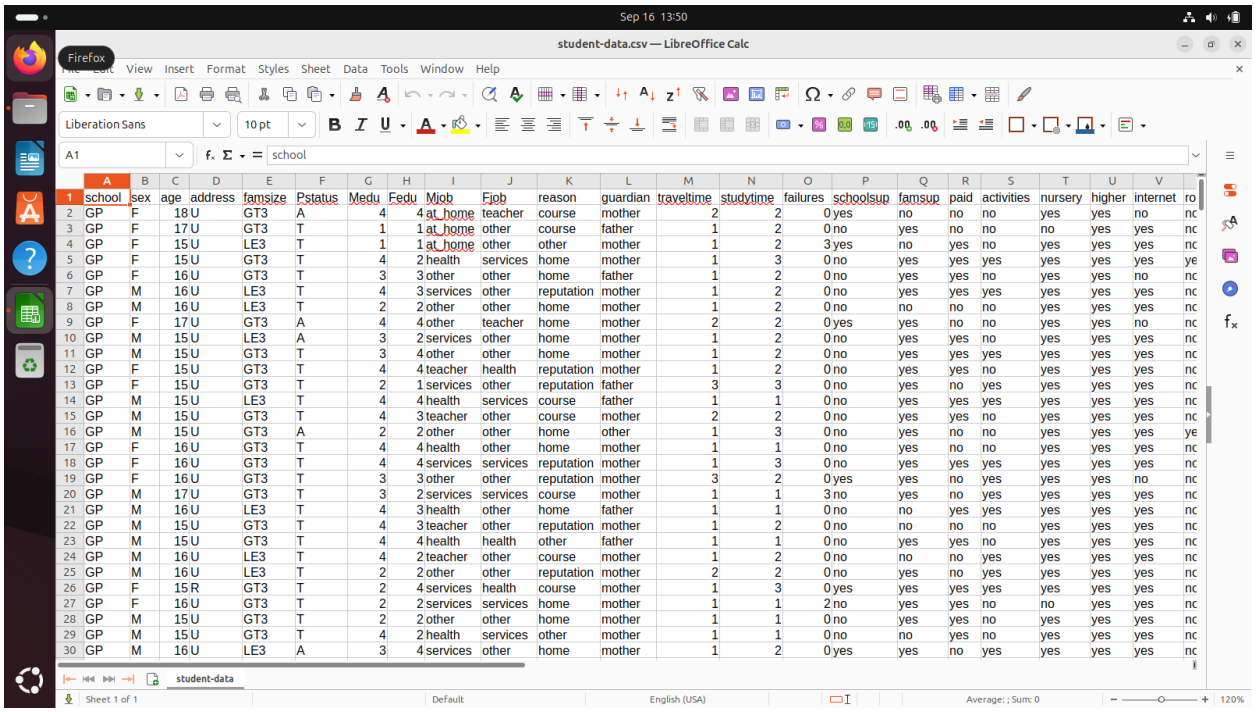
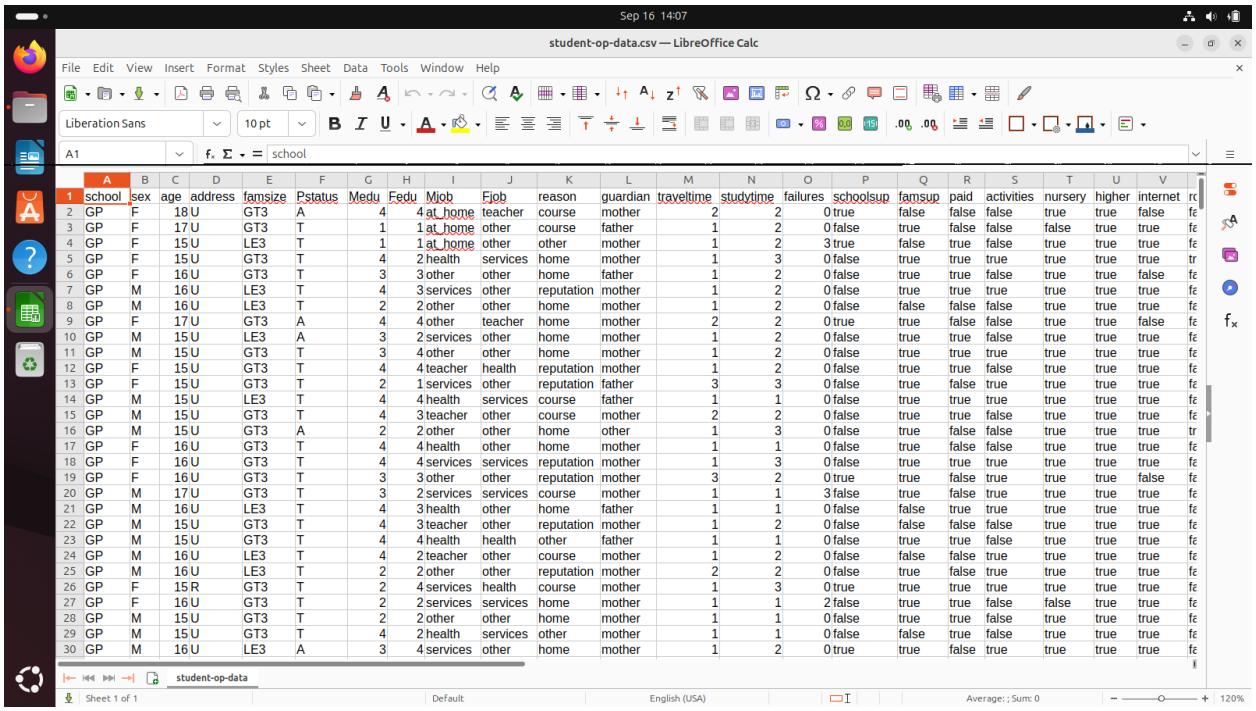


