A blue and black logo

Description automatically generated

**CCP6114**

**PROGRAMMING FUNDAMENTALS**

**ASSIGNMENT DOCUMENTATION**

|  |  |  |
| --- | --- | --- |
| Lecture Section | : | TC7L |
| Tutorial Section | : | T13L |
| Group Number/Group Name | : | G03 |
| Submitted to | : | Dr. Lee Fong Yee |

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Email Address** |
| **243UC247BR** | **LIM PEI SHAN** | **LIM.PEI.SHAN@student.mmu.edu.my** |
| **243UC247D6** | **CHAN KER JING** | **CHAN.KER.JING@student.mmu.edu.my** |
| **243UC247C1** | **BERNICE GAN XINYI** | **BERNICE.GAN.XINYI@student.mmu.edu.my** |
| **243UC247CB** | **TEOW YAN PING** | **TEOW.YAN.PING @student.mmu.edu.my** |

Table of Contents

[Pseudocodes 3](#_Toc188782493)

[Program 1: TC7L\_G03\_main1.cpp 3](#_Toc188782494)

[Program 2: TC7L\_G03\_main2.cpp 7](#_Toc188782495)

[Pseudocode Explanation of Program 2 9](#_Toc188782496)

[Structure diagrams 11](#_Toc188782497)

[Program 1: TC7L\_G03\_main1.cpp 11](#_Toc188782498)

[Explanation of Program 1 Structure Diagram 11](#_Toc188782499)

[Program 2: TC7L\_G03\_main2.cpp 13](#_Toc188782500)

[Explanation of Program 2 Structure Diagram 14](#_Toc188782501)

[Input 15](#_Toc188782502)

[Sample 1 15](#_Toc188782503)

[Sample 2 15](#_Toc188782504)

[Sample 3 16](#_Toc188782505)

[Output 17](#_Toc188782506)

[Sample 1 17](#_Toc188782507)

[Explanation of fileOutput1.txt 17](#_Toc188782508)

[fileOutput1.csv 18](#_Toc188782509)

[Sample 2 19](#_Toc188782510)

[Explanation of fileOutput2.txt 19](#_Toc188782511)

[fileOutput2.csv 21](#_Toc188782512)

[Sample 3 21](#_Toc188782513)

[Output of Sample 3 to command line 21](#_Toc188782514)

[Explanation of Sample Output 3 22](#_Toc188782515)

[fileOutput3.csv 23](#_Toc188782516)

[Code segments explanation 24](#_Toc188782517)

[Program 1: TC7L\_G03\_main1.cpp 24](#_Toc188782518)

[Program 2: TC7L\_G03\_main2.cpp 31](#_Toc188782519)

[Task Distribution 37](#_Toc188782520)

[Student 1 37](#_Toc188782521)

[Student 2 37](#_Toc188782522)

[Student 3 37](#_Toc188782523)

[Student 4 37](#_Toc188782524)

# Pseudocodes

## Program 1: TC7L\_G03\_main1.cpp

**START**

**OPEN the input file**

**IF filename is equal to "fileInput1.mdb"**

**Open "fileOutput1.csv"**

**Open "fileOutput1.txt"**

**ELSE**

**Open "fileOutput2.txt"**

**Open "fileOutput2.csv"**

**IF the input file opened successfully**

**IF the file is empty**

**Display the message “The file is empty.”**

**ELSE**

**WHILE the file containing lines**

**IF found a semicolon in the line**

**IF the file contains the line “CREATE”**

**Call the function createTable**

**ELSE IF the file contains the line “DATABASES”**

**Output the path of the file**

**ELSE IF the file contains the line “INSERT INTO”**

**Call the function insertIntoTable**

**ELSE IF the file contains the line “SELECT \* FROM”**

**Call the function selectFromTable**

**ELSE IF the file contains the line “TABLES”**

**Output the table name**

**ELSE IF the file contains the line “UPDATE”**

**Call the function updateFromTable**

**ELSE IF the file contains the line “DELETE”**

**Call the function deleteFromTable**

**ELSE IF the file contains the line “SELECT COUNT”**

**Call the function countFromTable**

**Clear the command buffer**

**ENDIF**

**END WHILE**

**CLOSE the input file after reading it**

**ELSE**

**Display message: “Error opening the file.”**

**ENDIF**

**IF there is no customer to display after reading the file**

**Display message “No customer available in the 'customer' table to view in CSV format.”**

**ELSE**

**Display the header of the table**

**WRITE the header of the table to output file**

**FOR counter start from 0 TO customer\_count STEP customer\_count - 1 DO**

**WRITE the columns to the csv files including the customer\_id, customer\_name, customer\_city, customer\_state, customer\_country, customer\_phone, customer\_email each separated with a comma in a csv file**

**ENDFOR**

**CLOSE the output file after writing data into it**

**CLOSE the csv file after writing data into it**

**ENDIF**

**END**

Pseudocode Explanation of Program 1

**Purpose of the program:** This program will request an input file from user and read from the file to write it into an output file int txt and csv format.

