# Diagnostic Test Tracker

Emily Zheng
2025/12/10
CIS25 Fall 2025

## Background

**Purpose**: Database for tracking patient and diagnostic test information at a clinic

# Data Tracking

- Data Storage Components
    - Track patient information in Patients table
        - Patient ID - starts at 1 for the first entry and increments by +1 for every additional entry
        - First Name
        - Last Name
        - Date of Birth
    - Track diagnostic test information in Results table
        - Test ID - starts at 1 for the first entry and increments by +1 for every additional entry
        - Test Name - name of the diagnostic test (Example: Covid 19)
        - Output - expects an integer or decimal number
            - 0 for negative and 1 for positive for binary tests
        - Units - string where user can input if the test was binary or can input the units the quantitative results are measured in (Example: mg/mL)
        - Patient ID - will reference the Patient ID column of the Patients table

## Data Analysis

$$\text{Average} = \frac{\text{Sum of all values}}{\text{Number of values}} = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n}$$

- Analyze diagnostic tests with binary outputs
  - Obtain the Positive and Negative rate for a test out of all the tests entered into Results table
    - Positive rate = (Count of Positive Instances for Test of Interest) / (Total Instances for Test of Interest) * 100
    - Negative Rate = 100 - Positive Rate
    - Positive and Negative rates are in the unit of percent
- Analyze diagnostic tests with quantitative outputs
  - Obtain average value of the output of the test
    - Average = (sum of all output values) / (Number of output values)
  - Obtain standard deviation of the output of the test
    - Standard deviation calculations follow the sample deviation equation for sample
    - Standard deviation tells us generally how far apart the data points are spread relative to the average
      - In other words, it tells us how much variation there is for the test results within the tests in the Results table

**Sample**

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

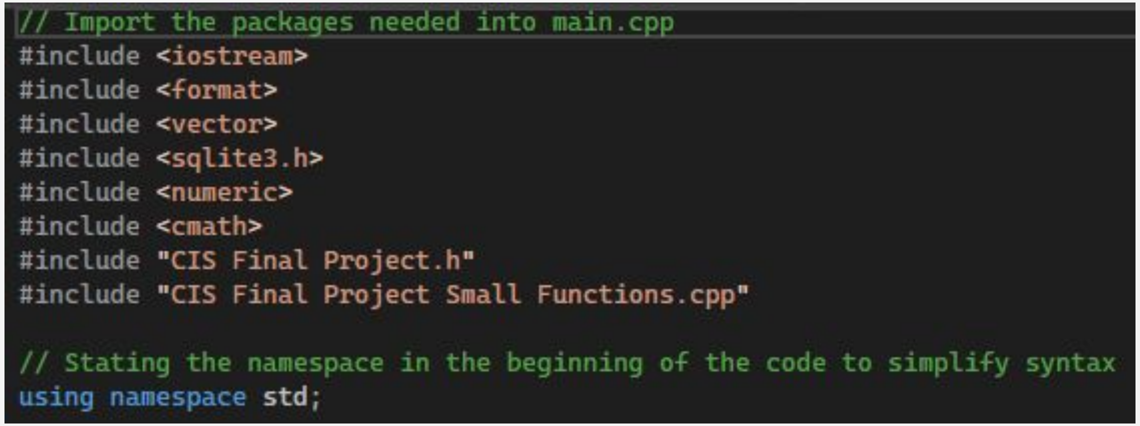
$x_i$ = elements in sample

$\bar{x}$ = sample mean

$n$ = sample size

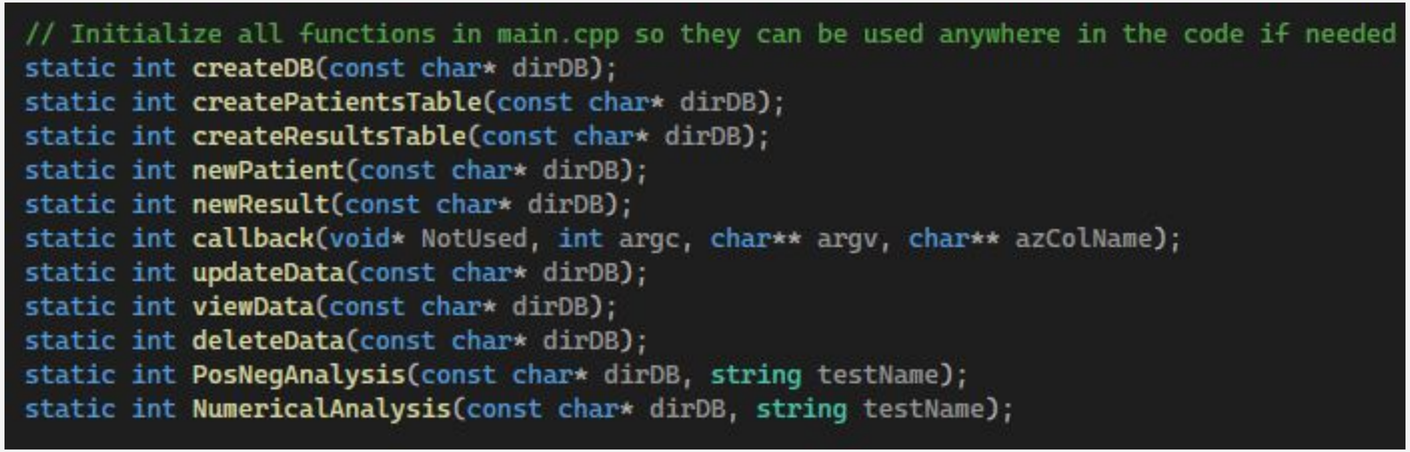
## Components

## Imports



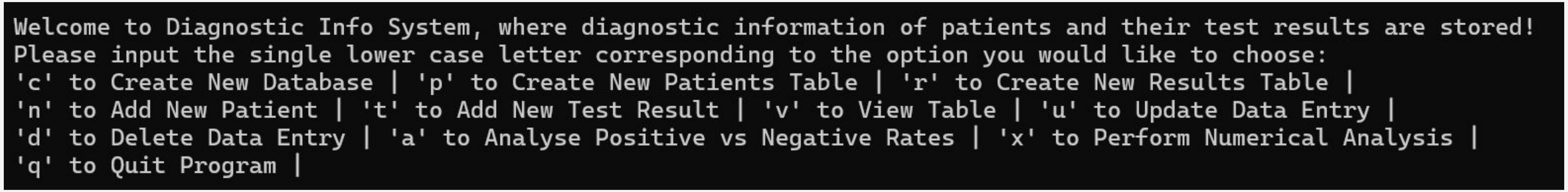
```
// Import the packages needed into main.cpp
#include <iostream>
#include <format>
#include <vector>
#include <sqlite3.h>
#include <numeric>
#include <cmath>
#include "CIS Final Project.h"
#include "CIS Final Project Small Functions.cpp"

// Stating the namespace in the beginning of the code to simplify syntax
using namespace std;
```

# Initialize Functions



```
// Initialize all functions in main.cpp so they can be used anywhere in the code if needed
static int createDB(const char* dirDB);
static int createPatientsTable(const char* dirDB);
static int createResultsTable(const char* dirDB);
static int newPatient(const char* dirDB);
static int newResult(const char* dirDB);
static int callback(void* NotUsed, int argc, char** argv, char** azColName);
static int updateData(const char* dirDB);
static int viewData(const char* dirDB);
static int deleteData(const char* dirDB);
static int PosNegAnalysis(const char* dirDB, string testName);
static int NumericalAnalysis(const char* dirDB, string testName);
```

# Menu



```
Welcome to Diagnostic Info System, where diagnostic information of patients and their test results are stored!
Please input the single lower case letter corresponding to the option you would like to choose:
'c' to Create New Database | 'p' to Create New Patients Table | 'r' to Create New Results Table |
'n' to Add New Patient | 't' to Add New Test Result | 'v' to View Table | 'u' to Update Data Entry |
'd' to Delete Data Entry | 'a' to Analyse Positive vs Negative Rates | 'x' to Perform Numerical Analysis |
'q' to Quit Program |
```

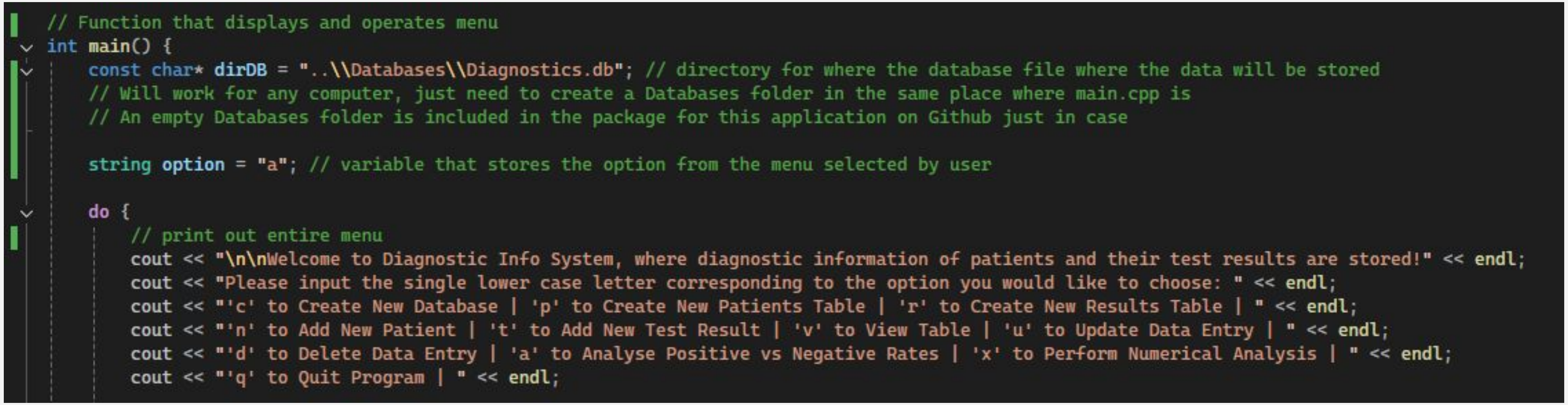
User Interface

- Upon starting the application, this menu pops up.
- The user will enter a lower case letter to select one of the menu items
- Menu displays every time an action is finished executing
- User exits the menu and application by selecting ‘q’