Figure 21- Unchanged input



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup	famsup	paid	activities	nursery	higher	internet
2	GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	true	false	false	false	true	true	false
3	GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	false	true	false	false	false	true	true
4	GP	F	15	U	LE3	T	1	1	at_home	other	other	mother	1	2	3	true	false	true	false	true	true	true
5	GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	false	true	true	true	true	true	true
6	GP	F	16	U	GT3	T	3	3	other	other	home	father	1	2	0	false	true	true	false	true	true	false
7	GP	M	16	U	LE3	T	4	3	services	other	reputation	mother	1	2	0	false	true	true	true	true	true	true
8	GP	M	16	U	LE3	T	2	2	other	other	home	mother	1	2	0	false	false	false	false	true	true	true
9	GP	F	17	U	GT3	A	4	4	other	teacher	home	mother	2	2	0	true	true	false	false	true	true	false
10	GP	M	15	U	LE3	A	3	2	services	other	home	mother	1	2	0	false	true	true	false	true	true	true
11	GP	M	15	U	GT3	T	3	4	other	other	home	mother	1	2	0	false	true	true	true	true	true	true
12	GP	F	15	U	GT3	T	4	4	teacher	health	reputation	mother	1	2	0	false	true	true	false	true	true	true
13	GP	F	15	U	GT3	T	2	1	services	other	reputation	father	3	3	0	false	true	false	true	true	true	true
14	GP	M	15	U	LE3	T	4	4	health	services	course	father	1	1	0	false	true	true	true	true	true	true
15	GP	M	15	U	GT3	T	4	3	teacher	other	course	mother	2	2	0	false	true	true	false	true	true	true
16	GP	M	15	U	GT3	A	2	2	other	other	home	other	1	3	0	false	true	false	false	true	true	true
17	GP	F	16	U	GT3	T	4	4	health	other	home	mother	1	1	0	false	true	false	false	true	true	true
18	GP	F	16	U	GT3	T	4	4	services	services	reputation	mother	1	3	0	false	true	true	true	true	true	true
19	GP	F	16	U	GT3	T	3	3	other	other	reputation	mother	3	2	0	true	true	false	true	true	true	false
20	GP	M	17	U	GT3	T	3	2	services	services	course	mother	1	1	3	false	true	false	true	true	true	true
21	GP	M	16	U	LE3	T	4	3	health	other	home	father	1	1	0	false	false	true	true	true	true	true
22	GP	M	15	U	GT3	T	4	3	teacher	other	reputation	mother	1	2	0	false	false	false	false	true	true	true
23	GP	M	15	U	GT3	T	4	4	health	health	other	father	1	1	0	false	true	false	true	true	true	true
24	GP	M	16	U	LE3	T	4	2	teacher	other	course	mother	1	2	0	false	false	false	true	true	true	true
25	GP	M	16	U	LE3	T	2	2	other	other	reputation	mother	2	2	0	false	true	false	true	true	true	true
26	GP	F	15	R	GT3	T	2	4	services	health	course	mother	1	3	0	true	true	true	true	true	true	true
27	GP	F	16	U	GT3	T	2	2	services	services	home	mother	1	1	2	false	true	true	false	false	true	true
28	GP	M	15	U	GT3	T	2	2	other	other	home	mother	1	1	0	false	true	true	false	true	true	true
29	GP	M	15	U	GT3	T	4	2	health	services	other	mother	1	1	0	false	false	true	false	true	true	true
30	GP	M	16	U	LE3	A	3	4	services	other	home	mother	1	2	0	true	true	false	true	true	true	true

Figure 22- Unchanged output

5. Shutting down the Virtual environment

It is important to power off the Ubuntu virtual environment, as improper powering off might lead to the crashing of the OS.