**Explanation:**

1. If the file opens successfully, it checks whether the file is empty. It will also check the filename, if the filename is “fileInput1.mdb”, open “fileOutput1.txt” and “fileOutput1.csv”. Otherwise, it will open “fileOutput2.txt” and “fileOutput2.csv”. If the file is empty, it will output “The file is empty.” Otherwise, it will enter a while loop. The while loop checks if the file contains any line to read. If find a semicolon in the line, continue to check if the file has a SQL command such as:
   1. “CREATE”: It will call the function createTable() and open csv files based on the input file.
   2. “DATABASES”: This command uses the file system library to output the file path.
   3. “INSERT INTO”: It will call the function insertIntoTable()
   4. “SELECT \* FROM”: Call the function selectFromTable()
   5. “TABLES”: Output the name of the table.
   6. “UPDATE”: Call the function updateFromTable()
   7. “DELETE FROM”: Call the function deleteFromTable()
   8. “SELECT COUNT”: Call the function countFromTable()

After that, clear command buffer to avoid program error. After no more lines to read from the input file, close the input file. If the file does not open successfully, output “Error opening the file.”

1. After reading the file, another if statement checks if there is any information to be written into output files: txt and csv file. If no information, output “No customer available in the ‘customer’ table to view in CSV format. If there is information, check which input files that user wishes to read. After writing into an output file, close the output files and csv files.
2. Explanation about the function:
   1. createTable(): This function with parameter line passed by reference will process the “CREATE TABLE” command. It will output to inform user that the table structure is been initiated to store customer data.
   2. selectFromTable(): This function with parameter line passed by reference will output to user that the program is processing the “SELECT” command.
   3. insertIntoTable(): This function has 9 parameters, containing the line of the input files and 8 vectors to store the customer data passed by reference. It has a regex library to detect the characters like space, ( , ) and ‘ .It will search a string for a match to the regex and push the data into vector.
   4. updateFromTable(): This function with parameter line passed by reference processes the “UPDATE” command. It uses a regex to extract the customer email and customer ID that needs to be updated to the list of customers. If the customer is found, its customer email changes based on the instruction given. If the customer is found, it updates it to the vector; otherwise, it outputs an error message if the format is invalid, or the customer ID is not found.
   5. deleteFromTable(): This function with the parameter line passed by reference processes the “DELETE FROM” command. It uses a regex to extract the customer ID that needs to be deleted from the list of customers. If the customer is found, it removes it from the vector; otherwise, it outputs an error message if the format is invalid, or the customer ID is not found.
   6. countFromTable(): This function with the parameter line passed by reference processes the “SELECT COUNT” command. It uses a regex to extract the table name and counts the number of customers in the vector. The resulting count is then printed to the console and written to the output file. If the format is invalid, it outputs an error message.

## Program 2: TC7L\_G03\_main2.cpp

**START**

**OPEN the input file**

**IF the input file opened successfully**

**IF the file is empty**

**Display the message “The file is empty.”**

**ELSE**

**WHILE the file containing lines**

**IF found a semicolon in the line**

**IF the file contains the line “CREATE TABLE”**

**Call the function createTable**

**Open "fileOutput3.csv"**

**ELSE IF the file contains the line “SELECT \* FROM”**

**Call the function selectFromTable**

**ELSE IF the file contains the line “INSERT INTO”**

**Call the function insertIntoTable**

**ELSE IF the file contains the line “TABLES”**

**Output the name of the table**

**ELSE IF the file contains the line “UPDATE”**

**Call the function updateFromTable**

**ELSE IF the file contains the line “DELETE FROM”**

**Call the function deleteFromTable**

**ELSE IF the file contains the line “SELECT COUNT”**

**Call the function countFromTable**

**Clear the command buffer**

**END WHILE**

**ENDIF**

**CLOSE the input file after reading it**

**ELSE**

**Display message: “Error opening the file.”**

**ENDIF**

**IF there is no book to display after reading the file**

**Display message “No books available in the 'books' table to view in CSV format.”**

**ELSE**

**Display the header of the table to terminal**

**WRITE the header of the table to output file**

**FOR counter start from 0 TO book\_count STEP book\_count - 1 DO**

**Display the columns including the book\_id, book\_name, book\_author, book\_year, book\_category, book\_status, book\_quantity, book\_pricing, each separated with a comma**

**WRITE the columns to the csv file including the book\_id, book\_name, book\_author, book\_year, book\_category, book\_status, book\_quantity, book\_pricing, each separated with a comma**

**ENDFOR**

**CLOSE the output file after writing data into it**

**ENDIF**

**END**

### Pseudocode Explanation of Program 2

**Purpose:** This program will request an input file from user and read from the file to write it into an output file int txt and csv format.