Methods and Tools

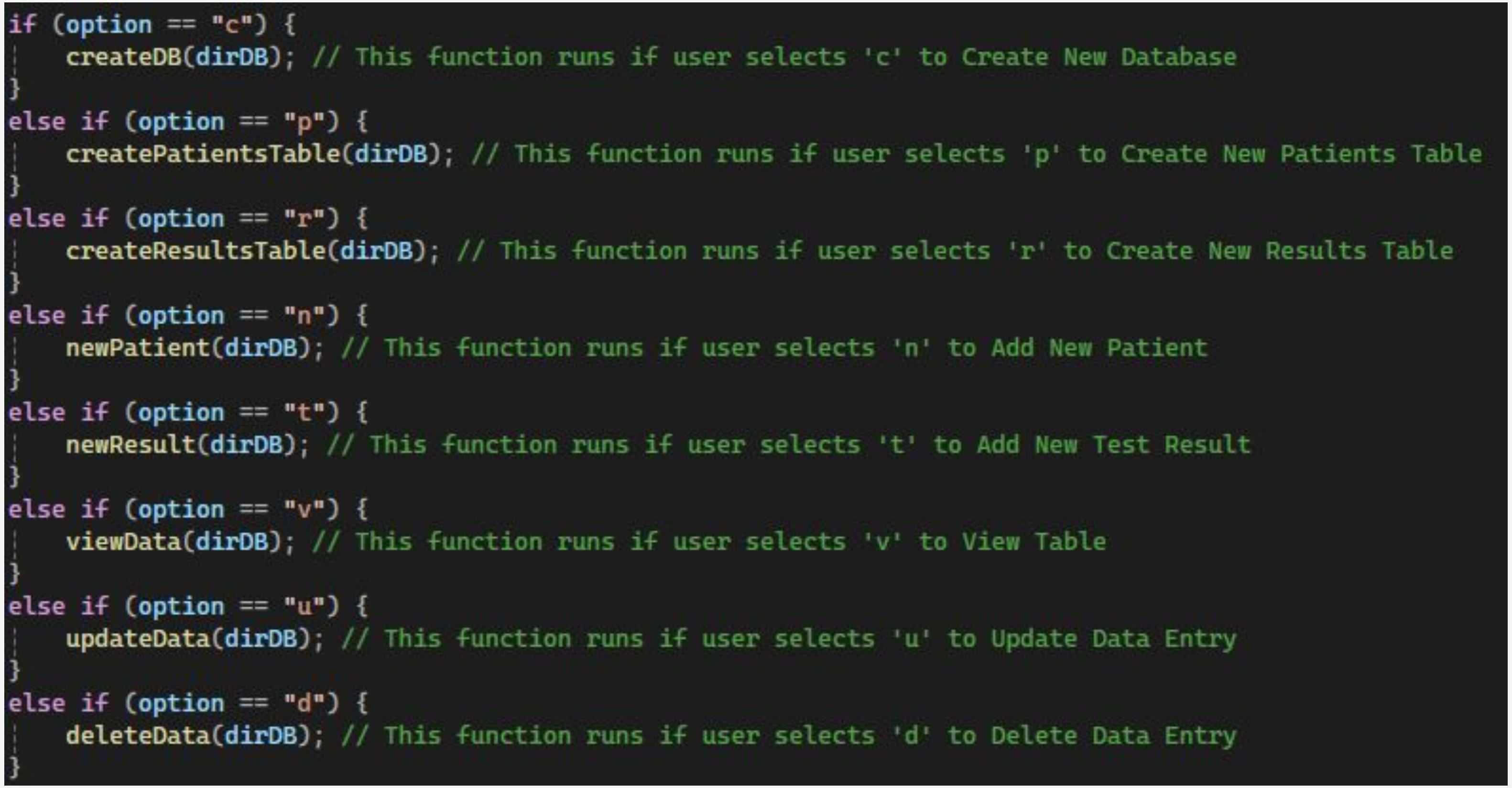
- Do While Loop
- If Else Statements
- Output and Input with cin and cout

# Menu - Code



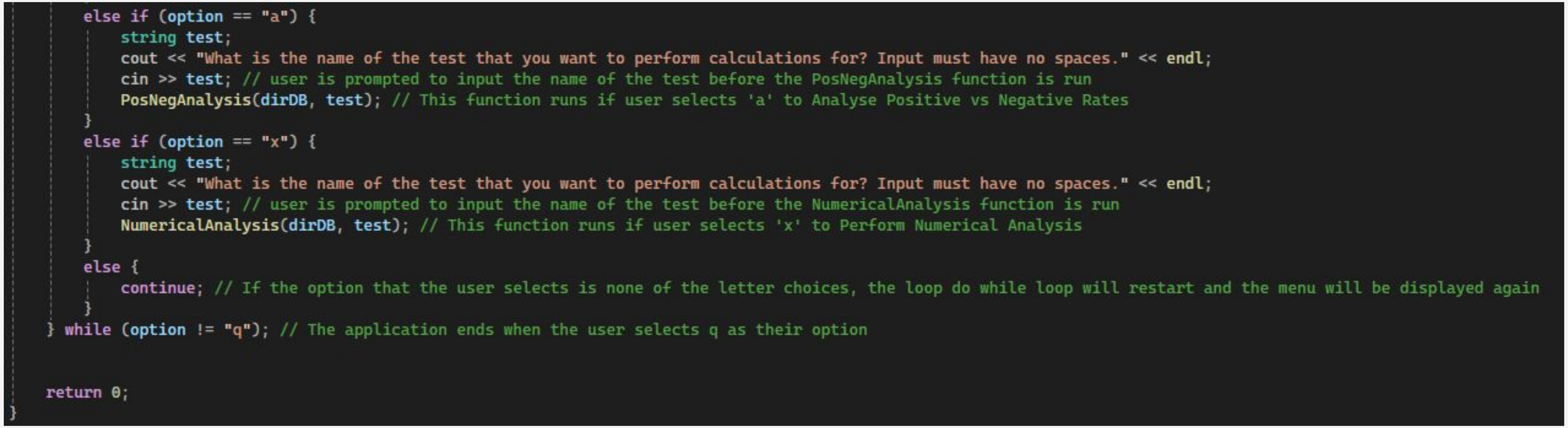
```
// Function that displays and operates menu
int main() {
    const char* dirDB = "..\\Databases\\Diagnostics.db"; // directory for where the database file where the data will be stored
    // Will work for any computer, just need to create a Databases folder in the same place where main.cpp is
    // An empty Databases folder is included in the package for this application on Github just in case

    string option = "a"; // variable that stores the option from the menu selected by user

    do {
        // print out entire menu
        cout << "\n\nWelcome to Diagnostic Info System, where diagnostic information of patients and their test results are stored!" << endl;
        cout << "Please input the single lower case letter corresponding to the option you would like to choose: " << endl;
        cout << "'c' to Create New Database | 'p' to Create New Patients Table | 'r' to Create New Results Table | " << endl;
        cout << "'n' to Add New Patient | 't' to Add New Test Result | 'v' to View Table | 'u' to Update Data Entry | " << endl;
        cout << "'d' to Delete Data Entry | 'a' to Analyse Positive vs Negative Rates | 'x' to Perform Numerical Analysis | " << endl;
        cout << "'q' to Quit Program | " << endl;
```

# Menu Code Continued



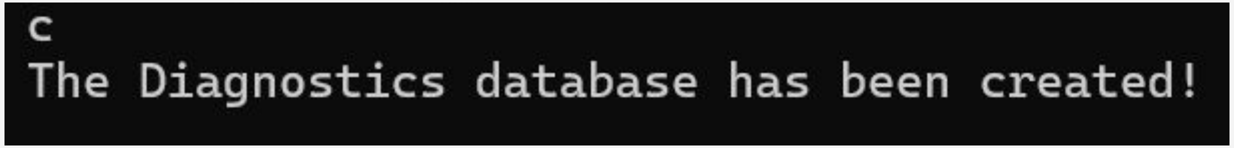
```
if (option == "c") {
    createDB(dirDB); // This function runs if user selects 'c' to Create New Database
}
else if (option == "p") {
    createPatientsTable(dirDB); // This function runs if user selects 'p' to Create New Patients Table
}
else if (option == "r") {
    createResultsTable(dirDB); // This function runs if user selects 'r' to Create New Results Table
}
else if (option == "n") {
    newPatient(dirDB); // This function runs if user selects 'n' to Add New Patient
}
else if (option == "t") {
    newResult(dirDB); // This function runs if user selects 't' to Add New Test Result
}
else if (option == "v") {
    viewData(dirDB); // This function runs if user selects 'v' to View Table
}
else if (option == "u") {
    updateData(dirDB); // This function runs if user selects 'u' to Update Data Entry
}
else if (option == "d") {
    deleteData(dirDB); // This function runs if user selects 'd' to Delete Data Entry
}
```

# Menu Code Continued



```
        else if (option == "a") {
            string test;
            cout << "What is the name of the test that you want to perform calculations for? Input must have no spaces." << endl;
            cin >> test; // user is prompted to input the name of the test before the PosNegAnalysis function is run
            PosNegAnalysis(dirDB, test); // This function runs if user selects 'a' to Analyse Positive vs Negative Rates
        }
        else if (option == "x") {
            string test;
            cout << "What is the name of the test that you want to perform calculations for? Input must have no spaces." << endl;
            cin >> test; // user is prompted to input the name of the test before the NumericalAnalysis function is run
            NumericalAnalysis(dirDB, test); // This function runs if user selects 'x' to Perform Numerical Analysis
        }
        else {
            continue; // If the option that the user selects is none of the letter choices, the loop do while loop will restart and the menu will be displayed again
        }
    } while (option != "q"); // The application ends when the user selects q as their option


    return 0;
}
```

# Create Database



```
c
The Diagnostics database has been created!
```

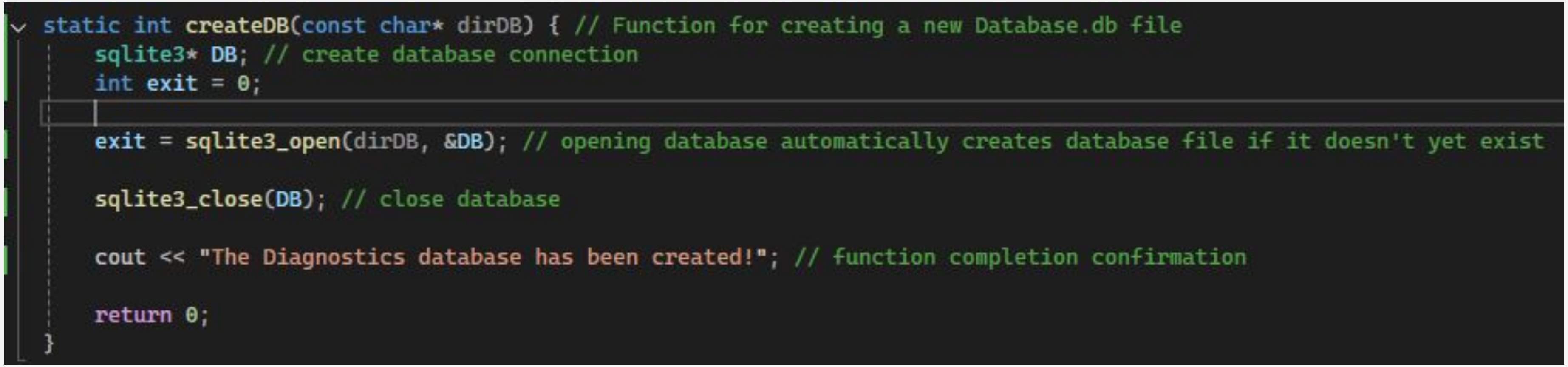
User Interface

- The user inputs ‘c’ after the menu appears
- A database called “Diagnostics.db” is created in the user’s directory
- Confirmation is printed

Methods and Tools

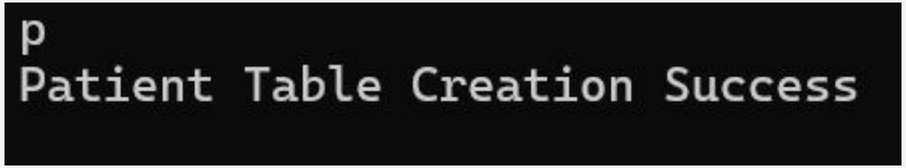
- Sqlite3 Library
- Output cout

## Create Database - Code



```
static int createDB(const char* dirDB) { // Function for creating a new Database.db file
    sqlite3* DB; // create database connection
    int exit = 0;

    exit = sqlite3_open(dirDB, &DB); // opening database automatically creates database file if it doesn't yet exist

    sqlite3_close(DB); // close database

    cout << "The Diagnostics database has been created!"; // function completion confirmation

    return 0;
}
```