1. Click on the battery icon in top right corner > click the power symbol > Power off… > confirm

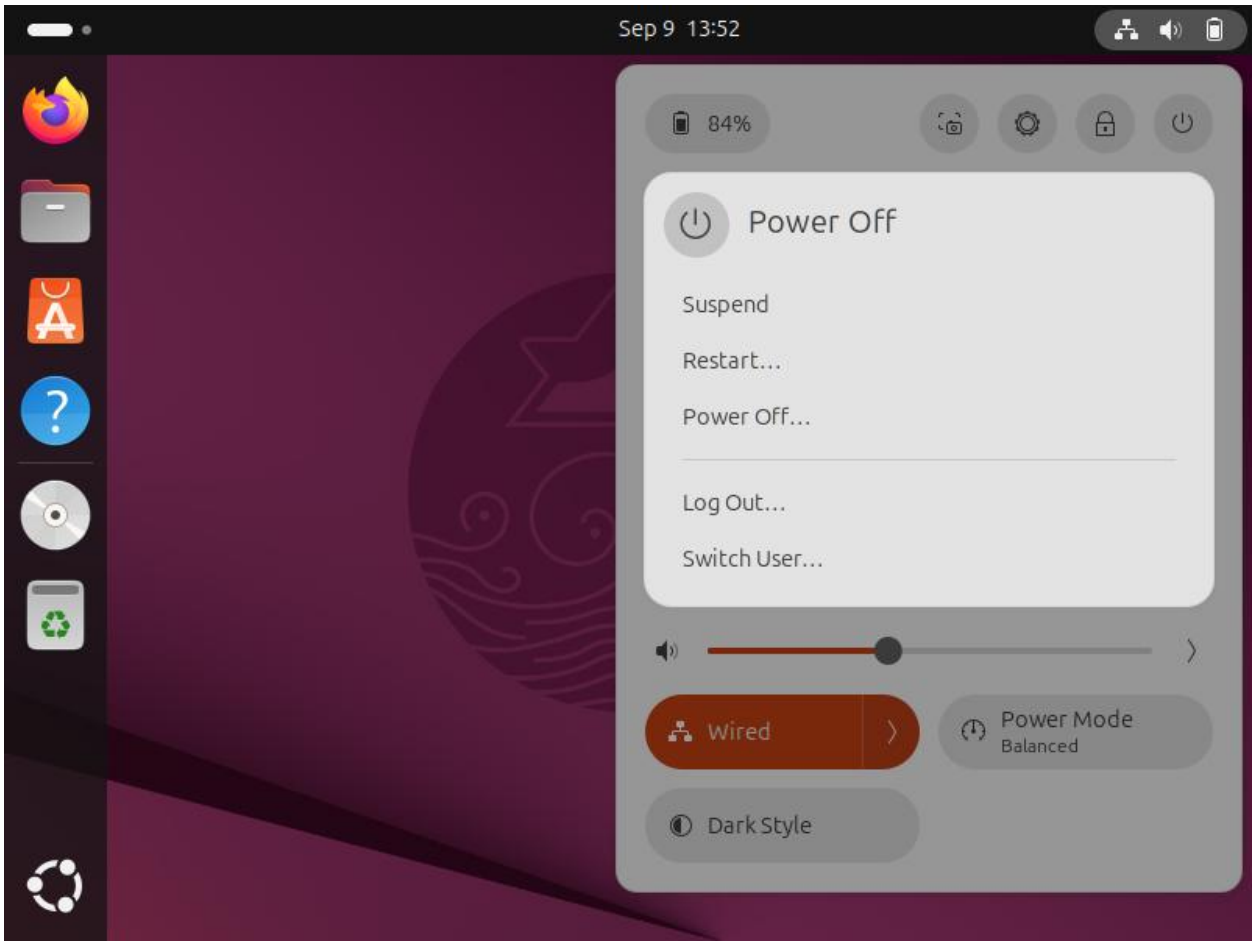


Figure 23- Shutting down Ubuntu

Key takeaways

Singly Linked List

It is a data structure; unlike arrays they offer **dynamic scaling**. They are made of interconnection blocks, the simple real-world parallel would be a **Train**, each block contain 2 parts one to have the **data** itself and the other to point to the location of the **next** block. By this way we can easily scale the number of block. Linked lists are the perfect choice when we need to allocate “n” data points, where **n is unknown**. They do have quite **a few tradeoffs** with array in terms of complexities both time and space.

Recursion

Recursion is a type of loops, written when we don’t know the exit condition before hand. A safe comparison would be an **infinite loop** with a **break**; condition in it. Also we have normal and **tail recursions**.

File IO

We have touched the basics of **reading** and **writing** in a CSV file. Also we have used **File Watcher** service which is good for noticing any action performed on the file like creation, deletion and modification, it returns if there is any event on the file which is per says the hint of **event driven programming paradigm**.

Multi Threads

It is a special case of programming where we use different program threads to accomplish a task in the same time, in the algorithm world it can be loosely equated to divide and conquer.

Conclusion

From the above execution screenshots is the confirmed to have build a small in-memory database using single linked list, that performs the actions,

1. Reads the contents of CSV file and add it to Linked List recursively
2. Adopts to any insertion/deletion in the CSV file
3. Writes the linked List to the output CSV file.

FAQ

1. *Why not store the contents of the CSV as a comma separated string in the Linked List?*

It is advised to use the Object Oriented approach, as it is easier to access and extend in the future

2. *Why not use String data type for all the class variables of the Student class?*

String is not a primitive data structure and consumes a lot of memory space, Always use primitive data types like int char and Boolean when possible

3. *Why dependency injection?*

Dependency is a good software design practice, where it is easier to plug-in different initialization values

4. *Is it ok to open both Reader and Writer object simultaneously, wont it cause synchronization problem*

There won't be any synchronizations issues due to opening two files concurrently, since we read and write into two different files, so there is no need to use any Mutex locks.

5. *Why multi-threads*

File watcher is a blocking service, even if we do an infinite loop until keyboard interrupt, the File watcher will still be blocking and won't return the control to the main threads.

6. What is the difference between == and .equals() method in Java?

'==' does shallow comparison i.e. return two object points to the same memory location, can be used for comparing two primitive data type items on the other hand we .equals() method perform deep comparison i.e. it compares all the member variable of both the classes to say wheatear they are same, used when comparing two class variables. Note by default .equals() method does shallow comparison until overwritten manually.

7. What is the difference between write and append

Write operation overwrites the existing line and append operation goes to the new line to write a new line

8. What is static in Java

Object defined by static keywords gets initialized when the program starts by the JVM itself. For example main() method is defined static so it will be triggered by the JVM. We use static variables to define filenames and a static inner class. Unnecessary use of static keyword is not advised.

Acknowledgement

I remember encountering a similar question in one of my interviews, confirming the relevance of the assignment to industry standards. I would like to acknowledge Professor Dr. **Wei "David" Dai**, for assigning this Assignment as it is a great practice for me. Also I would like to appreciate **PNW CES** and **CS** department for providing me the adequate opportunities and letting take this course.

Also I would like to thanks **Kaggle** dataset, which was a crucial part of this assignment also **draw.io** which was use to draw the UML diagram.

