SOLUTIONS TO QUBITTECH ASSESSMENT QUESTIONS;

I have utilized visual studio code with visual studio build tools as my coding platform and developer command tool to run and test my codes for each specific question answered;

Answer to Question 2 (sql.cpp):

The code demonstrates a basic SQL Builder library in C++ that supports method chaining to construct SQL queries. Fow now, I have only written the code for the SQL builder that only generates SELECT, INSERT, and DELETE queries as show below;

```
#include <iostream>

#include <string>

#include <vector>


class SQLBuilder {

private:

   std::string query;

public:

   SQLBuilder& select(const std::string& columns) {

      query = "SELECT " + columns;

      return *this;

   }


   SQLBuilder& from(const std::string& table) {

      query += " FROM " + table;

      return *this;

   }


   SQLBuilder& where(const std::string& condition) {

      query += " WHERE " + condition;

      return *this;

   }
```

```
    SQLBuilder& insertInto(const std::string& table, const std::vector<std::string>& columns,
const std::vector<std::string>& values) {
        query = "INSERT INTO " + table + " (";
        for (size_t i = 0; i < columns.size(); ++i) {
            query += columns[i] + (i < columns.size() - 1 ? ", " : "");
        }
        query += ") VALUES (";
        for (size_t i = 0; i < values.size(); ++i) {
            query += "'" + values[i] + "'" + (i < values.size() - 1 ? ", " : "");
        }
        query += ")";
        return *this;
    }

    SQLBuilder& deleteFrom(const std::string& table) {
        query = "DELETE FROM " + table;
        return *this;
    }

    SQLBuilder& limit(int n) {
        query += " LIMIT " + std::to_string(n);
        return *this;
    }

    std::string build() const {
        return query + ";";
    }
};
```

```
int main() {
    // Example usage of the SQL Builder; some sql queries constructed are
    SQLBuilder sql;
    std::string selectQuery = sql.select("*").from("users").where("age > 18").limit(10).build();
    std::string insertQuery = sql.insertInto("users", {"name", "age"}, {"Shimbe Majestic",
"30"}).build();
    std::string deleteQuery = sql.deleteFrom("users").where("age < 18").build();

    // Output results
    std::cout << "Select Query: " << selectQuery << std::endl;
    std::cout << "Insert Query: " << insertQuery << std::endl;
    std::cout << "Delete Query: " << deleteQuery << std::endl;

    return 0;
}
```

Its Output was:

```
C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools>sql.exe
Select Query: SELECT * FROM users WHERE age > 18 LIMIT 10;
Insert Query: INSERT INTO users (name, age) VALUES ('Shimbe Majestic', '30');
Delete Query: DELETE FROM users WHERE age < 18;
```

Answer to Question 4 (json.cpp);

The code creates a JSON parser and creator using C++. It allows you to create JSON objects, parse them, and retrieve values:

```
#include <iostream>
#include <string>
#include <unordered_map>
#include <sstream>
```

```
class JSON {
private:
    std::unordered_map<std::string, std::string> data;
public:
    void add(const std::string& key, const std::string& value) {
        data[key] = value;
    }

    std::string get(const std::string& key) const {
        auto it = data.find(key);
        return it != data.end() ? it->second : "";
    }

    std::string toString() const {
        std::ostringstream oss;
        oss << "{ ";
        for (auto it = data.begin(); it != data.end(); ++it) {
            oss << "\"" << it->first << "\": \"" << it->second << "\"";
            if (std::next(it) != data.end()) oss << ", ";
        }
        oss << " }";
        return oss.str();
    }

    static JSON parse(const std::string& jsonStr) {
        JSON json;
        std::istringstream iss(jsonStr);
```

```
        std::string key, value;
        char ch;
        while (iss >> ch) {
            if (ch == '\"') {
                getline(iss, key, '\"');
                iss >> ch >> ch;
                getline(iss, value, '\"');
                json.add(key, value);
            }
        }
        return json;
    }
};

int main() {
    // Create JSON object
    JSON json;
    json.add("name", "Shimbe Majestic");
    json.add("age", "25");

    // Print JSON string
    std::cout << "JSON Object: " << json.toString() << std::endl;

    // Parse JSON string
    std::string jsonStr = "{\"city\": \"Dar es salaam\", \"country\": \"Tanzania\"}";
    JSON parsedJson = JSON::parse(jsonStr);
    std::cout << "Parsed JSON - City: " << parsedJson.get("city") << std::endl;
    std::cout << "Parsed JSON - Country: " << parsedJson.get("country") << std::endl;
```

```
    return 0;
}
```

Its Output was;

```
C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools>json.exe
JSON Object: { "name": "Shimbe Majestic", "age": "25" }
Parsed JSON - City: Dar es salaam
Parsed JSON - Country: Tanzania
```

Answer to Question 5 (logfile.cpp);

The code below processes a log file line-by-line and stores each line in a database (simulated here with a simple array). It uses multithreading to speed up processing large log files.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <thread>
#include <mutex>

std::mutex dbMutex;  // Mutex for thread safety
std::vector<std::string> database;  // Simulated database

// Function to process log line
void processLine(const std::string& line) {
    std::lock_guard<std::mutex> lock(dbMutex);
    database.push_back(line);  // Simulate storing in database
}
```

```
// Function to read and process log file
void processLogFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Error opening file." << std::endl;
        return;
    }

    std::string line;
    std::vector<std::thread> threads;
    while (std::getline(file, line)) {
        threads.emplace_back(processLine, line);  // Start a thread for each line
    }

    // Join threads to ensure they complete before continuing
    for (auto& t : threads) {
        t.join();
    }
}

int main() {
    // Simulated log file processing
    processLogFile("sample_log.txt");

    // Output processed log entries
    std::cout << "Processed Log Entries: " << std::endl;
    for (const auto& entry : database) {
        std::cout << entry << std::endl;
```

```
    }

    return 0;
}
```

Its Output was;

```
C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools>logfile.exe
Processed Log Entries:
Hello there, My name is Shimbe Majestic and i am applying for the post of software developer at Qubittech
```