CCP6114 Programming Fundamentals

Trimester: 2430

Lecture: TC8L

Tutorial: T16L

Group:G04

Assignment title: Light Mariadb Interpreter

No.	Student Name	Student ID
1.	Wong Kai Shen	243UC247DH
2.	Teh Shin Rou	243UC2467K
3.	Nyiam Zi Qin	243UC2466T
4.	YEN MING JUN	243UC246NQ

Submission Dates

Assignment milestone 1: 25 Dec 2024, Wed, 5:00pm, Week 8

Assignment milestone 2: 22 Jan 2024, Wed, 5:00pm, Week 12

Table of contents:

Introduction (introduce the basic idea of the program).

The *Light MariaDB Interpreter* assignment aims to provide a simplified simulation of a database management system using C++. This project involves building a program capable of processing and executing SQL-like commands to manage a lightweight database. The primary functionalities include creating databases and tables, inserting and updating records, and retrieving data through query commands. Through this assignment, we explore key programming concepts such as file handling, class design, data structures (e.g., vectors), and error handling in C++. The project emphasizes understanding database principles and applying them programmatically, offering a practical approach to solving real-world problems using programming fundamentals. This document outlines the design, implementation, testing, and functionality of the *Light MariaDB Interpreter*, showcasing how it meets the assignment's requirements.

1. **File Operations (Opening Files):**

Opening Input/Output File:

```
ifstream inFile("fileinput1.mdb");
ofstream outFile("fileoutput1.txt");
string line;
vector<string> headers;
vector<vector<string>> data;
bool inCreateTable = false;
string tableName;

if (!inFile) {
    cout << "Error opening fileinput1.mdb" << endl;
    return 1;
}
```

The ifstream object inFile opens the file fileinput1.mdb for reading.

The ofstream object outFile opens the file fileoutput1.txt for writing.

2. **Reading and Parsing SQL:**

Main Loop (Processing Each Line):

```
while (getline(inFile, line)) {
```

This loop reads each line from the fileinput1.txt file using getline and processes it until the end of the file.

Detecting CREATE TABLE Block:

```
if (line.find("CREATE TABLE") != string::npos) {
    inCreateTable = true;
    size_t start = line.find("CREATE TABLE") + 12;
    size_t end = line.find("(");
    tableName = line.substr(start, end - start);  // Extract table name
}
```

When the line contains CREATE TABLE, the program sets inCreateTable to true and extracts the table name from the line. This marks the start of a CREATE TABLE block.

Extracting Column Names (Inside CREATE TABLE)

```
// Parse headers within the CREATE TABLE block
if (inCreateTable) {
    size_t start = line.find('(');
    size_t end = line.find(')');
    if (start != string::npos) {
        string columns = line.substr(start + 1);  // Skip the '('
        stringstream ss(columns);
        string column;

        // Process columns from the current line
        while (getline(ss, column, ',')) {
            // Remove spaces from the column name and data type
            size_t spacePos = column.find(' ');
            if (spacePos != string::npos) {
                // Extract only the column name
                headers.push_back(column.substr(0, spacePos));
            }
        }
    }
```

This block is executed when inCreateTable is true. It processes the columns inside the parentheses and extracts the column names from the SQL CREATE TABLE statement. These names are added to the headers vector.

End of CREATE TABLE Block:

```
    // If line contains the closing parenthesis, finish parsing the table schema
    else if (end != string::npos) {
        stringstream ss(line);
        string column;
        while (getline(ss, column, ',')) {
            size_t spacePos = column.find(' ');
            if (spacePos != string::npos) {
                // Extract only the column name
                headers.push_back(column.substr(0, spacePos));
            }
        }
        inCreateTable = false; // End of CREATE TABLE block
    }
}
```

If the line contains the closing parenthesis), it means the CREATE TABLE block has ended. The columns are processed and inCreateTable is set to false to indicate the block is finished.

3. **Parsing VALUES (Data Insertion):**

Detecting VALUES Block:

```
else if (line.find("VALUES") != string::npos) {
    // Parse the VALUES line to extract data
    size_t start = line.find('(') + 1;
    size_t end = line.find(')');
    string values = line.substr(start, end - start);
```

When a line containing VALUES is found, it extracts the data inside the parentheses by finding the positions of the opening and closing parentheses and capturing the substring.

Processing Data Values:

```
        vector<string> row;
        string value;
        bool inQuote = false;

        for (char c : values) {
            if (c == '\'') {
                inQuote = !inQuote;
            } else if (c == ',' && !inQuote) {
                if (!value.empty()) {
                    row.push_back(value);
                    value.clear();
                }
            } else if (inQuote || (!inQuote && (isdigit(c) || isalpha(c) || c == ' '))) {
                value += c;
            }
        }
        if (!value.empty()) {
            row.push_back(value);
        }
        data.push_back(row);
    }
}
```

This block processes the values within the VALUES line:

It handles quoted strings by toggling the inQuote flag whenever a single quote (') is encountered.It splits the values based on commas (ignoring commas inside quoted strings) and stores them in a row vector. After processing all values, the row is added to the data vector.

4. **Writing Data to Output File and Terminal (CSV Format)**

Writing the Table Structure (Headers):

```
// Write the CSV headers to terminal and file
outFile << "CREATE TABLE " << tableName << "(" << endl;
for (size_t i = 0; i < headers.size(); i++) {
    outFile << "  " << headers[i] << " TEXT";
    if (i < headers.size() - 1) {
        outFile << ",";
    }
    outFile << endl;
}
outFile << ");" << endl << endl;
```

This block writes the table creation statement (CREATE TABLE) in CSV format to the output file. It loops through the headers vector to output the column names.