Bibliography

baeldung. (2024, January 2024). *java-nio2-watchservice*. Retrieved September 14, 2025, from baeldung: https://www.baeldung.com/java-nio2-watchservice

devna... (2025, July 23). *how-to-detect-file-changes-using-python*. Retrieved September 14, 2025, from geeksforgeeks: https://www.geeksforgeeks.org/python/how-to-detect-file-changes-using-python/

Kartik. (2025, Septmber 08). *geeksforgeeks*. Retrieved September 13, 2025, from multithreading-in-java: https://www.geeksforgeeks.org/java/multithreading-in-java/

Kartik. (2025, January 02). *geeksforgeeks*. Retrieved Septmber 14, 2025, from multithreading-python: https://www.geeksforgeeks.org/python/multithreading-python-set-1/

kellygakii. (n.d.). *student-data-csv*. Retrieved Septmber 14, 2025, from kaggle: https://www.kaggle.com/datasets/kellygakii/student-data-csv

oracle. (n.d.). *WatchService*. Retrieved September 16, 2025, from oracle: https://docs.oracle.com/javase/8/docs/api/java/nio/file/WatchService.html

programiz. (n.d.). *Linked List*. Retrieved Septemeber 12, 2025, from programwiz: https://www.programiz.com/dsa/linked-list

Programmer, T. (2024, August 13). *How to install Ubuntu 24.04 LTS in VirtualBox 2024*. Retrieved September 03, 2025, from Youtube: https://www.youtube.com/watch?v=Hva8lsV2nTk

Python. (n.d.). *threading*. Retrieved Septmber 16, 2025, from python: https://docs.python.org/3/library/threading.html

queue. (n.d.). Retrieved Septemeber 04, 2025, from programiz: https://www.programiz.com/dsa/queue

Stack. (n.d.). Retrieved September 04, 2025, from programiz: https://www.programiz.com/dsa/stack

Appendix

1. Using Java

1.1 Student Class

```java
/*
  @author: Kowshick Srinivasan
 * @version: 1.0
 * @Assignment: Hw2
 */

import java.util.Objects;

public class Student {
    String school;
    String mJob;
    String fJob;
    String reason;
```

```
String guardian;
String famSize;

//Can convert one letters to char instead of default string
//for efficient memory usage
char sex;
char address;
char pStatus;

//Can convert Integers to int instead of default string
//for efficient memory usage
int age;
int famRel;
int freeTime;
int goOut;
int dalc;
int walc;
int health;
int absence;
int travelTime;
int studyTime;
int failures;
int medu;
int fedu;


//Can convert yes/no to boolean instead of default string
//for efficient memory usage
boolean schoolsUp;
boolean farmsUp;
boolean paid;
boolean activities;
boolean nursery;
boolean higher;
boolean internet;
boolean romantic;
boolean passed;


//Constructor to initialize the class variables
public Student(String school, char sex, int age, char address,
               String famSize, char pStatus, int medu, int fedu,
               String mJob, String fJob, String reason,
               String guardian, int travelTime, int studyTime,
               int failures, boolean schoolsUp, boolean farmsUp,
               boolean paid, boolean activities, boolean nursery,
               boolean higher, boolean internet, boolean romantic,
               int famRel, int freeTime, int goOut, int dalc, int walc,
               int health, int absence, boolean passed) {
    this.school = school;
    this.sex = sex;
    this.age = age;
    this.address = address;
    this.famSize = famSize;
    this.pStatus = pStatus;
    this.medu = medu;
    this.fedu = fedu;
```

```
        this.mJob = mJob;
        this.fJob = fJob;
        this.reason = reason;
        this.guardian = guardian;
        this.travelTime = travelTime;
        this.studyTime = studyTime;
        this.failures = failures;
        this.schoolsUp = schoolsUp;
        this.farmsUp = farmsUp;
        this.paid = paid;
        this.activities = activities;
        this.nursery = nursery;
        this.higher = higher;
        this.internet = internet;
        this.romantic = romantic;
        this.famRel = famRel;
        this.freeTime = freeTime;
        this.goOut = goOut;
        this.dalc = dalc;
        this.walc = walc;
        this.health = health;
        this.absence = absence;
        this.passed = passed;
    }

    //To String method to convert student object to comma separated Strings
    @Override
    public String toString() {
        return school +
                "," + sex +
                "," + age +
                "," + address +
                "," + famSize +
                "," + pStatus +
                "," + medu +
                "," + fedu +
                "," + mJob +
                "," + fJob +
                "," + reason +
                "," + guardian +
                "," + travelTime +
                "," + studyTime +
                "," + failures +
                "," + convertBoolToString(schoolsUp) +
                "," + convertBoolToString(farmsUp) +
                "," + convertBoolToString(paid) +
                "," + convertBoolToString(activities) +
                "," + convertBoolToString(nursery) +
                "," + convertBoolToString(higher) +
                "," + convertBoolToString(internet) +
                "," + convertBoolToString(romantic) +
                "," + famRel +
                "," + freeTime +
                "," + goOut +
                "," + dalc +
                "," + walc +
                "," + health +
```

```
                "," + absence +
                "," + convertBoolToString(passed);
    }


//Convert the boolean object back to user specified String
private String convertBoolToString(boolean studentVariable) {
    return studentVariable?"yes":"no"; //If bool is True convert to yes else
no
}


    //Equals method compare two object of same type (deep comparison) to tell
if they are both equal
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;                    //If both are the
same object
        if (!(obj instanceof Student student)) return false;   // If your
comparing with some other object tye
        //Convert the type of Object to Student
        return Objects.equals(school, student.school) &&
                Objects.equals(mJob, student.mJob) &&
                Objects.equals(fJob, student.fJob) &&
                Objects.equals(reason, student.reason) &&
                Objects.equals(guardian, student.guardian) &&
                Objects.equals(famSize, student.famSize) &&
                Objects.equals(sex, student.sex) &&
                Objects.equals(address, student.address) &&
                Objects.equals(pStatus, student.pStatus) &&
                Objects.equals(age, student.age) &&
                Objects.equals(famRel, student.famRel) &&
                Objects.equals(freeTime, student.freeTime) &&
                Objects.equals(goOut, student.goOut) &&
                Objects.equals(dalc, student.dalc) &&
                Objects.equals(walc, student.walc) &&
                Objects.equals(health, student.health) &&
                Objects.equals(absence, student.absence) &&
                Objects.equals(travelTime, student.travelTime) &&
                Objects.equals(studyTime, student.studyTime) &&
                Objects.equals(failures, student.failures) &&
                Objects.equals(medu, student.medu) &&
                Objects.equals(fedu, student.fedu) &&
                Objects.equals(schoolsUp, student.schoolsUp) &&
                Objects.equals(farmsUp, student.farmsUp) &&
                Objects.equals(paid, student.paid) &&
                Objects.equals(activities, student.activities) &&
                Objects.equals(nursery, student.nursery) &&
                Objects.equals(higher, student.higher) &&
                Objects.equals(internet, student.internet) &&
                Objects.equals(romantic, student.romantic) &&
                Objects.equals(passed, student.passed);

    }
}
```