**Explanation:**

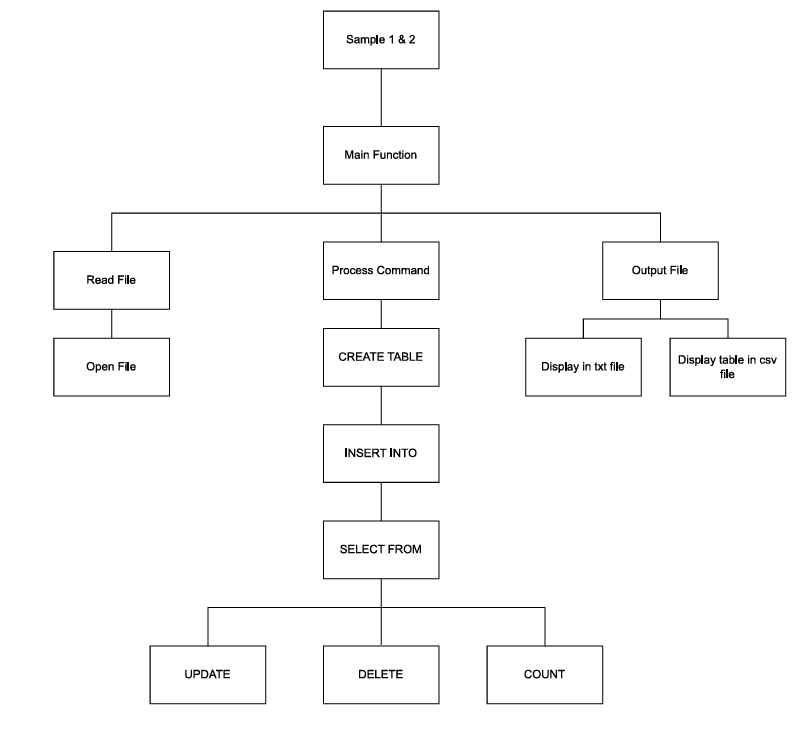
1. If the file opens successfully, it checks whether the file is empty. If the file is empty, it will output “The file is empty.” Otherwise, it will enter a while loop. The while loop checks if the file contains any line to read. If find a semicolon in the line, continue to check if the file has a SQL command such as:
   1. “CREATE TABLE”: It will call the function createTable() and open csv files based on the input file.
   2. “INSERT INTO”: It will call the function insertIntoTable()
   3. “SELECT \* FROM”: Call the function selectFromTable()
   4. “TABLES”: Output the name of the table.
   5. “UPDATE”: Call the function updateFromTable()
   6. “DELETE FROM”: Call the function deleteFromTable()
   7. “SELECT COUNT”: Call the function countFromTable()

After that, clear command buffer to avoid program error. After no more lines to read from the input file, close the input file. If the file does not open successfully, output “Error opening the file.”

1. After reading the file, another if statement checks if there is any information to be written into output files: txt and csv file. If no information, output “No books available in the ‘book’ table to view in CSV format. If there is information, display the header to terminal and write table header to csv file. Then, write the information get from database into the csv file and display to terminal. After writing into an output file, close the output files and csv files.
2. Explanation about the function:
   1. createTable(): This function with parameter line passed by reference will process the “CREATE TABLE” command. It will output to inform user that the table structure is been initiated to store book data.
   2. selectFromTable(): This function with parameter line passed by reference will output to user that the program is processing the “SELECT” command.
   3. insertIntoTable(): This function has parameter to pass line with reference and to pass the struct that stores the book vectors. It has a regex library to detect the characters like space, ( , ) and ‘ .It will search a string for a match to the regex and push the data into vector.
   4. updateFromTable(): This function with parameter line passed by reference processes the “UPDATE” command. It uses a regex to extract the book status and book ID that needs to be updated to the list of books. If the book is found, its book status changes based on the instruction given. If the book is found, it updates it to the vector; otherwise, it outputs an error message if the format is invalid, or the book ID is not found.
   5. deleteFromTable(): This function with the parameter line passed by reference processes the “DELETE FROM” command. It uses a regex to extract the book ID that needs to be deleted from the list of books. If the book is found, it removes it from the vector; otherwise, it outputs an error message if the format is invalid, or the book ID is not found.
   6. countBooks(): This function simulates the SQL “COUNT” function by returning the number of books in the books vector using the size() method. It is used after a “DELETE FROM” command to show how many books remain, and also in response to a “SELECT COUNT” query to display the current total number of books in the system.