# Create Patient Table



```
p
Patient Table Creation Success
```

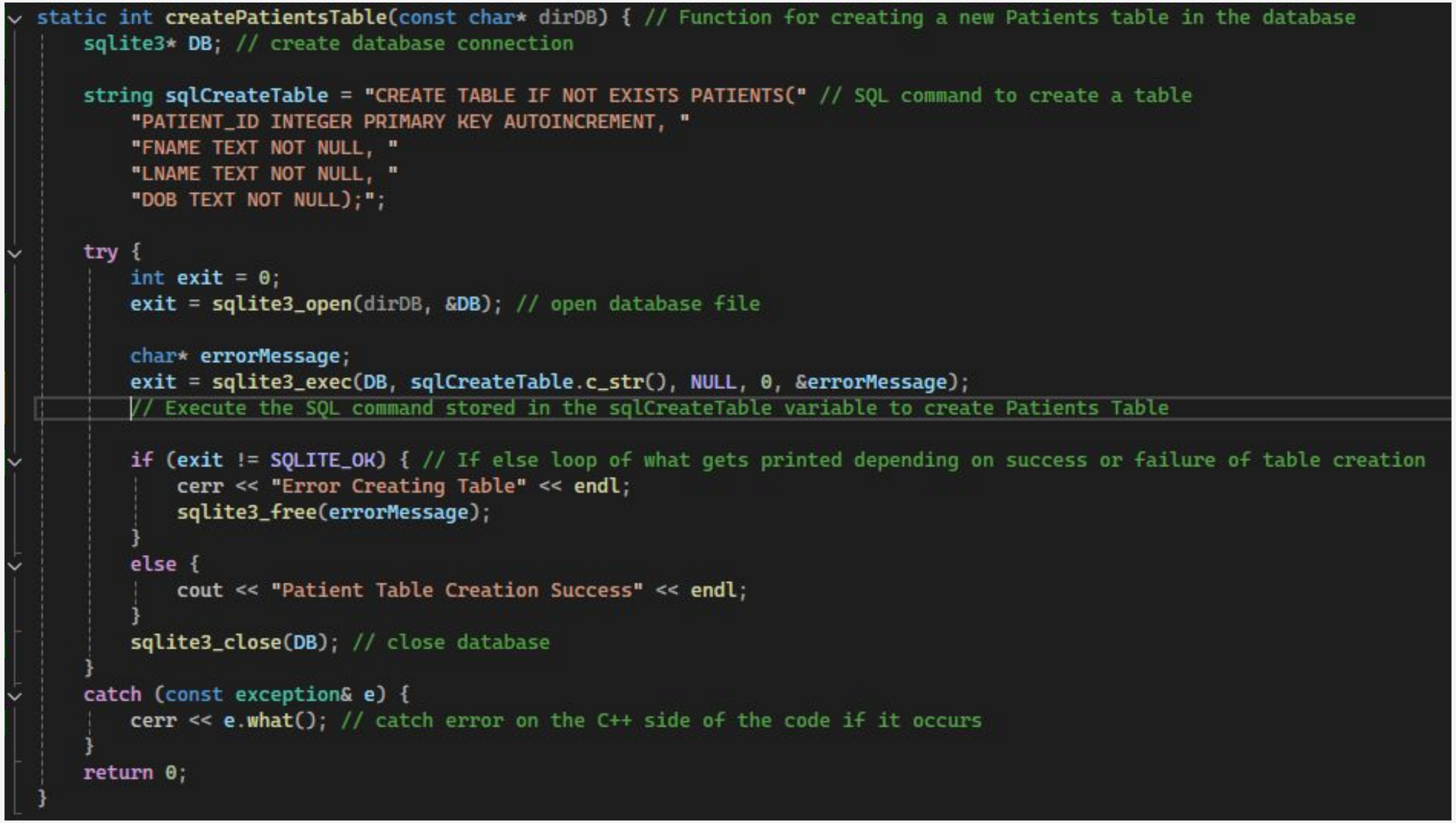
User Interface

- User inputs ‘p’ after the menu is displayed
- A table called Patients is created within the Diagnostics.db file
- Message is printed to indicate Success or Error in table creation

Methods and Tools

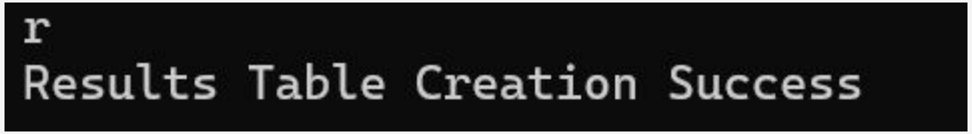
- Sqlite3 Library
- Try Catch Block
- SQL Table Creation
- If Else Statement

# Create Patient Table - Code



```cpp
static int createPatientsTable(const char* dirDB) { // Function for creating a new Patients table in the database
    sqlite3* DB; // create database connection

    string sqlCreateTable = "CREATE TABLE IF NOT EXISTS PATIENTS(" // SQL command to create a table
        "PATIENT_ID INTEGER PRIMARY KEY AUTOINCREMENT, "
        "FNAME TEXT NOT NULL, "
        "LNAME TEXT NOT NULL, "
        "DOB TEXT NOT NULL);";

    try {
        int exit = 0;
        exit = sqlite3_open(dirDB, &DB); // open database file

        char* errorMessage;
        exit = sqlite3_exec(DB, sqlCreateTable.c_str(), NULL, 0, &errorMessage);
        // Execute the SQL command stored in the sqlCreateTable variable to create Patients Table

        if (exit != SQLITE_OK) { // If else loop of what gets printed depending on success or failure of table creation
            cerr << "Error Creating Table" << endl;
            sqlite3_free(errorMessage);
        }
        else {
            cout << "Patient Table Creation Success" << endl;
        }
        sqlite3_close(DB); // close database
    }
    catch (const exception& e) {
        cerr << e.what(); // catch error on the C++ side of the code if it occurs
    }
    return 0;
}
```

## Create Results Table



```
r
Results Table Creation Success
```

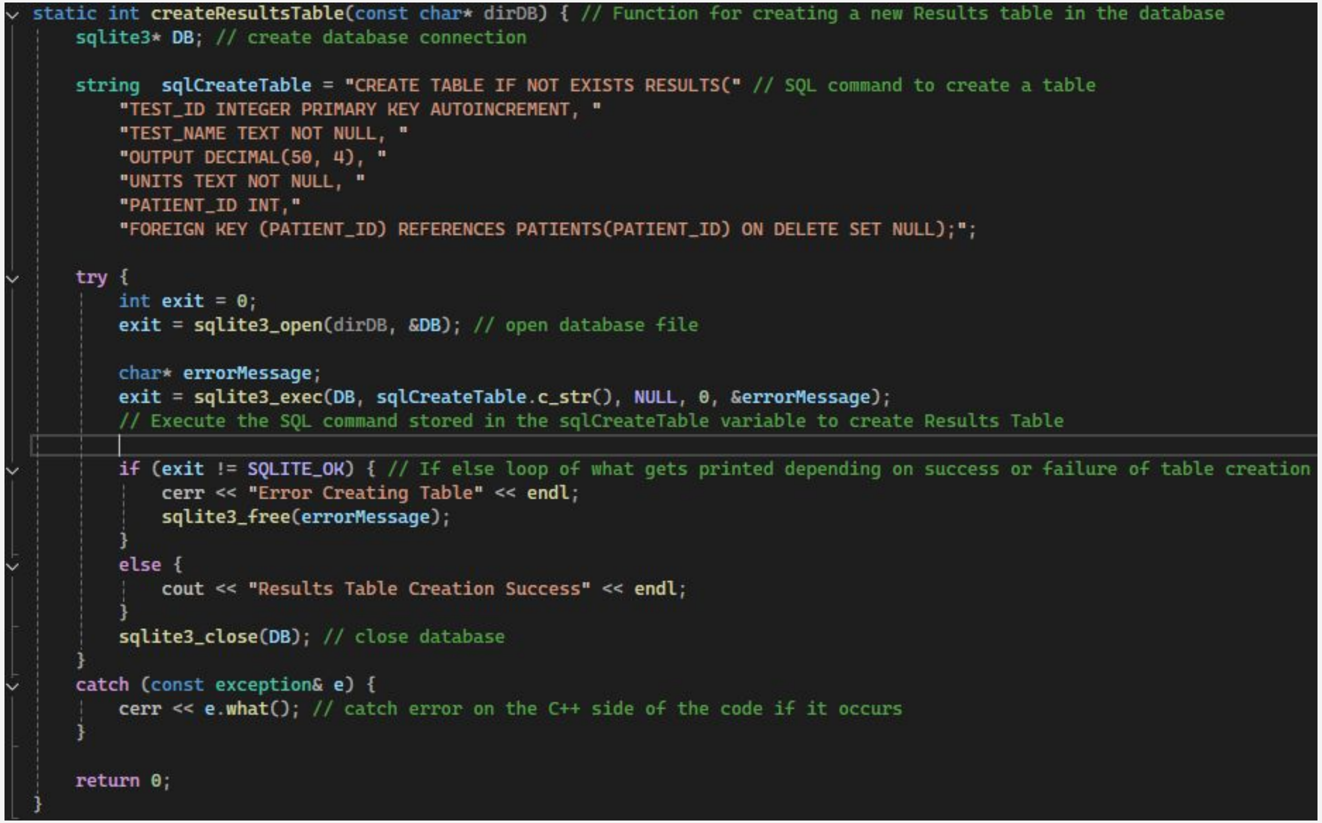
User Interface

- User inputs ‘r’ after the menu is displayed
- A table called Results is created within the Diagnostics.db file
- Message is printed to indicate Success or Error in table creation

Methods and Tools

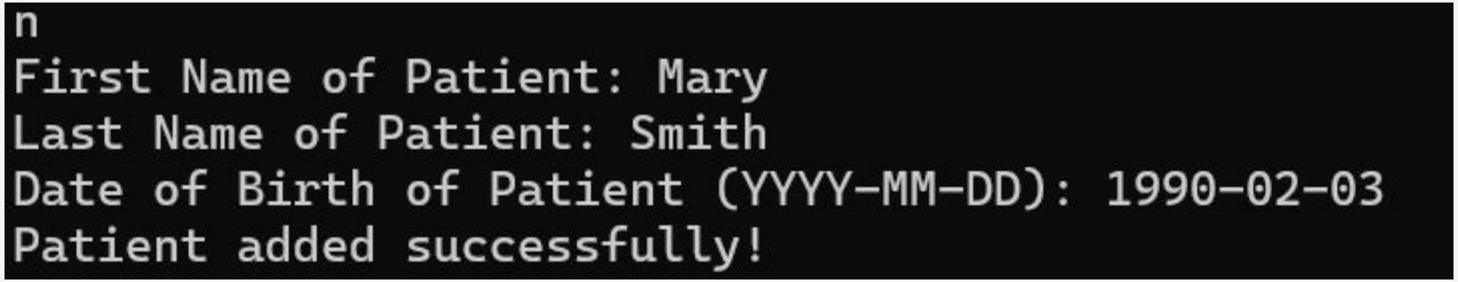
- Sqlite3 Library
- Try Catch Block
- SQL Table Creation
- If Else Statement

# Create Results Table - Code



```cpp
static int createResultsTable(const char* dirDB) { // Function for creating a new Results table in the database
    sqlite3* DB; // create database connection

    string  sqlCreateTable = "CREATE TABLE IF NOT EXISTS RESULTS(" // SQL command to create a table
        "TEST_ID INTEGER PRIMARY KEY AUTOINCREMENT, "
        "TEST_NAME TEXT NOT NULL, "
        "OUTPUT DECIMAL(50, 4), "
        "UNITS TEXT NOT NULL, "
        "PATIENT_ID INT,"
        "FOREIGN KEY (PATIENT_ID) REFERENCES PATIENTS(PATIENT_ID) ON DELETE SET NULL);";

    try {
        int exit = 0;
        exit = sqlite3_open(dirDB, &DB); // open database file

        char* errorMessage;
        exit = sqlite3_exec(DB, sqlCreateTable.c_str(), NULL, 0, &errorMessage);
        // Execute the SQL command stored in the sqlCreateTable variable to create Results Table

        if (exit != SQLITE_OK) { // If else loop of what gets printed depending on success or failure of table creation
            cerr << "Error Creating Table" << endl;
            sqlite3_free(errorMessage);
        }
        else {
            cout << "Results Table Creation Success" << endl;
        }
        sqlite3_close(DB); // close database
    }
    catch (const exception& e) {
        cerr << e.what(); // catch error on the C++ side of the code if it occurs
    }

    return 0;
}
```

# Add Patient



```
n
First Name of Patient: Mary
Last Name of Patient: Smith
Date of Birth of Patient (YYYY-MM-DD): 1990-02-03
Patient added successfully!
```