1.2 Linked List Class

```
/*
  @author: Kowshick Srinivasan
 * @version: 1.0
 * @Assignment: Hw2
 */

/**
 * Class to define the structure of the linked list
 */
public class LinkedList {
    Node head; //Contains  object of type Node

    public static class Node {  //static inner class to define the structure
of node

        Student student; //Contains object of student

        Node next;  //Contains an object of itself

        Node(Student student) {  //Constructor to initialise
            this.student = student;
            next = null;  //the next node is null by default
        }
    }

    /**
     * @param prevItem The current last Node/pointer
     * @param student  The student object that needs to be inserted
     * @param nextNode The following Node which needs to be pointed by the
inserted node
     * @return new last node
     */
    public Node insert(Node prevItem, Student student, Node nextNode) {
        Node newStudent = new Node(student);  //Create the new student node
        newStudent.next = nextNode;
        if (prevItem == null)
            this.head = newStudent;  //If linked list is empty, make the new
student as the first/head node
        else
            prevItem.next = newStudent; //Connect the previous node next to
the new student node, extending the linked list chain
        return newStudent;
    }

    /**
     * The method is to delete the current node from the linked list.
     *
     * @param prevNode The current last Node/pointer
     * @param nextNode The immediate next node after the current Node
     */
    public void delete(Node prevNode, Node nextNode) {
        //We can assign the next node pointer of the previous node to the
next node instead of the current node.
```

```
        // Since the current node is hanging in the heap space the garbage
collector will remove it.
        if (prevNode == null) head = nextNode; //Delete at the head
        else prevNode.next = nextNode;
    }
}
```