# Structure diagrams

### Program 1: TC7L\_G03\_main1.cpp



Explanation of Program 1 Structure Diagram

**Overall Representation**

* **Title**: The diagram is labelled "Sample 1 & 2," indicating that it represents a specific example focused on managing customer data

**Main Function**

* **Main Function Block:** This is the central controller of the program, which manages the flow of operations.

**Key Components**

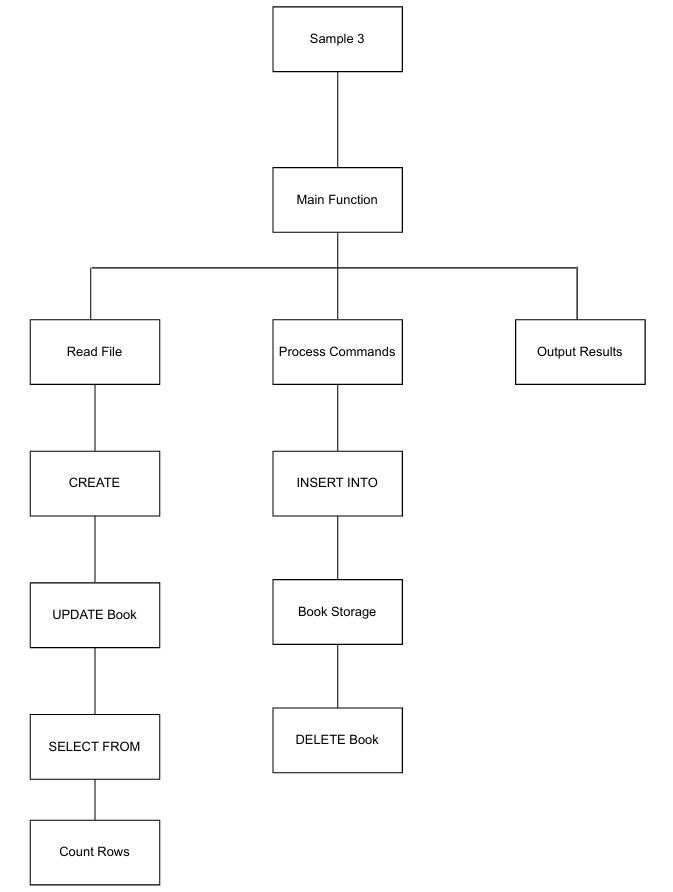
* **Read File:** This block handles the file input operation. It ensures that the program accesses the user-provided file.
* **Open File:** 
  1. Ensures that the input file is opened successfully.
  2. Includes error handling for cases like "The file is empty" or "Error opening the file."
* **Process Command:**

1. This block processes each line of the file and determines the type of SQL operation (like CREATE, INSERT, SELECT, etc.) to execute:
2. **CREATE:** Handles the creation of database tables based on the provided SQL command. It is called the createTable() function.
3. **DATABASES:** This command uses the file system library to output the file path.
4. **INSERT INTO:** Inserts new customer’s data into the table. This is the insertIntoTable() function.
5. **SELECT \* FROM:** Queries data from the table and retrieves it using the selectFromTable() function.
6. **TABLES:** Output the name of the table.
7. **UPDATE:** Updates existing customers’ records in the table with the updateFromTable() function.
8. **DELETE:** Deletes the records from the table connected to the deleteFromTable() function.
9. **COUNT:** Counts the number of rows in a table or performs aggregations. This is the countFromTable() function.

* **Output File:**

1. This block writes the results of the processed commands into output files in both TXT and CSV formats.
2. **Display in txt file:** Writes the data to a text file format.
3. **Display table in csv file:** Exports the table data in a CSV format for easy readability and compatibility with spreadsheet applications.

### Program 2: TC7L\_G03\_main2.cpp



### Explanation of Program 2 Structure Diagram

**Overall Representation**

* **Title**: The diagram is labelled "Sample 3," indicating that it represents a specific example focused on managing book data.

**Main Function**

* **Main Function Block**: This is where everything starts. It starts the program, handling file operations and processing commands.