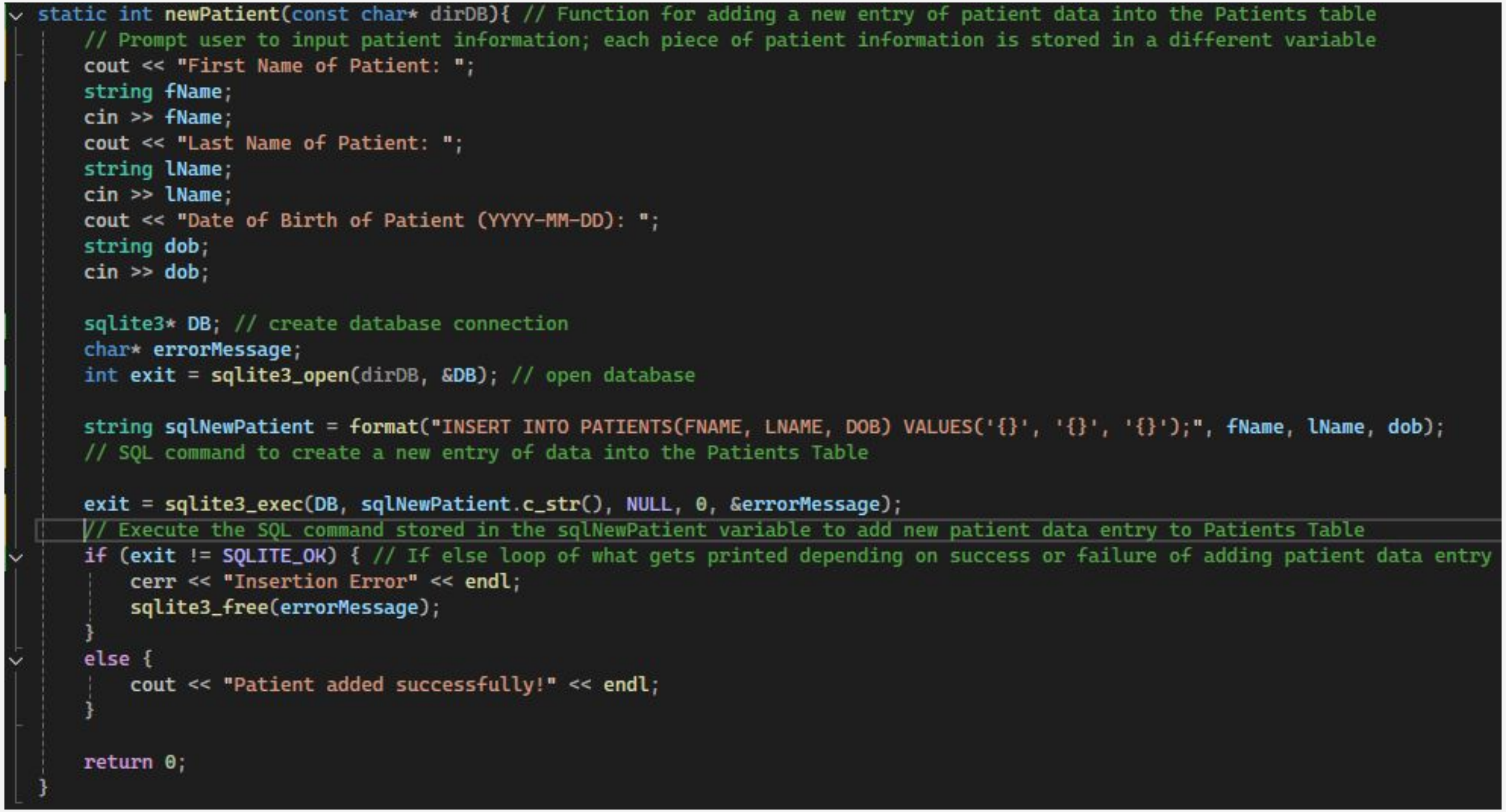
User Interface

- User inputs ‘n’ after the menu is displayed
- The User is prompted to enter the first name, last name, and date of birth of the patient
- A success message is output after the entry is created in the Patients table.

Methods and Tools

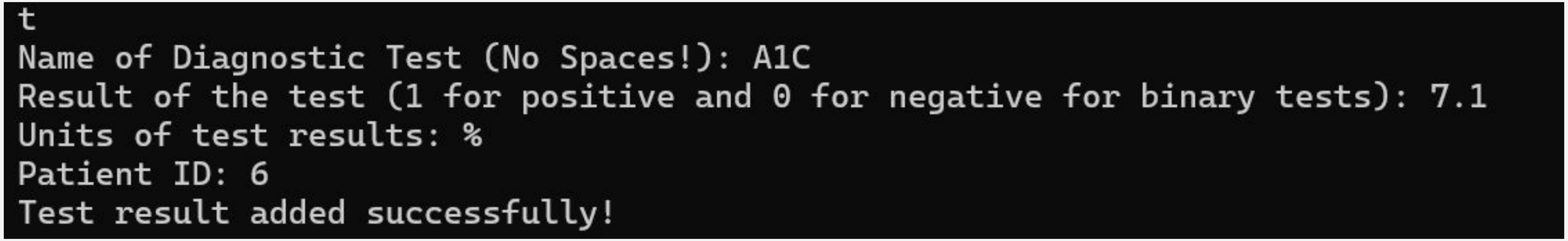
- Sqlite Library
- Input and Output with cin and cout
- SQL insertion command
- If Else Statement

# Add Patient - Code



```cpp
static int newPatient(const char* dirDB){ // Function for adding a new entry of patient data into the Patients table
    // Prompt user to input patient information; each piece of patient information is stored in a different variable
    cout << "First Name of Patient: ";
    string fName;
    cin >> fName;
    cout << "Last Name of Patient: ";
    string lName;
    cin >> lName;
    cout << "Date of Birth of Patient (YYYY-MM-DD): ";
    string dob;
    cin >> dob;

    sqlite3* DB; // create database connection
    char* errorMessage;
    int exit = sqlite3_open(dirDB, &DB); // open database

    string sqlNewPatient = format("INSERT INTO PATIENTS(FNAME, LNAME, DOB) VALUES('{}', '{}', '{}');", fName, lName, dob);
    // SQL command to create a new entry of data into the Patients Table

    exit = sqlite3_exec(DB, sqlNewPatient.c_str(), NULL, 0, &errorMessage);
    // Execute the SQL command stored in the sqlNewPatient variable to add new patient data entry to Patients Table
    if (exit != SQLITE_OK) { // If else loop of what gets printed depending on success or failure of adding patient data entry
        cerr << "Insertion Error" << endl;
        sqlite3_free(errorMessage);
    }
    else {
        cout << "Patient added successfully!" << endl;
    }

    return 0;
}
```

## Add Test Result



```
t
Name of Diagnostic Test (No Spaces!): A1C
Result of the test (1 for positive and 0 for negative for binary tests): 7.1
Units of test results: %
Patient ID: 6
Test result added successfully!
```

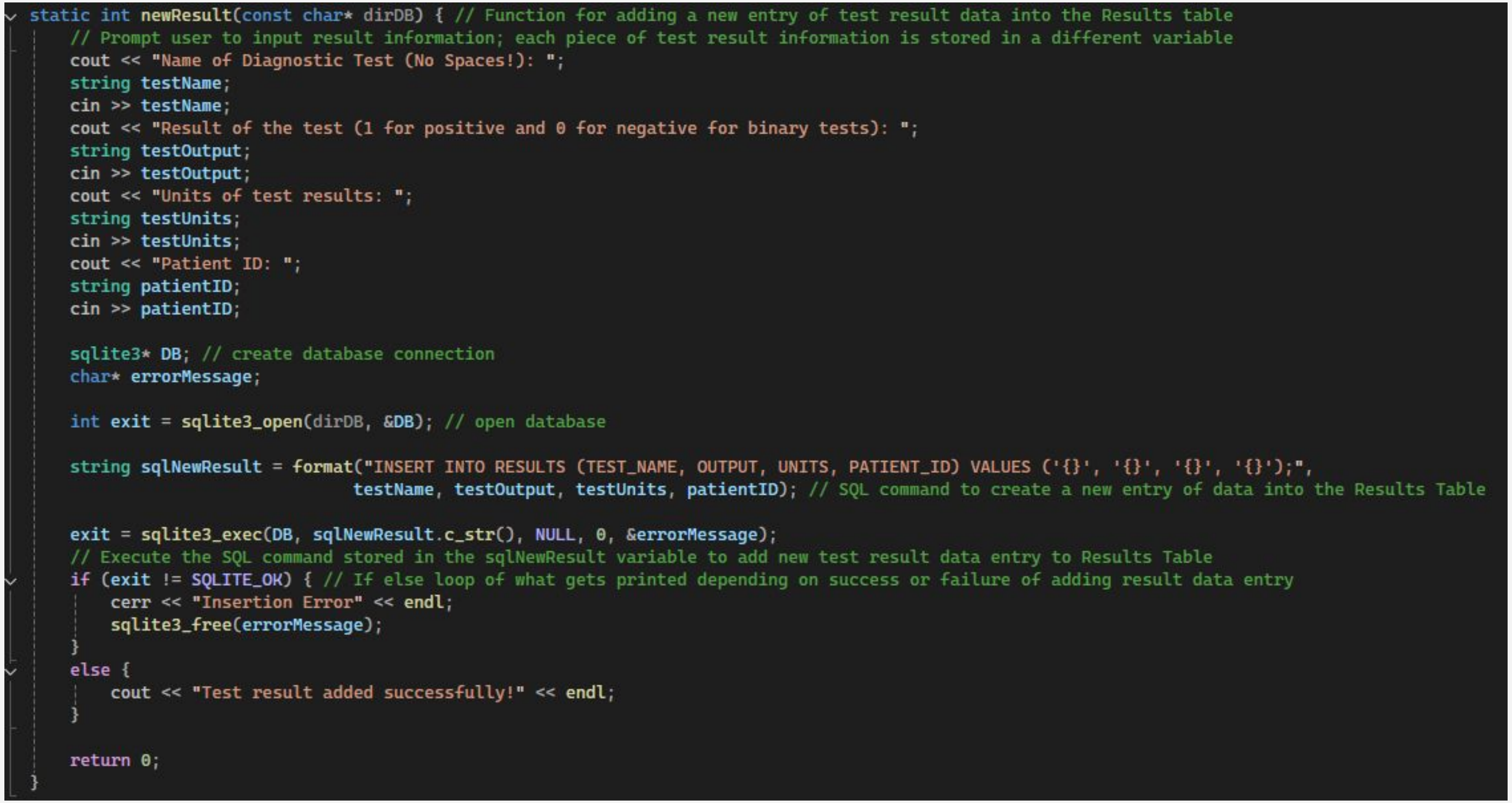
User Interface

- User inputs ‘t’ after the menu is displayed
- The User is prompted to enter the test name, test result, result units, and patient ID
- A success message is output after the entry is created in the Results table

Methods and Tools

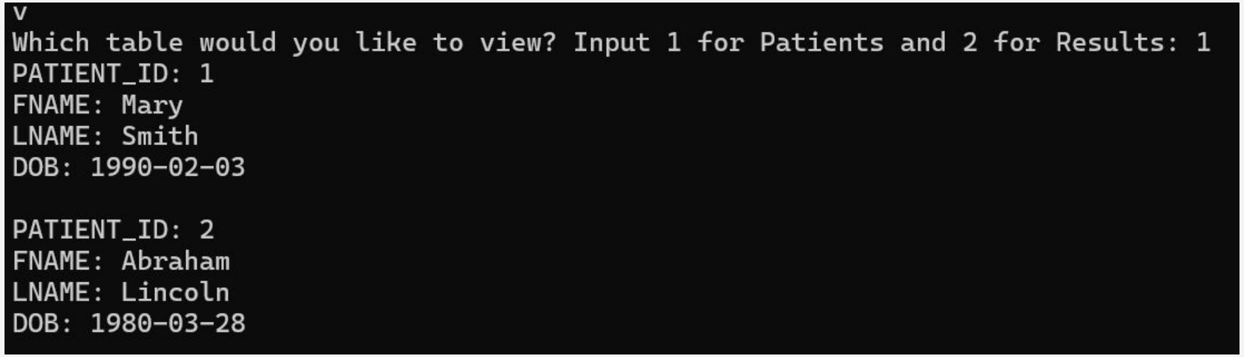
- Sqlite Library
- Input and Output with cin and cout
- SQL insertion command
- If Else Statement