1.3 MemoryDatabase class

```
import java.io.*;
import java.nio.file.*;
import java.util.Objects;
import java.util.Scanner;
/*
  @author: Kowshick Srinivasan
 * @version: 1.0
 * @Assignment: Hw2
 */

/**
 * Memory database class that does:
 * <p>
 * a) load the student-data.csv file into the memory database through
recursive function.
 * <p>
 * b) When a row is added or removed from the CSV file, the memory database
will adapt to the changes.
 * <p>
 * c) export the memory database into a csv file through recursive function.
 */
public class MemoryDatabase {

    LinkedList list;  //Object of the linked list

    private static final String BASEPATH =
"/home/kowshick/Downloads/AlgoIndHw2-main/IndHw2/Data"; // target directory


    private static final String FILENAME = "student-data.csv";  //input file
name

    private static final String OUTFILE = "student-op-data.csv"; //output
file name

    LinkedList.Node prevNode;

    Scanner sc;  //Scanner object

    private static boolean running = true;  //Condition variable

    //Constructor based dependency injection
    public MemoryDatabase(LinkedList list, Scanner sc, LinkedList.Node
prevNode) {
        this.list = list;
        this.sc = sc;
```

```
        this.prevNode = prevNode;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        MemoryDatabase database = new MemoryDatabase(
                new LinkedList()  //Initialize the list object
                , new Scanner(System.in), null); //Initialize the scanner
object
File inputFile = new File(BASEPATH,FILENAME);
File outputFile = new File(BASEPATH,OUTFILE);

        //Since we used try with resources, we won't be needing a finally
block
        try (BufferedReader br = new BufferedReader(new
FileReader(inputFile)); //Buffered reader to enable to the Java methods to
read from the given file
             PrintWriter printWriter = new PrintWriter(new
FileWriter(outputFile))) //Buffered writer to enable the Java methods to
write on the given file
        {
            String header = br.readLine();  //read the Header of the csv
            database.read(br, database.list.head);   //Read the student
details from the file and add it to the linked list
            System.out.println("Press 0 to exit");   //KeyBoard interrupt to
signal make watcher thread to stop and join back to the main
            Thread watcherThread = new Thread(() -> {  //Load the watcher
method to the watcherThread
                try {
                    database.watcher();
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            });
            watcherThread.start();  //Start the watcher thread
            while (true) {
                if (scanner.hasNext() && Objects.equals(scanner.nextLine(),
"0")) {  //If there is any keyboard input, and it is 0
                    running = false; //change the contention variable to
false
                    //watcherThread.join();   //Resume after the completion
of watcher thread
                    System.out.println("Stopping watcher..."); //Stop the
watcher service
                    break;
//                    }
                }
            }


            printWriter.println(header);  //print the Header of the csv
            database.print(database.list.head, printWriter);  //Method to
print the traverse the Linked list and print in the csv


        } catch (Exception e) { //Global catch block to handle all the
```

```
exception thrown by the program
            e.printStackTrace(System.out);  //print the exception stack trace
in console

        }

    }


    /**
     * Method recursively read each student details from the CSV file and
adds it to the end of the linked list
     *
     * @param br      : The BufferedReader object
     * @param current : Current Node/pointer of the linked list
     * @return : Completion status of the read operation
     */
    private int read(BufferedReader br, LinkedList.Node current) throws
IOException {
        String l = br.readLine();  //Read each line
        if (l == null) return 0;  //When reached EOF return
        String[] line = l.split(",");  //Split the row using "," as the
separator
        Student student = convertLineToStudent(line);  //Convert the String
array to the student class
        return read(br, list.insert(current, student, null));   //Recursively
call the function with the buffered reader object and updated last
Node/Pointer
    }


    /**
     * Accepts the String array and convert it to student object
     *
     * @param line Original String array formed after comma separation of the
line
     * @return Student object
     */
    private Student convertLineToStudent(String[] line) {
        return new Student(line[0], line[1].charAt(0),
Integer.parseInt(line[2]), line[3].charAt(0), line[4], line[5].charAt(0),
                Integer.parseInt(line[6]), Integer.parseInt(line[7]),
line[8], line[9], line[10], line[11],
                Integer.parseInt(line[12]), Integer.parseInt(line[13]),
Integer.parseInt(line[14]), line[15].equals("yes"), line[16].equals("yes"),
line[17].equals("yes"), line[18].equals("yes"), line[19].equals("yes"),
                line[20].equals("yes"), line[21].equals("yes"),
line[22].equals("yes"), Integer.parseInt(line[23]),
Integer.parseInt(line[24]), Integer.parseInt(line[25]),
Integer.parseInt(line[26]),
                Integer.parseInt(line[27]), Integer.parseInt(line[28]),
Integer.parseInt(line[29]), line[30].equals("yes"));

    }


    /**
```

```
     * Recursively prints the student data to the csv file
     *
     * @param current     : Accepts the current NOde/pointer of the Linked
List
     * @param printWriter : Accepts the Printed writer object
     * @return : return the completion status of the printing action
     */
    private int print(LinkedList.Node current, PrintWriter printWriter) {
        String row = current.student.toString(); //Converts the student
object to comma separated string
        printWriter.println(row);  //Prints the string to the csv file
        if (current.next == null)  //When reached end node return completion
status
            return 0;
        return print(current.next, printWriter); //recursively call the print
method with updated last node pointer
    }

    /**
     * Watches for any changes in the csv file
     */
    private void watcher() throws Exception {
        WatchService watchService =
FileSystems.getDefault().newWatchService(); //create a watcher service
        Path path = Paths.get(BASEPATH);
        path.register(watchService, StandardWatchEventKinds.ENTRY_MODIFY);
//Notify when there is a modification event

        while (running) {  //run the loop until the condition variable is
true
            WatchKey key = watchService.take(); // blocks until an event
            for (WatchEvent<?> event : key.pollEvents()) {
                WatchEvent.Kind<?> kind = event.kind();
                Path changed = (Path) event.context();
                if (changed.endsWith(FILENAME) && kind ==
StandardWatchEventKinds.ENTRY_MODIFY) {
                    System.out.println("student-data.csv has changed!");
//Notify there is a change detected in the file
                    BufferedReader br = new BufferedReader(new
FileReader(FILENAME)); //buffered reader object of the file
                    br.readLine();  //skip the header
                    reload(list.head, br);  //Reload only the changes

                }

            }
            boolean valid = key.reset();
            if (!valid) break;
        }
    }


    private int reload(LinkedList.Node current, BufferedReader br) throws
IOException {
        String row = br.readLine();
        if (current.next != null && row != null) {  //When both the linked
list and file is not at the end,
```

```
            // i.e. insertion/deletion happens at head in the middle
            String[] line = row.split(",");
            Student newStudent = convertLineToStudent(line);
            //Change detected
            if (!newStudent.equals(current.student)) {  //There is a change
in the current node and the file row, investigate further
                LinkedList.Node nextNode = current.next;
                if (nextNode.next == null ||
nextNode.student.equals(newStudent)) {   //if the current node is the last or
if the next node is same as the current row,
                    // then the current row has been deleted, so we need to
delete current node
                    list.delete(prevNode, nextNode); //deletion operation
                } else
                    list.insert(prevNode, newStudent, nextNode); //Insertion
operation
            }
            prevNode = current;  //Store the current node
            return reload(current.next, br);  //recursively call with the
next node and reader object
        } else if (current.next == null && row != null) {   //If the linked
list has reached its end but not the file,
            // then there is something to be inserted at the end of the
Linked list
            String[] line = row.split(",");
            Student newStudent = convertLineToStudent(line);
            LinkedList.Node newNode = list.insert(prevNode, newStudent,
null);
            prevNode = newNode; //Store the current node
            return reload(newNode, br);  //recursively call with the next
node and reader object
        } else if (current.next != null) {   //If the File has reached its
end and not the Linked List,
            // there is something to be deleted from the end of the
LinkedList
            list.delete(prevNode, null);  //delete all the Node that are
excess than the File
            //No need to recursively call since we need to delete everything
that are excess
        }
        return 0;     //both File and Linked list has reached the end
    }
}
```