**Key Components**

1. **Read File**:
   1. This part shows that the program is set to open and read a file that the user specifies. The file contains SQL commands for managing book records.
   2. If the file opens correctly and isn’t empty, the program gets to work interpreting the SQL commands inside.
2. **Process Commands**:
   1. The program looks at the commands it just read and sorts them based on their type:
      1. **CREATE**: This command indicates that a new table structure is being created to hold book data. It calls the createTable() function.
      2. **INSERT INTO**: This is all about adding new books to the collection and triggers the insertIntoTable() function.
      3. **SELECT FROM**: This prepares to fetch book data using the selectFromTable() function.
      4. **UPDATE**: This is a future feature for updating book records, linked to the updateFromTable() function.
      5. **DELETE**: Similarly, this one will handle removing books, connected to deleteFromTable().
      6. **COUNT**: This is for counting how many records are present, planned for the countFromTable() function.
3. **Book Storage**:
   1. **Books Vector**: This is where the program keeps track of the book data while it’s running. It uses a vector to store all the book objects that are added.
4. **Output Results**:
   1. After processing the commands, the program prepares to display results. If there’s any data, it writes it out to files in a format, like CSV, that’s easy to read.

# Input

## Sample 1

A close-up of a computer screen

Description automatically generated

## Sample 2

A close-up of a text

Description automatically generated

## Sample 3

A screenshot of a computer

Description automatically generated

# Output

## Sample 1

A computer screen shot of a computer screen

Description automatically generated

### Explanation of fileOutput1.txt

The commands in fileOutput1.txt represent a simple database operation for managing customer data.

**Command Breakdown**

1. **CREATE fileOutput1.txt;**
   1. This command indicates the intention to create a new output file named fileOutput1.txt. This is where results from the subsequent operations will be stored.
2. **DATABASES;**
   1. This statement suggests a command to switch or reference a database. It indicates the program is targeting a specific database using file system library.
3. **CREATE TABLE customer(...);**
   1. This command creates a new table named customer in the database. The table structure includes several fields to store information about customers:
      1. customer\_id INT: An integer field for the customer's ID.
      2. customer\_name TEXT: A text field for the customer's name.
      3. customer\_city TEXT: A text field for the customer's city.
      4. customer\_state TEXT: A text field for the customer's state.
      5. customer\_country TEXT: A text field for the customer's country.
      6. customer\_phone TEXT: A text field for the customer's phone number.
      7. customer\_email TEXT: A text field for the customer's email address.
4. **TABLES;**
   1. This command is possibly intended to list existing tables in the current database. It signifies an action related to the database structure.
5. **INSERT INTO customer(...);**
   1. These commands add new customer records into the customer table. Each INSERT statement includes the customer ID and related information for each customer:
      1. Customers 1 through 4 are added with corresponding details like names, cities, states, countries, phone numbers, and email addresses.
6. **SELECT \* FROM customer;**
   1. This command retrieves all records from the customer table. It selects every column for each customer entry available in the table.

### fileOutput1.csv

The final database will also be output as a csv file.

A screenshot of a spreadsheet

Description automatically generated

## Sample 2

A close-up of a computer screen

Description automatically generatedA screenshot of a computer code

Description automatically generated

### Explanation of fileOutput2.txt

The commands in fileOutput2.txt illustrate a more complex set of operations for managing customer data. This file not only creates records but also updates and deletes them while demonstrating how to gather data from the customer table

**Command Breakdown**

1. **CREATE fileOutput2.txt;**
   1. This command signals the creation of a new output file called fileOutput2.txt, where results of the subsequent operations will be stored.
2. **DATABASES;**
   1. Similar to the previous file, this command indicates a context switch to a specific database located at C:\mariadb\fileInput2.mdb, setting the stage for table management.
3. **CREATE TABLE customer(...);**
   1. The creation of a new table named customer is defined here. It has the same structure as in the previous file, including fields for:
      1. customer\_id INT
      2. customer\_name TEXT
      3. customer\_city TEXT
      4. customer\_state TEXT
      5. customer\_country TEXT
      6. customer\_phone TEXT
      7. customer\_email TEXT
4. **INSERT INTO customer(...);**
   1. Multiple INSERT statements are used to add new customer records to the table. Each record is populated with details for customers 1 through 4, just like in the previous operations.
5. **SELECT \* FROM customer;**
   1. This command fetches all records from the customer table, displaying them. The output would look like this:
6. Customer ID, Customer Name, Customer City, Customer State, Customer Country, Customer Phone, Customer Email
7. 1,name1,city1,state1,country1,phone1,email1
8. 2,name2,city2,state2,country2,phone2,email2
9. 3,name3,city3,state3,country3,phone3,email3
10. 4,name4,city4,state4,country4,phone4,email4
11. **TABLES;**
    1. This command is presumably intended to list existing tables in the database.
12. **UPDATE customer SET customer\_email='email333' WHERE customer\_id=3;**
    1. This command updates the email address of the customer with customer\_id 3 to 'email333'. This demonstrates how to modify existing records in the database.
13. **SELECT \* FROM customer;**
    1. After the update, this command retrieves all records from the customer table again, allowing us to see the changes made. The output would reflect that customer 3 now has the updated email.
14. **DELETE FROM customer WHERE customer\_id=4;**
    1. This command deletes the record of the customer with customer\_id 4 from the table, demonstrating how to remove entries.
15. **SELECT \* FROM customer;**
    1. This retrieves all remaining records, showing the updated list of customers after the deletion.
16. **SELECT COUNT(\*) FROM customer;**
    1. This command counts the total number of records currently in the customer table. The output provides a simple integer indicating how many customers remain after the previous operations.