# Add Test Result - Code



```cpp
static int newResult(const char* dirDB) { // Function for adding a new entry of test result data into the Results table
    // Prompt user to input result information; each piece of test result information is stored in a different variable
    cout << "Name of Diagnostic Test (No Spaces!): ";
    string testName;
    cin >> testName;
    cout << "Result of the test (1 for positive and 0 for negative for binary tests): ";
    string testOutput;
    cin >> testOutput;
    cout << "Units of test results: ";
    string testUnits;
    cin >> testUnits;
    cout << "Patient ID: ";
    string patientID;
    cin >> patientID;

    sqlite3* DB; // create database connection
    char* errorMessage;

    int exit = sqlite3_open(dirDB, &DB); // open database

    string sqlNewResult = format("INSERT INTO RESULTS (TEST_NAME, OUTPUT, UNITS, PATIENT_ID) VALUES ('{}', '{}', '{}', '{}');",
                                testName, testOutput, testUnits, patientID); // SQL command to create a new entry of data into the Results Table

    exit = sqlite3_exec(DB, sqlNewResult.c_str(), NULL, 0, &errorMessage);
    // Execute the SQL command stored in the sqlNewResult variable to add new test result data entry to Results Table
    if (exit != SQLITE_OK) { // If else loop of what gets printed depending on success or failure of adding result data entry
        cerr << "Insertion Error" << endl;
        sqlite3_free(errorMessage);
    }
    else {
        cout << "Test result added successfully!" << endl;
    }

    return 0;
}
```

## View Table



```
v
Which table would you like to view? Input 1 for Patients and 2 for Results: 1
PATIENT_ID: 1
FNAME: Mary
LNAME: Smith
DOB: 1990-02-03

PATIENT_ID: 2
FNAME: Abraham
LNAME: Lincoln
DOB: 1980-03-28
```

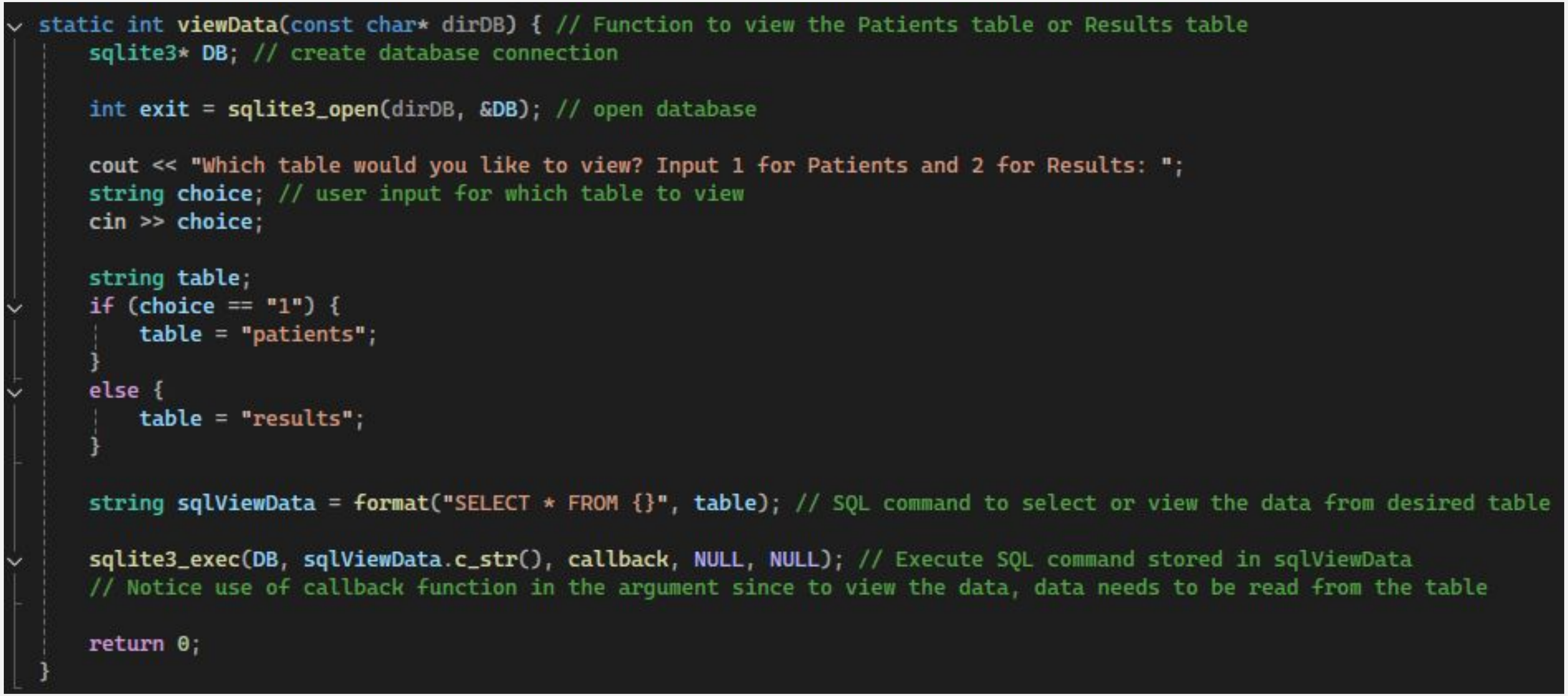
User Interface

- User inputs ‘v’ after menu is displayed
- User selects if they want to view the Patients table or the Results table
- The selected table is displayed

Methods and Tools

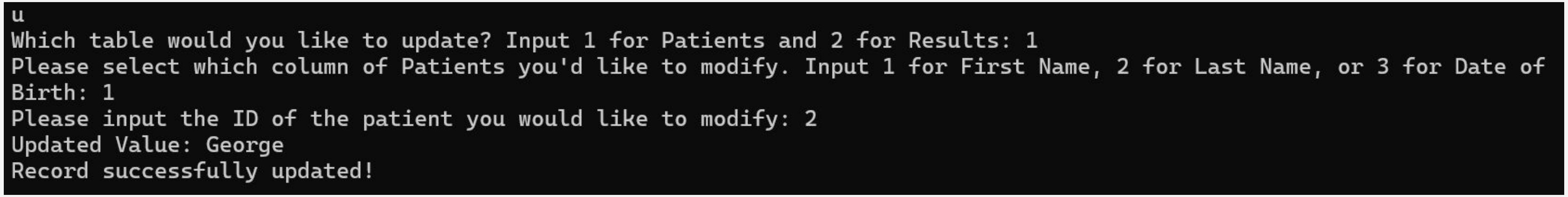
- Sqlite Library
- Input and Output with cin and cout
- If Else Statement
- Callback Function
- SQL Select Statement

# View Table - Code



```
static int viewData(const char* dirDB) { // Function to view the Patients table or Results table
    sqlite3* DB; // create database connection

    int exit = sqlite3_open(dirDB, &DB); // open database

    cout << "Which table would you like to view? Input 1 for Patients and 2 for Results: ";
    string choice; // user input for which table to view
    cin >> choice;

    string table;
    if (choice == "1") {
        table = "patients";
    }
    else {
        table = "results";
    }

    string sqlViewData = format("SELECT * FROM {}", table); // SQL command to select or view the data from desired table

    sqlite3_exec(DB, sqlViewData.c_str(), callback, NULL, NULL); // Execute SQL command stored in sqlViewData
    // Notice use of callback function in the argument since to view the data, data needs to be read from the table

    return 0;
}
```

# Update Data Entry



```
u
Which table would you like to update? Input 1 for Patients and 2 for Results: 1
Please select which column of Patients you'd like to modify. Input 1 for First Name, 2 for Last Name, or 3 for Date of
Birth: 1
Please input the ID of the patient you would like to modify: 2
Updated Value: George
Record successfully updated!
```

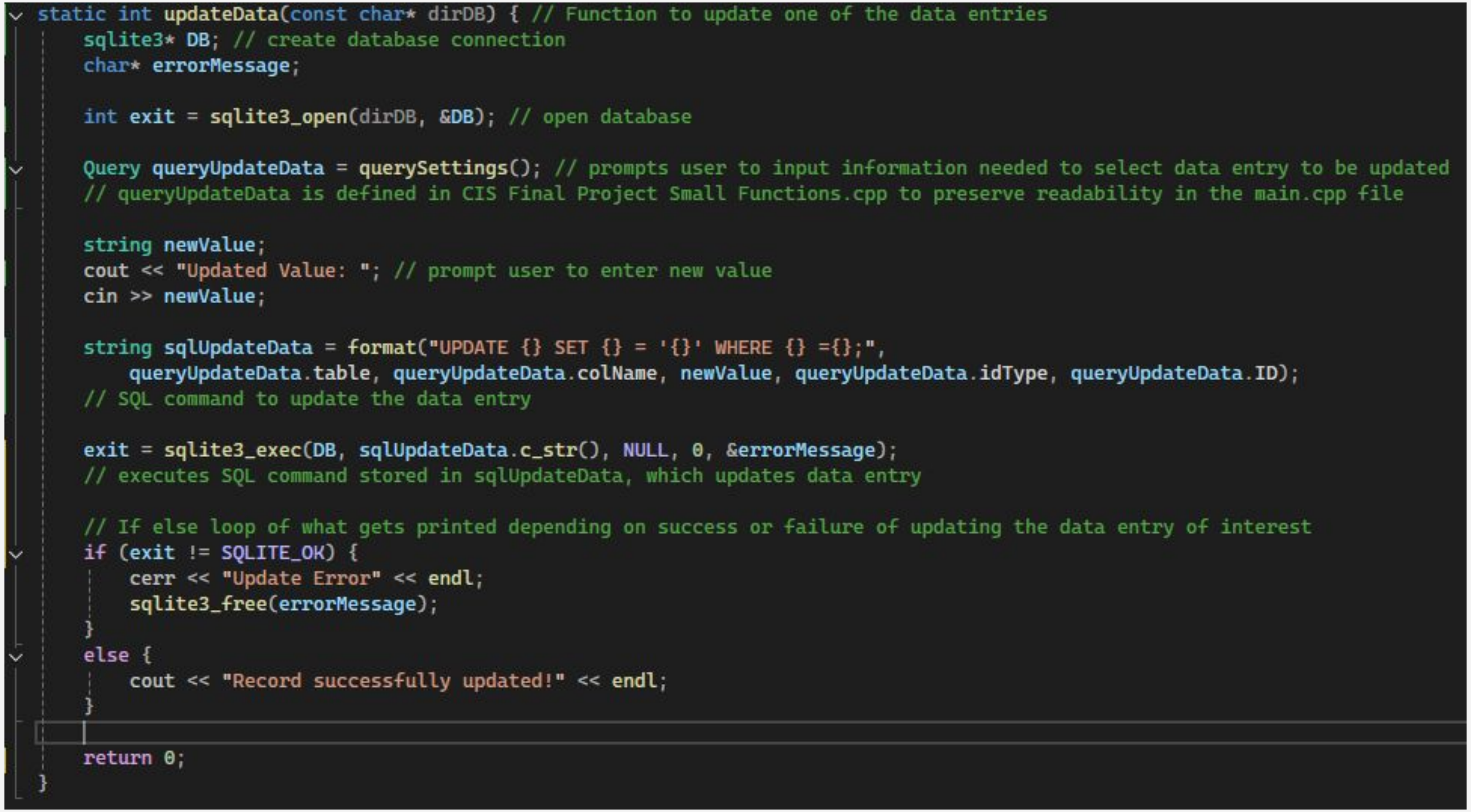
User Interface

- User inputs ‘u’ after the menu is displayed
- User selects which table the entry they want to change is in, Patients table or Results table
- User selects column of the table they want to change
- User selects Patient ID or Test ID of the entry they want to change
- User inputs new value
- Success message is printed once the entry is updated

Methods and Tools

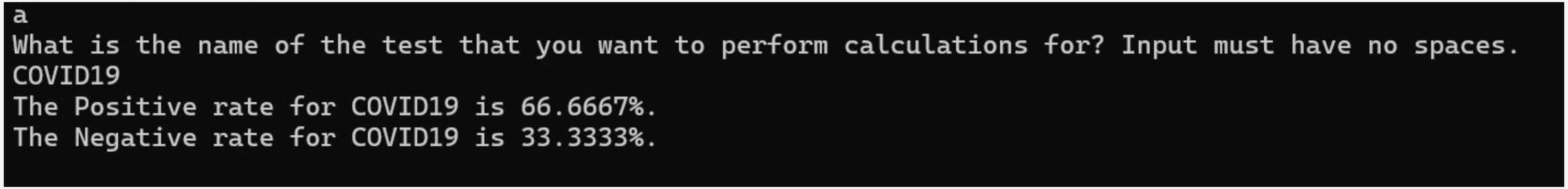
- Sqlite Library
- Input and Output with cin and cout
- If Else Statement
- SQL Update Statement