Writing CSV Header Row:

```
// Write the CSV header row
for (size_t i = 0; i < headers.size(); i++) {
    cout << headers[i];
    outFile << headers[i];
    if (i < headers.size() - 1) {
        cout << ",";
        outFile << ",";
    }
}
cout << endl;
outFile << endl;
```

The program outputs the column headers (from headers) both to the terminal and to the output file in CSV format.

```
// Write CSV rows (data) to terminal and file
for (const auto& row : data) {
    for (size_t i = 0; i < row.size(); i++) {
        cout << row[i];
        outFile << row[i];
        if (i < row.size() - 1) {
            cout << ",";
            outFile << ",";
        }
    }
    cout << endl;
    outFile << endl;
}
```

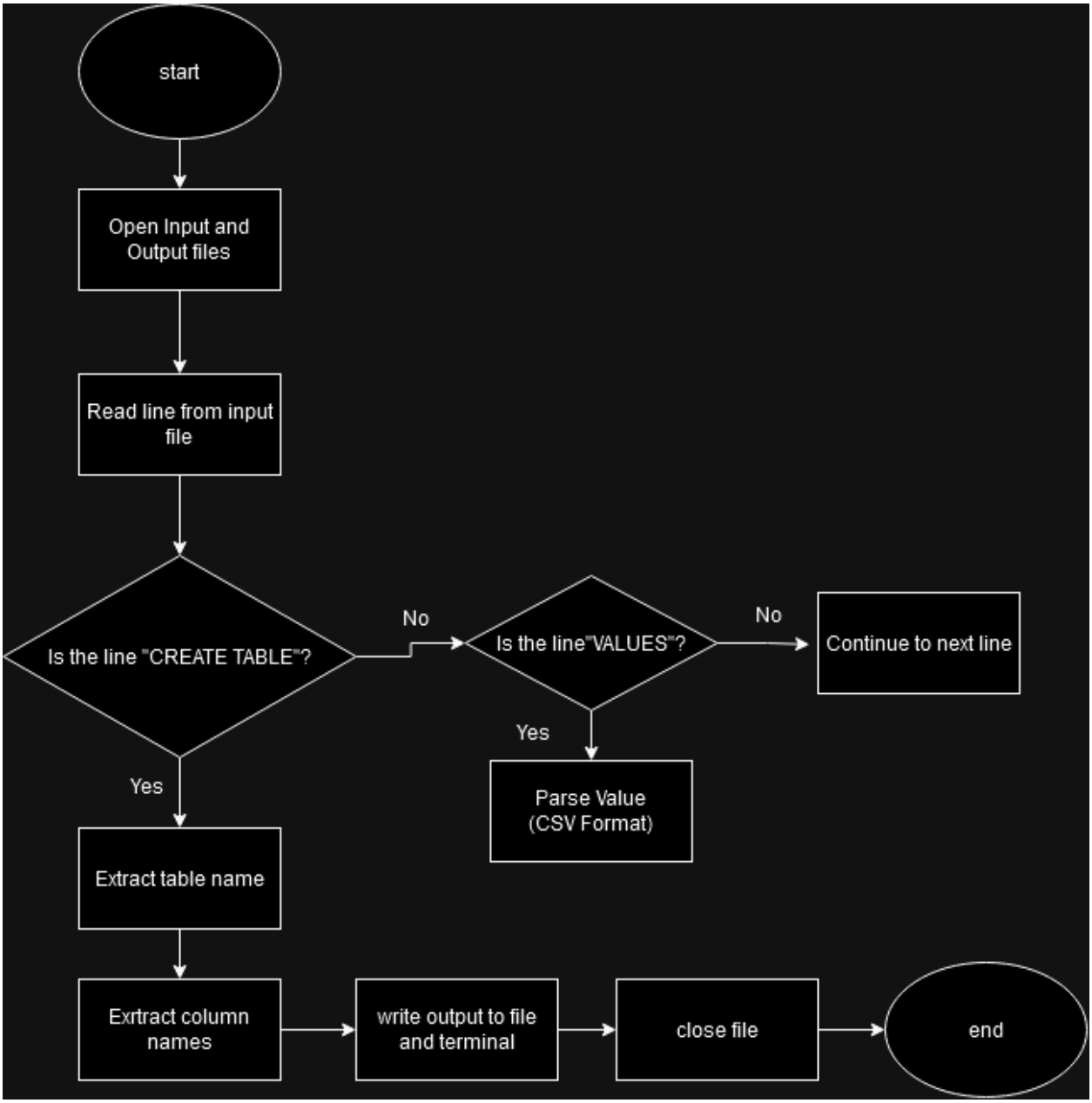
This block writes each row of data (from data) to both the terminal and the output file in CSV format, separating each value with commas.

5) Closing Files

```
inFile.close();
outFile.close();
```

Finally, the input and output files are closed using close() to free up resources.

5. Flowchart:


start
Open Input and Output files
Read line from input file
Is the line "CREATE TABLE"?
No
Is the line"VALUES""?
No
Continue to next line
Yes
Parse Value (CSV Format)
Yes
Extract table name
Exrtract column names
write output to file and terminal
close file
end