### fileOutput2.csv

The final database will be output as a csv file.

A screenshot of a computer

Description automatically generated

## Sample 3

### Output of Sample 3 to command line

A screen shot of a computer program

Description automatically generated

### Explanation of Sample Output 3

The commands processed in the context of the book database demonstrate a series of operations for managing book records within a database, showcasing table creation, data manipulation, and output generation.

**Command Breakdown**

1. **Filename Entry:**  
    The user enters the filename, indicating that the program will work with fileInput3.mdb.
2. **Table Creation:**  
    The program successfully processes a command to create a table, initializing the structure for storing book data.
3. **Displaying Existing Tables:**  
    The program shows the available tables in the database, confirming that the books table is ready to use.
4. **Book Records Display:**  
    A list of all books stored in the books table is displayed, providing detailed information for each entry, including ID, title, author, publication year, genre, availability status, quantity, price, language, and location. This gives a clear overview of the current inventory.
5. **Update Notification:**  
    The program lets the user know that the status of the book with ID 5 (The Hobbit) has been successfully changed to "Available," showing that records can be modified when needed.
6. **Post-Update Book Records Display:**  
    After the update, the program provides the complete list of books again, reflecting the changes, with ID 5 now showing as "Available."
7. **Deletion Notification:**  
    The program announces the successful deletion of the book with ID 6 (The Da Vinci Code) and updates the user on the new total count of available books, which is now 11.
8. **Final Book Records Display:**  
    The remaining books are listed again, confirming that ID 6 is no longer present and showing the updated inventory.
9. **Final Book Count:**  
    The program outputs the final count of books in the books table, indicating there are 11 books remaining after the deletion operation.
10. **CSV File Generation Notification:**  
     Finally, the program confirms that a CSV file containing the details of the books was successfully created, allowing for easy data export and management.

### fileOutput3.csv

**A screenshot of a computer

Description automatically generated**

# Code segments explanation

In this assignment, we have created two programs.

Program 1 named TC7L\_G03\_main1.cpp is to read the input files from sample 1 and 2 and write to output files.

Program 2 name TC7L\_G03\_main2.cpp is to read the input files from sample 3 and write to output files.

## Program 1: TC7L\_G03\_main1.cpp

1. **Header files**

|  |
| --- |
|  |
| Explanation: The header files included at the beginning of the program are essential for accessing various functionalities required throughout the code. |

1. **Function prototype**

|  |
| --- |
|  |
| Explanation: Function prototypes declare the functions that will be defined later in the code. This allows the main function and other parts of the code to be aware of the functions' signatures (names, return types, and parameters) before they are actually implemented, ensuring proper functioning and preventing conflicts. |

1. **Main function**

|  |
| --- |
| A computer code with text  Description automatically generated with medium confidence |
| Explanation: Vector that stores the data of the customers like customer id, customer name, customer city, customer state, customer country, customer phone and customer email. Initialized customer count to 0. |
| A close up of words  Description automatically generated |
| Explanation: **ifstream** is a class used to read file with the variable named **inputFile**; **ofstream** is a class used to write data to a file, **outputCsvFile** representing a file stream to write the file to a CSV file and **outputTxtFile** is to write it to a TXT file; **string** is a class to handle text and **filename** is a variable to hold the name of the file (e.g. “fileInput1.mdb”) to be opened and processed; **line** is a varibale to use for finding the keywords to process command and **fullCommand** used to complete command built from other strings or input. |
| A white background with black dots  Description automatically generated |
| Explaination: The program starts by prompting the user to enter the filename of the input file containing SQL-like commands. It opens the file for reading and if the filename is “fileInput1.mdb”, it will open CSV and TXT file for output 1. Otherwise, it will open both file for output 2. |
| A screenshot of a computer code  Description automatically generated |
| Explaination: This checks if it is empty. This allows for immediate feedback to the user if there are issues with file accessibility or content. |
| A screenshot of a computer code  Description automatically generated  A screenshot of a computer code  Description automatically generated  A white background with black dots  Description automatically generated |
| Explaination: The provided code processes SQL-like commands from an input file line by line, constructing a complete command until a semicolon (;) indicates the end. It checks the command type, calling the appropriate functions such as createTable, insertIntoTable, selectFromTable, updateFromTable, deleteFromTable, or countFromTable to execute various database operations based on the command syntax. Other than that, it handles DATABASES commands by printing file path from user device by using file system library to both command line and TXT file. Additionally, it handles TABLES commands by printing the table name. After processing each complete command, the fullCommand string is cleared to prepare for the next command, promoting modular code design for efficient command handling. The input file close after the commands ended. |
| A screenshot of a computer program  Description automatically generated |
| Explaination: The code checks whether there are any customers available by evaluating the value of customer\_count. If customer\_count is zero, it outputs a message indicating that there are no customers to view in CSV format. Otherwise, it iterates through the customer data using a for loop, writing each customer's details (such as ID, name, city, state, country, phone, and email) into a CSV file. The first row of the CSV file contains the column headers. After processing all customer records, the code closes both the CSV file (outputCsvFile) and a text file (outputTxtFile). |