# Update Data Entry - Code



```cpp
static int updateData(const char* dirDB) { // Function to update one of the data entries
    sqlite3* DB; // create database connection
    char* errorMessage;

    int exit = sqlite3_open(dirDB, &DB); // open database

    Query queryUpdateData = querySettings(); // prompts user to input information needed to select data entry to be updated
    // queryUpdateData is defined in CIS Final Project Small Functions.cpp to preserve readability in the main.cpp file

    string newValue;
    cout << "Updated Value: "; // prompt user to enter new value
    cin >> newValue;

    string sqlUpdateData = format("UPDATE {} SET {} = '{}' WHERE {} ={};",
        queryUpdateData.table, queryUpdateData.colName, newValue, queryUpdateData.idType, queryUpdateData.ID);
    // SQL command to update the data entry

    exit = sqlite3_exec(DB, sqlUpdateData.c_str(), NULL, 0, &errorMessage);
    // executes SQL command stored in sqlUpdateData, which updates data entry

    // If else loop of what gets printed depending on success or failure of updating the data entry of interest
    if (exit != SQLITE_OK) {
        cerr << "Update Error" << endl;
        sqlite3_free(errorMessage);
    }
    else {
        cout << "Record successfully updated!" << endl;
    }

    return 0;
}
```

# Analysis of Binary Data



```
a
What is the name of the test that you want to perform calculations for? Input must have no spaces.
COVID19
The Positive rate for COVID19 is 66.6667%.
The Negative rate for COVID19 is 33.3333%.
```

User Interface

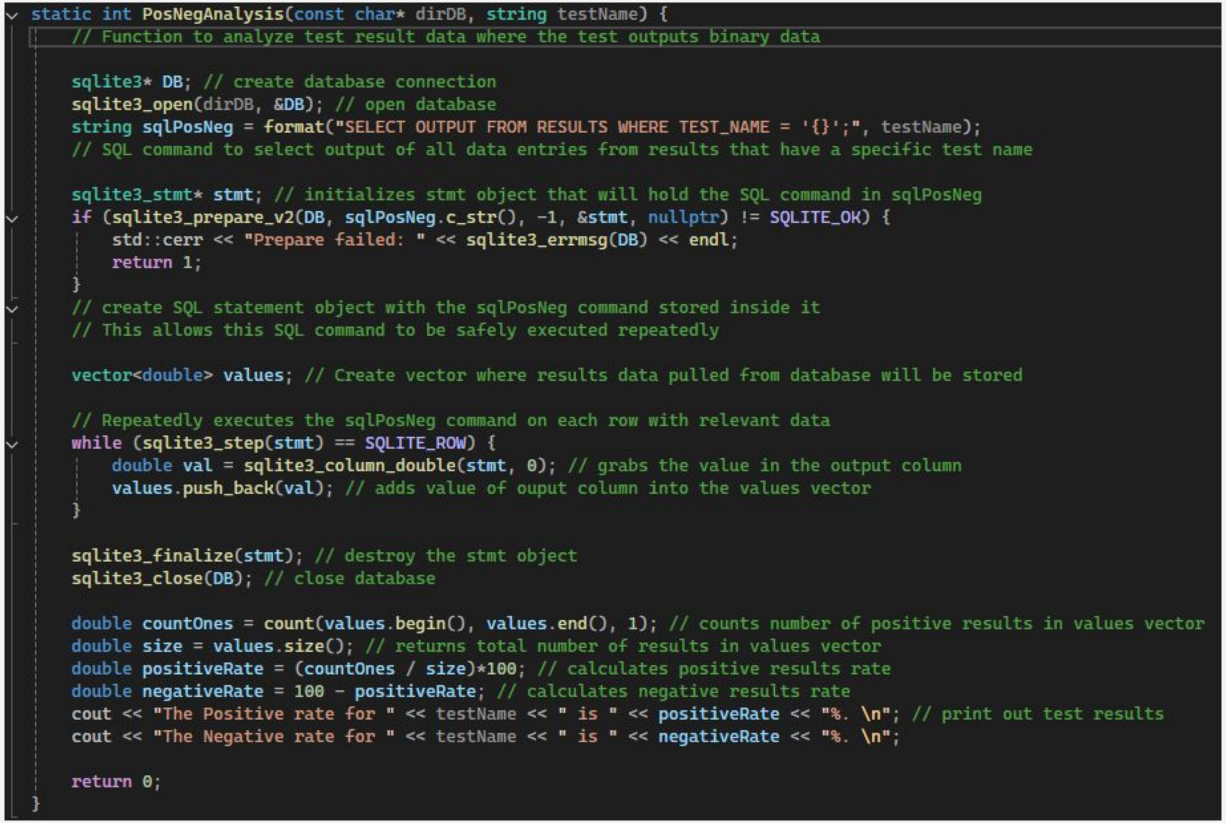
- User inputs ‘a’ after menu is displayed
- User inputs name of diagnostic test
- Outputs percentage of test results that are Negative and percentage of test results that are Positive

Methods and Tools

- Switch Statement
- Sqlite Library
- Input and Output with cin and cout
- Vectors

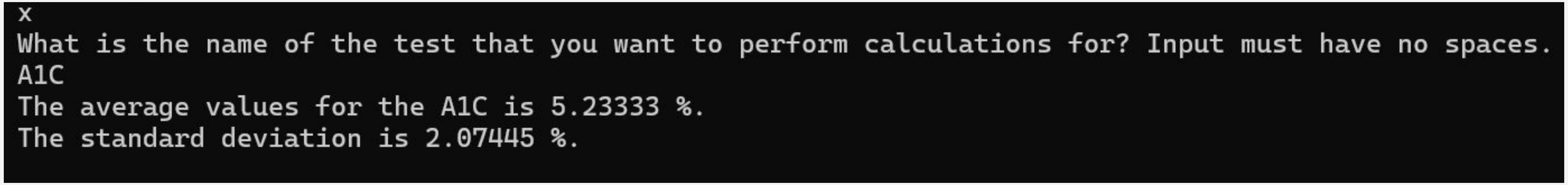
Binary Data - data with output of two forms or two classifications, in this case Positive or Negative

# Analysis of Binary Data



```cpp
static int PosNegAnalysis(const char* dirDB, string testName) {
    // Function to analyze test result data where the test outputs binary data

    sqlite3* DB; // create database connection
    sqlite3_open(dirDB, &DB); // open database
    string sqlPosNeg = format("SELECT OUTPUT FROM RESULTS WHERE TEST_NAME = '{}';", testName);
    // SQL command to select output of all data entries from results that have a specific test name

    sqlite3_stmt* stmt; // initializes stmt object that will hold the SQL command in sqlPosNeg
    if (sqlite3_prepare_v2(DB, sqlPosNeg.c_str(), -1, &stmt, nullptr) != SQLITE_OK) {
        std::cerr << "Prepare failed: " << sqlite3_errmsg(DB) << endl;
        return 1;
    }
    // create SQL statement object with the sqlPosNeg command stored inside it
    // This allows this SQL command to be safely executed repeatedly

    vector<double> values; // Create vector where results data pulled from database will be stored

    // Repeatedly executes the sqlPosNeg command on each row with relevant data
    while (sqlite3_step(stmt) == SQLITE_ROW) {
        double val = sqlite3_column_double(stmt, 0); // grabs the value in the output column
        values.push_back(val); // adds value of ouput column into the values vector
    }

    sqlite3_finalize(stmt); // destroy the stmt object
    sqlite3_close(DB); // close database

    double countOnes = count(values.begin(), values.end(), 1); // counts number of positive results in values vector
    double size = values.size(); // returns total number of results in values vector
    double positiveRate = (countOnes / size)*100; // calculates positive results rate
    double negativeRate = 100 - positiveRate; // calculates negative results rate
    cout << "The Positive rate for " << testName << " is " << positiveRate << "%. \n"; // print out test results
    cout << "The Negative rate for " << testName << " is " << negativeRate << "%. \n";

    return 0;
}
```

# Analysis of Numerical Results



```
x
What is the name of the test that you want to perform calculations for? Input must have no spaces.
A1C
The average values for the A1C is 5.23333 %.
The standard deviation is 2.07445 %.
```

User Interface

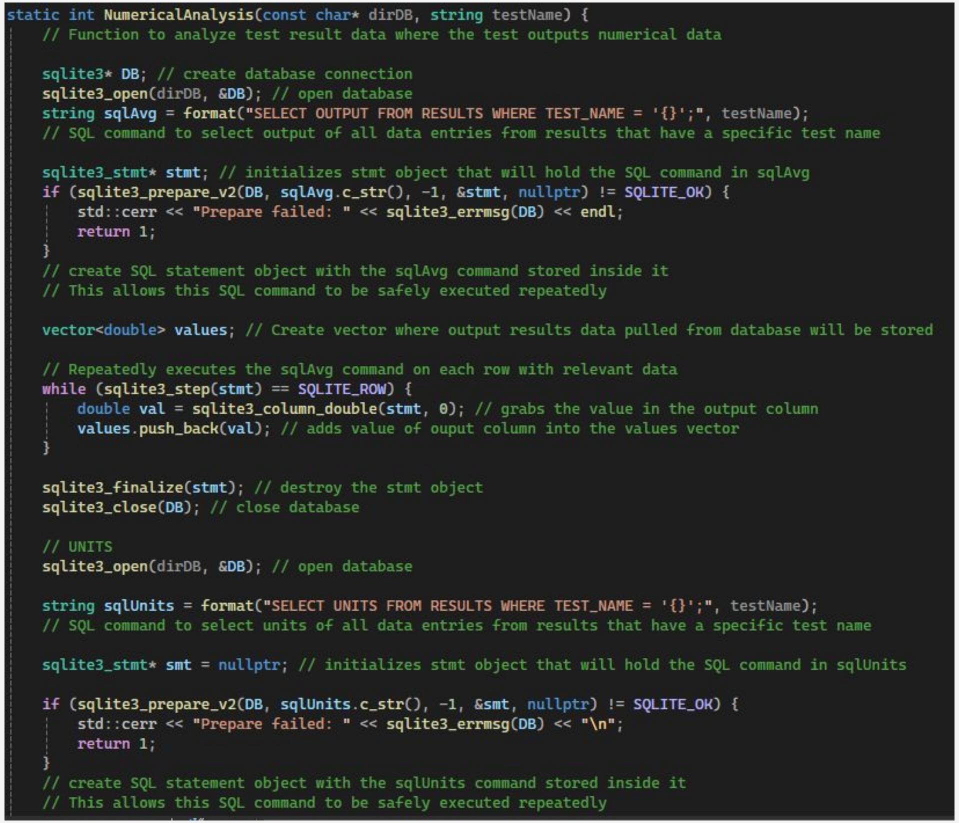
- User inputs ‘x’ after menu is displayed
- User inputs name of diagnostic test
- Outputs average and standard deviation of test results