2.Using Python

2.1Student Class

```
#@author Kowshick Srinivasan
#@version: 1.0
#@Assignment: Hw2
class Student:
    def __init__(self, school,
                 sex,
```

```
            age,
            address,
            fam_size,
            p_status,
            medu,
            fedu,
            m_job,
            f_job,
            reason,
            guardian,
            travel_time,
            study_time,
            failures,
            schools_up,
            farms_up,
            paid,
            activities,
            nursery,
            higher,
            internet,
            romantic,
            fam_rel,
            free_time,
            go_out,
            dalc,
            walc,
            health,
            absence,
            passed):
        self.school = school
        self.sex = sex
        self.age = age
        self.address = address
        self.fam_size = fam_size
        self.p_status = p_status
        self.medu = medu
        self.fedu = fedu
        self.m_job = m_job
        self.f_job = f_job
        self.reason = reason
        self.guardian = guardian
        self.travel_time = travel_time
        self.study_time = study_time
        self.failures = failures
        self.schools_up = schools_up
        self.farms_up = farms_up
        self.paid = paid
        self.activities = activities
        self.nursery = nursery
        self.higher = higher
        self.internet = internet
        self.romantic = romantic
        self.fam_rel = fam_rel
        self.free_time = free_time
        self.go_out = go_out
        self.dalc = dalc
        self.walc = walc
```

```
        self.health = health
        self.absence = absence
        self.passed = passed

    def __str__(self):
    return [self.school, self.sex, self.age, self.address, self.fam_size,
self.p_status, self.medu, self.fedu,
            self.m_job, self.f_job, self.reason, self.guardian,
self.travel_time, self.study_time, self.failures,
            self.convert_bool_to_string(self.schools_up),
self.convert_bool_to_string(self.farms_up),
self.convert_bool_to_string(self.paid),
            self.convert_bool_to_string(self.activities),
self.convert_bool_to_string(self.nursery),
self.convert_bool_to_string(self.higher),

self.convert_bool_to_string(self.internet),self.convert_bool_to_string(self.r
omantic), self.fam_rel, self.free_time, self.go_out,
            self.dalc, self.walc, self.health,
self.absence,self.convert_bool_to_string(self.passed)]


    def __eq__(self, obj):
        if not isinstance(obj, Student):
            return False
        return (self.school == obj.school and
                self.sex == obj.sex and
                self.age == obj.age and
                self.address == obj.address and
                self.fam_size == obj.fam_size and
                self.p_status == obj.p_status and
                self.medu == obj.medu and
                self.fedu == obj.fedu and
                self.m_job == obj.m_job and
                self.f_job == obj.f_job and
                self.reason == obj.reason and
                self.guardian == obj.guardian and
                self.travel_time == obj.travel_time and
                self.study_time == obj.study_time and
                self.failures == obj.failures and
                self.schools_up == obj.schools_up and
                self.farms_up == obj.farms_up and
                self.paid == obj.paid and
                self.activities == obj.activities and
                self.nursery == obj.nursery and
                self.internet == obj.internet and
                self.higher == obj.higher and
                self.romantic == obj.romantic and
                self.fam_rel == obj.fam_rel and
                self.free_time == obj.free_time and
                self.go_out == obj.go_out and
                self.dalc == obj.dalc and
                self.walc == obj.walc and
                self.health == obj.health and
                self.absence == obj.absence and
                self.passed == obj.passed)
```

```
def convert_bool_to_string(self, boolean_variable):
    if boolean_variable:
        return "yes"
    else:
        return "no"
```

2.2 Linked List Class

```
#@author Kowshick Srinivasan
#@version: 1.0
#@Assignment: Hw2
class Node: #class to define the structure of node
    def __init__(self, student):
        self.student = student #Contains object of student
        self.next = None  #Contains an object of itself,the next node is null
by default

class LinkedList:
    def __init__(self):
        self.head = None #Contains  object of type Node

    def insert(self, current_node, student, next_node):
        """

        :param current_node: The current last Node/pointer
        :param student: The student object that needs to be inserted
        :param next_node: The following Node which needs to be pointed by the
inserted node
        :return: new last node
        """
        new_student = Node(student) #Create the new student node
        new_student.next = next_node
        if current_node is None:
            self.head = new_student #If linked list is empty, make the new
student as the first/head node
        else:
            current_node.next = new_student #Connect the previous node next
to the new student node, extending the linked list chain
        return new_student

    def delete(self, prev_node, next_node):
        """
        The method is to delete the current node from the linked list.
        :param prev_node: The current last Node/ pointer
        :param next node: The immediate next node after the current Node
        """
        # We can assign the next node pointer of the previous node to the
next node instead of the current node.
        # Since the current node is hanging in the heap space the garbage
collector will remove it.
        if prev_node is None:

            self.head = next_node  # Delete at the head
        else:
            prev_node.next = next_node
```