1. **createTable function**

|  |
| --- |
| A white background with black text  Description automatically generated |
| Explanation: This will create the table to store the information from the database. |

1. **selectFromTable function**

|  |
| --- |
| A screen shot of a computer  Description automatically generated |
| Explanation: When this function is called, the table will be output to the terminal and write to the output file. |

1. **insertIntoTable function**

|  |
| --- |
| A white screen with text  Description automatically generated |
| **Explanation:**  **Parameter:** All arguments are passed by reference.  **Idea:** Used the library <regex> to extract the data from the command “VALUES” in the input file, and then write into output files.  **Functions template used in regex:**   1. **regex pattern:** R(...) = raw string literals, it helps to ignore all special characters like \n \t   **Format:** R(“(), ' () '”)   * put () around the regex, to create a capturing group(grouped them together). * having open and close brackets () to store vector * if have single quotation (' '), means the vector stored is a string * Inside the (), we use a series of pattern to detect spaces, open and closed bracket(), quotation(‘) and to capture INT and TEXT to be stored in vector: * \( : to detect the open bracket '(' * \d : to detect digit number 0-9. In code, now we have \d+ because we need to capture one or more digit like customer id. * [^']+ : to capture TEXT inside a single quotation. In this program, we do not need to output single quotation, so we put ^' to detect and ignore it. The plus sign (+) here means one or more character(string) * \s\*: to detect space  1. **smatch:** string match  * create an object called match and then use store the result of the matched regex above.  1. **regex\_search:** Searches a string for a match to the regex  * **Format:** (first, second, third) * first = to go through the line in database * second = hold the results of the matched value * third = the pattern that you want to match with * After detecting characters, extract the data needed, and push it into a vector. |

1. **updateFromTable function**

|  |
| --- |
|  |
| Explanation: This function processes the UPDATE command and extracts the new email and customer ID to be updated using a regex. It searches for the customer ID in the customer\_id vector. If found, it updates the corresponding email to the newest one given in input file. If the ID is not found or if the format is invalid, appropriate error messages are displayed. |

1. **deleteFromTable function**

|  |
| --- |
|  |
| Explanation: This function processes the DELETE FROM command and extracts the customer ID to be deleted using a regex. It searches for the customer ID in the customer\_id vector. If found, it erases the corresponding entries from all customer-related vectors and decrements the customer count. If the ID is not found or if the format is invalid, appropriate error messages are displayed. |

1. **countFromTable function**

|  |
| --- |
|  |
| Explanation: This function processes the SELECT COUNT command and extracts information to count the number of customer records. It uses a regex to validate the command format and identify the relevant table name. If the command is valid, it outputs the customer count to both the console and the specified output file. If the command format is invalid, it displays an appropriate error message, indicating the syntax issue. |

## Program 2: TC7L\_G03\_main2.cpp

1. **Header files**

|  |
| --- |
|  |
| Explanation: The header files included at the beginning of the program are essential for accessing various functionalities required throughout the code. |

1. **Function prototype**

|  |
| --- |
|  |
| Explanation: Function prototypes declare the functions that will be defined later in the code. This allows the main function and other parts of the code to be aware of the functions' signatures (names, return types, and parameters) before they are actually implemented, ensuring proper functioning and preventing conflicts. |

1. **Main function**

|  |
| --- |
|  |
| Explanation: The program starts by prompting the user to enter the filename of the input file containing SQL-like commands. It opens the file for reading and checks if it is empty. This allows for immediate feedback to the user if there are issues with file accessibility or content. |
|  |
| **Explanation:** The provided code processes SQL-like commands from an input file line by line, constructing a complete command until a semicolon (;) indicates the end. It checks the command type, calling the appropriate functions such as createTable, insertIntoTable, selectFromTable, updateFromTable, deleteFromTable, or countBooks to execute various database operations based on the command syntax. Additionally, it handles TABLES commands by printing the table name. After processing each complete command, the fullCommand string is cleared to prepare for the next command. |
|  |
| **Explanation:** After processing the commands, the program checks if any books were successfully inserted into the vector. It then outputs the book details to both the console and a CSV file (fileOutput3.csv). Lastly, it ensures that the output file is properly closed, maintaining good resource management practices and preparing the program to exit. |