Methods and Tools

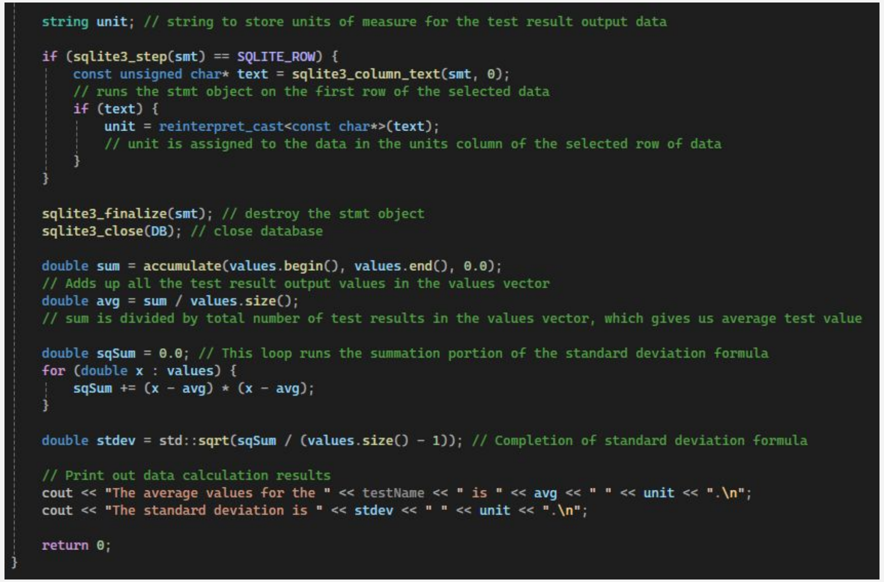
- Math Library
- Sqlite Library
- Input and Output with cin and cout
- Vectors

Numerical Results - quantitative result that is frome a continuous number scale

# Analysis of Numerical Results - Code

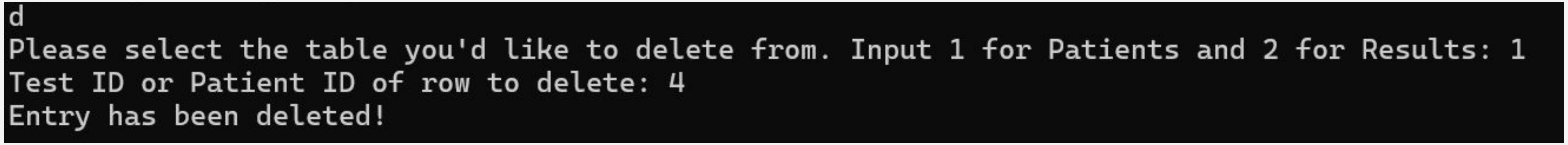


```cpp
static int NumericalAnalysis(const char* dirDB, string testName) {
    // Function to analyze test result data where the test outputs numerical data

    sqlite3* DB; // create database connection
    sqlite3_open(dirDB, &DB); // open database
    string sqlAvg = format("SELECT OUTPUT FROM RESULTS WHERE TEST_NAME = '{}';", testName);
    // SQL command to select output of all data entries from results that have a specific test name

    sqlite3_stmt* stmt; // initializes stmt object that will hold the SQL command in sqlAvg
    if (sqlite3_prepare_v2(DB, sqlAvg.c_str(), -1, &stmt, nullptr) != SQLITE_OK) {
        std::cerr << "Prepare failed: " << sqlite3_errmsg(DB) << endl;
        return 1;
    }
    // create SQL statement object with the sqlAvg command stored inside it
    // This allows this SQL command to be safely executed repeatedly

    vector<double> values; // Create vector where output results data pulled from database will be stored

    // Repeatedly executes the sqlAvg command on each row with relevant data
    while (sqlite3_step(stmt) == SQLITE_ROW) {
        double val = sqlite3_column_double(stmt, 0); // grabs the value in the output column
        values.push_back(val); // adds value of ouput column into the values vector
    }

    sqlite3_finalize(stmt); // destroy the stmt object
    sqlite3_close(DB); // close database

    // UNITS
    sqlite3_open(dirDB, &DB); // open database

    string sqlUnits = format("SELECT UNITS FROM RESULTS WHERE TEST_NAME = '{}';", testName);
    // SQL command to select units of all data entries from results that have a specific test name

    sqlite3_stmt* smt = nullptr; // initializes stmt object that will hold the SQL command in sqlUnits

    if (sqlite3_prepare_v2(DB, sqlUnits.c_str(), -1, &smt, nullptr) != SQLITE_OK) {
        std::cerr << "Prepare failed: " << sqlite3_errmsg(DB) << "\n";
        return 1;
    }
    // create SQL statement object with the sqlUnits command stored inside it
    // This allows this SQL command to be safely executed repeatedly
```



```cpp
    string unit; // string to store units of measure for the test result output data

    if (sqlite3_step(smt) == SQLITE_ROW) {
        const unsigned char* text = sqlite3_column_text(smt, 0);
        // runs the stmt object on the first row of the selected data
        if (text) {
            unit = reinterpret_cast<const char*>(text);
            // unit is assigned to the data in the units column of the selected row of data
        }
    }

    sqlite3_finalize(smt); // destroy the stmt object
    sqlite3_close(DB); // close database

    double sum = accumulate(values.begin(), values.end(), 0.0);
    // Adds up all the test result output values in the values vector
    double avg = sum / values.size();
    // sum is divided by total number of test results in the values vector, which gives us average test value

    double sqSum = 0.0; // This loop runs the summation portion of the standard deviation formula
    for (double x : values) {
        sqSum += (x - avg) * (x - avg);
    }

    double stdev = std::sqrt(sqSum / (values.size() - 1)); // Completion of standard deviation formula

    // Print out data calculation results
    cout << "The average values for the " << testName << " is " << avg << " " << unit << ".\n";
    cout << "The standard deviation is " << stdev << " " << unit << ".\n";

    return 0;
}
```

# Delete Data Entry



```
d
Please select the table you'd like to delete from. Input 1 for Patients and 2 for Results: 1
Test ID or Patient ID of row to delete: 4
Entry has been deleted!
```

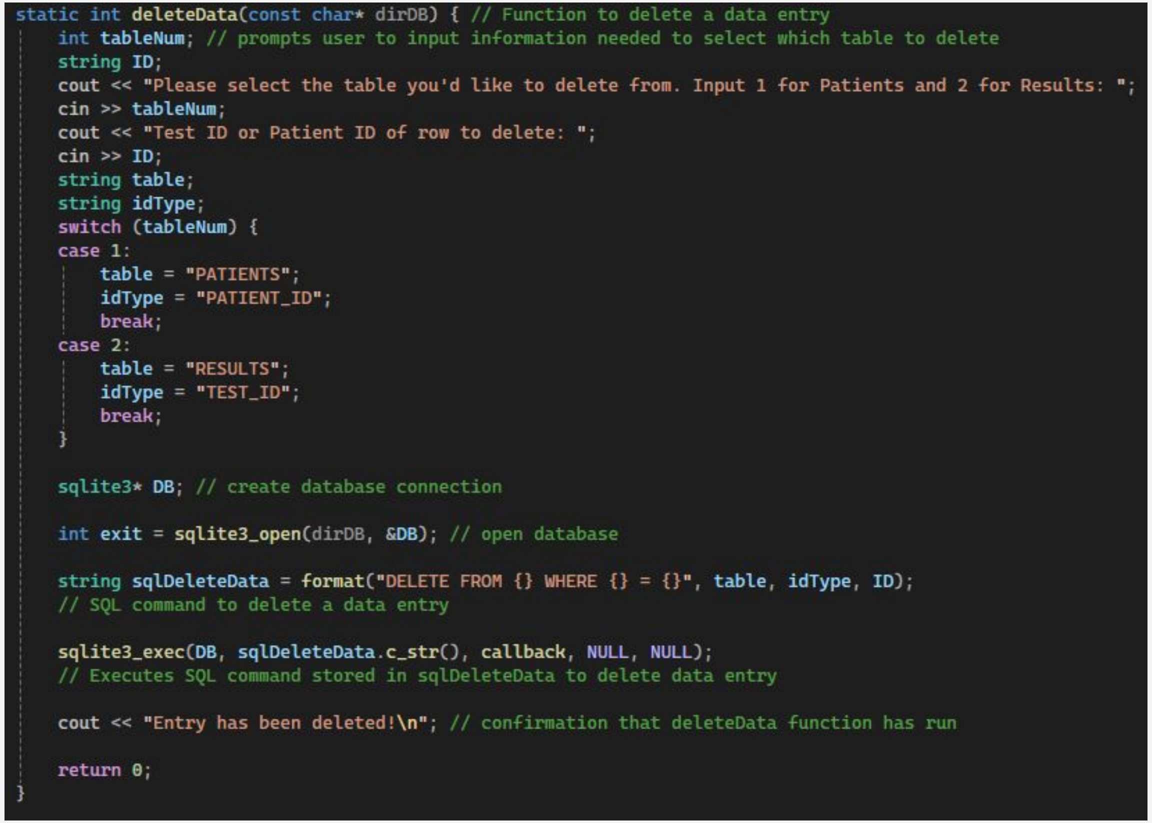
User Interface

- User inputs ‘d’ after menu is displayed
- User selects which table the entry they want to delete is in
- User inputs the Test ID or Patient ID of the entry they want to delete
- Success message is output once the entry is deleted

Methods and Tools

- Switch Statement
- Sqlite Library
- Input and Output with cin and cout
- SQL Delete Statement

## Delete Data - Code



```
static int deleteData(const char* dirDB) { // Function to delete a data entry
    int tableNum; // prompts user to input information needed to select which table to delete
    string ID;
    cout << "Please select the table you'd like to delete from. Input 1 for Patients and 2 for Results: ";
    cin >> tableNum;
    cout << "Test ID or Patient ID of row to delete: ";
    cin >> ID;
    string table;
    string idType;
    switch (tableNum) {
    case 1:
        table = "PATIENTS";
        idType = "PATIENT_ID";
        break;
    case 2:
        table = "RESULTS";
        idType = "TEST_ID";
        break;
    }

    sqlite3* DB; // create database connection

    int exit = sqlite3_open(dirDB, &DB); // open database

    string sqlDeleteData = format("DELETE FROM {} WHERE {} = {}", table, idType, ID);
    // SQL command to delete a data entry

    sqlite3_exec(DB, sqlDeleteData.c_str(), callback, NULL, NULL);
    // Executes SQL command stored in sqlDeleteData to delete data entry

    cout << "Entry has been deleted!\n"; // confirmation that deleteData function has run

    return 0;
}
```

## Notes on Components

Data Storage

- SQL is used via sqlite3 throughout the code as the main way to store patient information

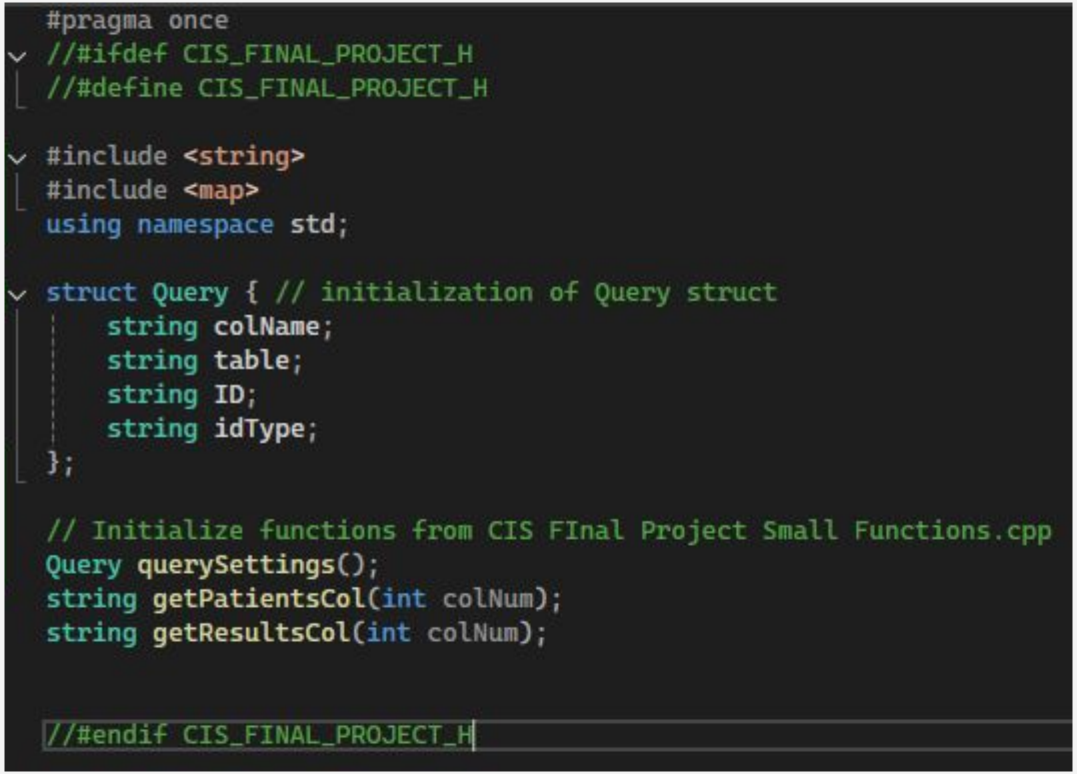
Other Tools Used

- Structures
- Methods/Functions
- .h Files
- iostream Library
- format Library
- numeric Library