2.3 MemoryDatabase class

```
#@author Kowshick Srinivasan
#@version: 1.0
#@Assignment: Hw2
import csv
import os
import threading

from LinkedList import LinkedList
from Student import Student


def convert_bool(param):
    if param == "yes":
        return True
    else:
        return False


def convert_to_student(row):
    line = list(row)
    return Student(line[0], line[1], int(line[2]), line[3], line[4], line[5],
                   int(line[6]), int(line[7]), line[8], line[9], line[10],
line[11],
                   int(line[12]), int(line[13]), int(line[14]),
convert_bool(line[15]), convert_bool(line[16]),
                   convert_bool(line[17]), convert_bool(line[18]),
convert_bool(line[19]),
                   convert_bool(line[20]), convert_bool(line[21]),
convert_bool(line[22]), int(line[23]), int(line[24]),
                   int(line[25]), int(line[26]),
                   int(line[27]), int(line[28]), int(line[29]),
convert_bool(line[30]))


# Memory database class that does:
# a) load the student-data. csv file into the memory database through
recursive function.
# b) When a row is added or removed from the CSV file, the memory database
will adapt to the changes.
# c) export the memory database into a csv file through recursive function.
class MemoryDatabase:
    BASEPATH = r"/home/kowshick/Downloads/AlgoIndHw2-main/IndHw2/Data"
    FILENAME = os.path.join(BASEPATH, 'student-data.csv')
    OUTFILE = os.path.join(BASEPATH, 'student-op-data.csv')

    def read(self, current_node, input_csv):
        """
        Recursive method to read the CSV file and add the contends to the
back of the Linked list
        :param current_node: The current Node
        :param input_csv: Reader object
        :return: Completion status of the read operation
        """
```

```
        row = next(input_csv, None)
        if row is None:
            return 0
        student = convert_to_student(row)
        return self.read(self.list.insert(current_node, student, None),
input_csv)

    # Constructor based dependency injection
    def __init__(self):
        self.list = LinkedList()
        self.running = True
        self.prev_node = None

    def run(self):
        with open(self.FILENAME, newline="") as inputFile:
            input_csv = csv.reader(inputFile)
            header = next(input_csv, None)  # read the Header of the csv
            self.read(self.list.head, input_csv)  # Read the student details
from the file and add it to the linked list
        print('enter 0 to stop watching')  # KeyBoard interrupt to signal
make watcher thread to stop and join back to the main
        watcher_thread = threading.Thread(target=self.file_watcher)  # Load
the watcher method to the watcherThread
        watcher_thread.start()  # Start the watcher thread

        if input() == '0':  # Blocking-instruction, makes the main thread to
wait until there is user input 0
            self.running = False  # Signal the watcher thread to stop

        watcher_thread.join()  # Resume after the completion of watcher
thread

        with open(self.OUTFILE, mode='w', newline="") as outputFile:
            output_csv_writer = csv.writer(outputFile)
            output_csv_writer.writerow(header)  # print the Header of the csv
            self.printer(self.list.head,
                         output_csv_writer)  # Method to print the traverse
the Linked list and print in the csv

    def printer(self, current, output_csv_writer):
        """
        Recursively prints the student data to the csv file
        :param current: Accepts the current NOde/ pointer of the Linked List
        :param output_csv_writer: Accepts the Printer writer object
        :return: completion status of the printing action
        """
        row = current.student.__str__()
        output_csv_writer.writerow(row)
        if current.next is None:
            return 0
        return self.printer(current.next, output_csv_writer)

    # Watches for any changes in the csv file
    def file_watcher(self):
        last_modified = None
        if os.path.exists(self.FILENAME):
            last_modified = os.path.getmtime(self.FILENAME)
```

```
        while self.running:  # run the loop until the condition variable is
true
            if os.path.exists(self.FILENAME):
                current_modified = os.path.getmtime(self.FILENAME)
                if last_modified is None or current_modified !=
last_modified:
                    print("changes detected in file")
                    with open(self.FILENAME, mode='r', newline='') as
modified_file:
                        modified_csv = csv.reader(modified_file)
                        next(modified_csv, None)  # skip header
                        self.reload(self.list.head, modified_csv)  # Reload
only the changes
                        last_modified = current_modified

    def reload(self, current_node, input_csv):
        row = next(input_csv, None)
        # When both the linked list and file is not at the end,
        # i.e. insertion/deletion happens at head in the middle
        if current_node.next is not None and row is not None:
            new_student = convert_to_student(row)
            # There is a change in the current node and the file row,
investigate further
            if not new_student.__eq__(current_node.student):  #######problem
                next_node = current_node.next
                # if the current node is the last or if the next node is same
as the current row,
                # then the current row has been deleted, so we need to delete
current node
                if next_node.next is None or
next_node.student.__eq__(new_student):
                    self.list.delete(self.prev_node, next_node)  # deletion
operation
                else:
                    self.list.insert(self.prev_node, new_student, next_node)
# Insertion operation

            self.prev_node = current_node  # Store the current node
            return self.reload(current_node.next, input_csv)  # recursively
call with the next node and reader object
        # If the linked list has reached its end but not the file,
        # then there is something to be inserted at the end of the Linked
list
        elif current_node.next is None and row is not None:
            new_student = convert_to_student(row)
            new_node = self.list.insert(self.prev_node, new_student, None)
            self.prev_node = new_node  # Store the current node
            return self.reload(new_node, input_csv)  # recursively call with
the next node and reader object
        # If the File has reached its end and not the Linked List,
        # there is something to be deleted from the end of the LinkedList
        elif current_node.next is not None:
            self.list.delete(self.prev_node, None)
            # No need to recursively call since we need to delete everything
that are excess
        return 0  # both File and Linked list has reached the end
```

```
if __name__ == '__main__':
    db = MemoryDatabase()
    db.run()
```

The above source code can also found in https://github.com/kowshick20/AlgoIndHw2.git

The source code can also be found in the onlineGDB

1. JAVA: https://onlinegdb.com/CdqiP8PKl
2. Python: https://onlinegdb.com/9X9gvdQdy