1. **createTable function**

|  |
| --- |
| A close-up of a computer screen  Description automatically generated |
| **Explanation:** The createTable function is responsible for creating a new table in memory or the database based on the defined structure provided through the input command. This function typically sets up the necessary data structures to store the table’s data, ensuring that the specified schema (fields and their types) is respected. |

1. **selectFromTable function**

|  |
| --- |
| A computer code with text  Description automatically generated |
| **Explanation:** This function retrieves and displays data from the specified table. It processes select queries, extracts the requested data based on any filters or conditions provided, and formats the output to reflect the current contents of the table. |

1. **insertIntoTable function**

|  |
| --- |
| A screen shot of a computer code  Description automatically generated |
| **Explanation:**  **Parameter:** All arguments are passed by reference.  **Idea:** Used the library <regex> to extract the data from the command “VALUES” in the input file, and then write into output files.  **Functions template used in regex:**   * + - 1. **regex pattern:** R(...) = raw string literals, it helps to ignore all special characters like \n \t   **Format:** R(“(), ' () '”)   * put () around the regex, to create a capturing group(grouped them together). * having open and close brackets () to store vector * if have single quotation (' '), means the vector stored is a string * Inside the (), we use a series of pattern to detect spaces, open and closed bracket(), quotation(‘) and to capture INT and TEXT to be stored in vector: * \( : to detect the open bracket '(' * \d : to detect digit number 0-9. In code, now we have \d+ because we need to capture one or more digit like customer id. * [^']+ : to capture TEXT inside a single quotation. In this program, we do not need to output single quotation, so we put ^' to detect and ignore it. The plus sign (+) here means one or more character(string) * \s\*: to detect space   + - 1. **smatch:** string match * create an object called match and then use store the result of the matched regex above.   + - 1. **regex\_search:** Searches a string for a match to the regex * **Format:** (first, second, third) * first = to go through the line in database * second = hold the results of the matched value * third = the pattern that you want to match with * After detecting characters, extract the data needed, and push it into a vector. |

1. **updateFromTable function**

|  |
| --- |
|  |
| Explanation: This function processes an SQL-like "UPDATE" command to modify a book\_status in a vector of Book objects. It uses a regex to extract the new status and book\_id, and updates the corresponding book. It searches for the customer ID in the customer\_id vector. If successful, it prints a success message; otherwise, it reports no matching book or invalid input format. |

1. **deleteFromTable function**

|  |
| --- |
|  |
| Explanation: This function processes the DELETE FROM command and extracts the customer ID to be deleted using a regex. It searches for the customer ID in the customer\_id vector. If found, it erases the corresponding entries from all customer-related vectors and decrements the customer count. If the ID is not found or if the format is invalid, appropriate error messages are displayed. |

1. **countFromTable function**

|  |
| --- |
| A white background with colorful text  Description automatically generated |
| Explanation: This function, countBooks, counts the number of books in a given list (vector) of Book objects. It takes the list as a constant reference, meaning it doesn’t copy or modify the original list. The function simply calls the **.size()** method of the vector, which returns the number of elements in it, and then returns this value as an integer. |

# Task Distribution

## Student 1

|  |  |
| --- | --- |
| Student ID | 243UC247BR |
| Student name | LIM PEI SHAN |
| Task percentage | 100% |
| Task descriptions | 1. Structure Diagram for sample 1 and 2 2. Create the input files 3. Output the TXT files in the same format as the sample given for sample 1 and 2 4. Create the update function |
| Total score (40m) |  |

## Student 2

|  |  |
| --- | --- |
| Student ID | 243UC247D6 |
| Student name | CHAN KER JING |
| Task percentage | 100% |
| Task descriptions | 1. Pseudocode for program 1 and 2 2. Insert rows to the table 3. View table in csv mode 4. Create vector to store the data for sample 1 and 2 |
| Total score (40m) |  |

## Student 3

|  |  |
| --- | --- |
| Student ID | 243UC247C1 |
| Student name | BERNICE GAN XINYI |
| Task percentage | 100% |
| Task descriptions | 1. Structure Diagram for Sample 3 2. Create the delete function |
| Total score (40m) |  |

## Student 4

|  |  |
| --- | --- |
| Student ID | 243UC247CB |
| Student name | TEOW YAN PING |
| Task percentage | 100% |
| Task descriptions | 1. Create vector to store the data for sample 3 2. Output the full path of the input file using file system library 3. Create the count function |
| Total score (40m) |  |