Files

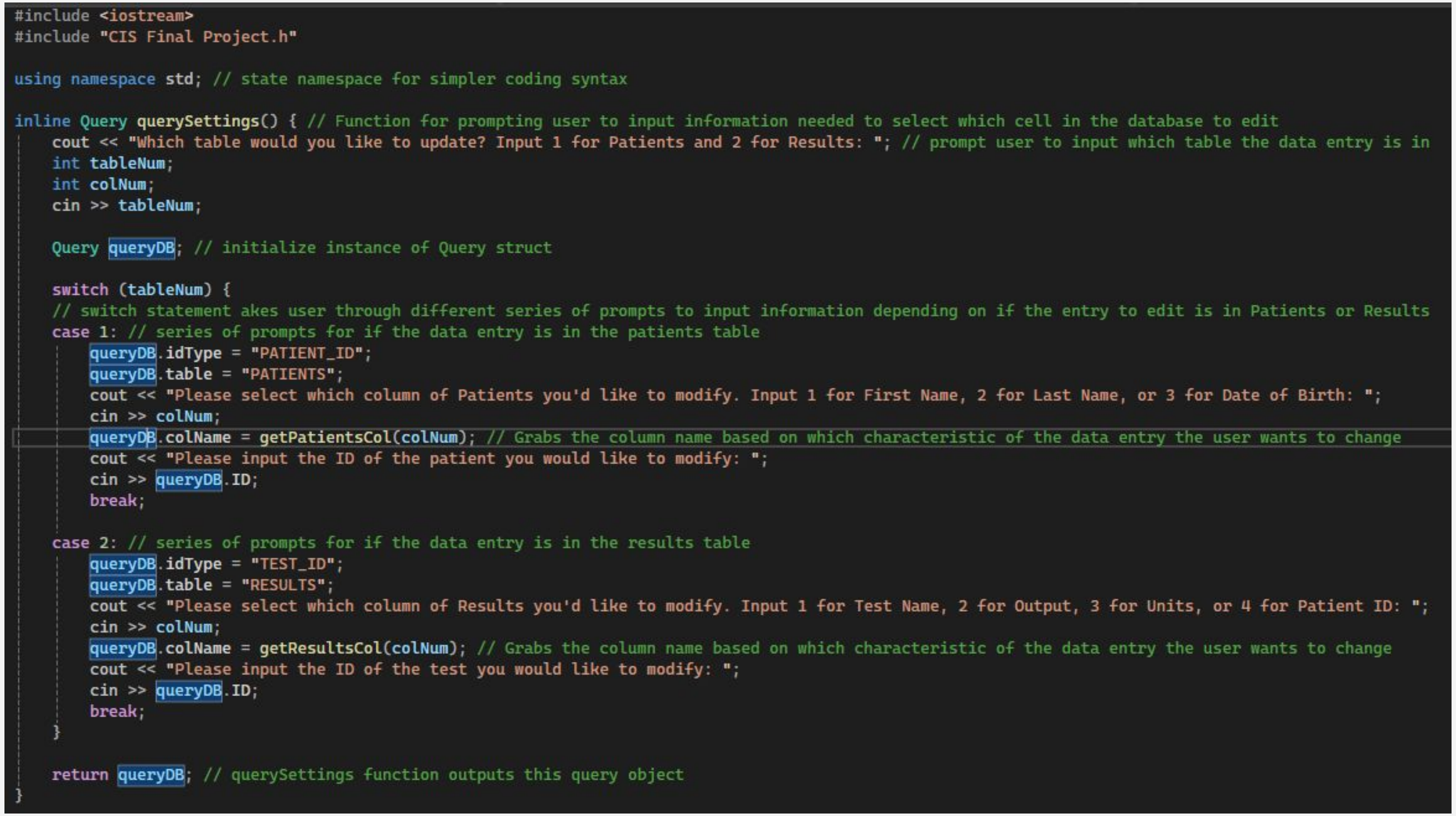
- main.cpp - run this file to run the application; contains the functions directly connected to the menu selections
- CIS Final Project.h - contains Query struct and links background functions to cpp
- CIS Final Small Functions.cpp - contains some functions related to searching the tables and entries
- Sqlite3.c - file that allows usage of the sqlite3 library

# CIS Final Project.h



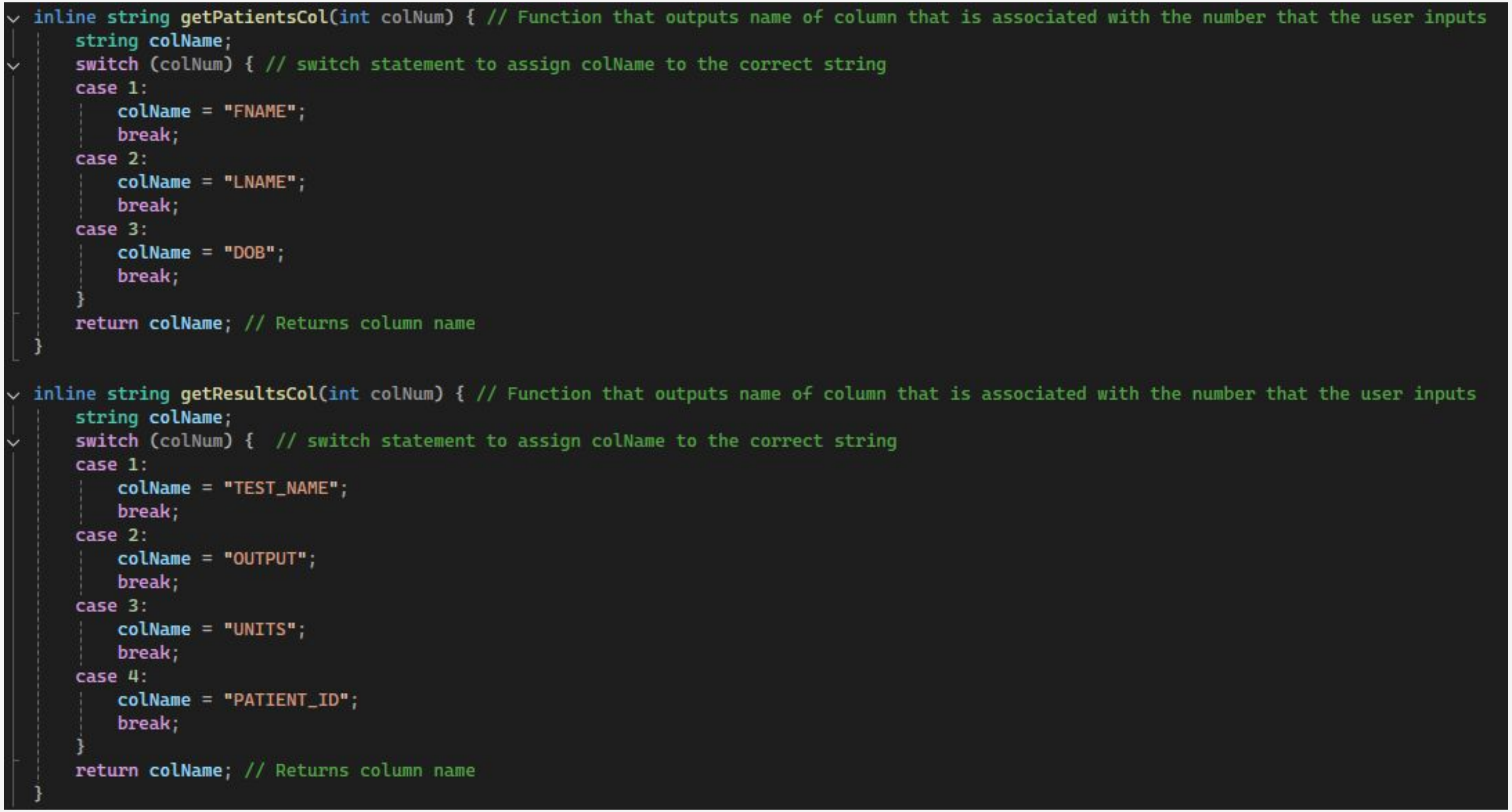
```cpp
#pragma once
//#ifdef CIS_FINAL_PROJECT_H
//#define CIS_FINAL_PROJECT_H

#include <string>
#include <map>
using namespace std;

struct Query { // initialization of Query struct
    string colName;
    string table;
    string ID;
    string idType;
};

// Initialize functions from CIS FInal Project Small Functions.cpp
Query querySettings();
string getPatientsCol(int colNum);
string getResultsCol(int colNum);


//#endif CIS_FINAL_PROJECT_H
```

# CIS Final Project Small Functions.cpp - Part 1



```cpp
#include <iostream>
#include "CIS Final Project.h"

using namespace std; // state namespace for simpler coding syntax

inline Query querySettings() { // Function for prompting user to input information needed to select which cell in the database to edit
    cout << "Which table would you like to update? Input 1 for Patients and 2 for Results: "; // prompt user to input which table the data entry is in
    int tableNum;
    int colNum;
    cin >> tableNum;

    Query queryDB; // initialize instance of Query struct

    switch (tableNum) {
    // switch statement akes user through different series of prompts to input information depending on if the entry to edit is in Patients or Results
    case 1: // series of prompts for if the data entry is in the patients table
        queryDB.idType = "PATIENT_ID";
        queryDB.table = "PATIENTS";
        cout << "Please select which column of Patients you'd like to modify. Input 1 for First Name, 2 for Last Name, or 3 for Date of Birth: ";
        cin >> colNum;
        queryDB.colName = getPatientsCol(colNum); // Grabs the column name based on which characteristic of the data entry the user wants to change
        cout << "Please input the ID of the patient you would like to modify: ";
        cin >> queryDB.ID;
        break;

    case 2: // series of prompts for if the data entry is in the results table
        queryDB.idType = "TEST_ID";
        queryDB.table = "RESULTS";
        cout << "Please select which column of Results you'd like to modify. Input 1 for Test Name, 2 for Output, 3 for Units, or 4 for Patient ID: ";
        cin >> colNum;
        queryDB.colName = getResultsCol(colNum); // Grabs the column name based on which characteristic of the data entry the user wants to change
        cout << "Please input the ID of the test you would like to modify: ";
        cin >> queryDB.ID;
        break;
    }

    return queryDB; // querySettings function outputs this query object
}
```

# CIS Final Project Small Functions.cpp - Part 1



```cpp
inline string getPatientsCol(int colNum) { // Function that outputs name of column that is associated with the number that the user inputs
    string colName;
    switch (colNum) { // switch statement to assign colName to the correct string
    case 1:
        colName = "FNAME";
        break;
    case 2:
        colName = "LNAME";
        break;
    case 3:
        colName = "DOB";
        break;
    }
    return colName; // Returns column name
}

inline string getResultsCol(int colNum) { // Function that outputs name of column that is associated with the number that the user inputs
    string colName;
    switch (colNum) {  // switch statement to assign colName to the correct string
    case 1:
        colName = "TEST_NAME";
        break;
    case 2:
        colName = "OUTPUT";
        break;
    case 3:
        colName = "UNITS";
        break;
    case 4:
        colName = "PATIENT_ID";
        break;
    }
    return colName; // Returns column name
}
```

## Conclusion

## Potential Future Additions to Application

- More complex analysis for different types of data
- Create a more user friendly GUI for viewing the data entries
- Create security for this application since patient and health data is sensitive information

# Links

**Link to Github Files:**
https://github.com/emizheng/CIS25_Fall2025/blob/2a1494e4ff04dcf521012ab375877bfb937c094f/CIS%20Final%20Project%20V1/main.cpp

Video:
https://youtu.be/88VbzdJG2EA