ASSIGNMENT 1

Q1. Compare the Von Neuman computer achitecture and the quatum computer and show how they are programable.
Q2. Write an algorithm and program to accept three tests for a course, compute and display the average for a class of 50 students. Modify this to work for 6 different units that the students registerred for in the semester.

Q2. **Algorithm (a)**

1. **Input:**
   - The number of students in the class (`student_count`) and the number of tests per student (`test_count`) are passed as command-line arguments.
   - For each student, the user's input includes:
     - The student's name.
     - The name of each test.
     - The score for each test.
2. **Process:**
   - Create an array (`students_names`) to store the names of the students.
   - Create a 2D array (`test_names`) to store the names of the tests for each student.
   - Create a 2D array (`test_scores`) to store the test scores for each student.
   - Loop through each student and each test to prompt for the student's name, the test name, and the test score. Store these values in the corresponding arrays.
   - Compute the sum and average of the test scores for each student.
   - Compute the class average by summing all the student averages and dividing by the total number of students.
3. **Output:**
   - Display the details of each student, including the student's name, each test name and score, and the student's average.
   - Display the total class average.

Q2. **Documentation (b)**

The program accepts two command-line arguments (`student_count` and `test_count`) to specify the number of students in the class and the number of tests each student takes. The user is prompted to enter the following for each student:

1. The student’s name.
2. The name of each test.
3. The score of each test.

The program calculates and displays:

- The details of each student, including their name, each test name, test score, and the average score.
- The total class average, which is the average score of all students.

Program Flow:

1. **Input**: The user inputs the number of students and tests as command-line arguments, then enters the names and test scores for each student.
2. **Processing**: The program calculates the average test scores for each student and accumulates the total class average.
3. **Output**: The program displays each student's details, their average, and the total class average.

Example Execution:

Command:

```
./main 3 2
```

Sample Input:

```
Enter student 1 name: Muchangi
Test [1]: Math
Math score: 85
Test [2]: Physics
Physics score: 90

Enter student 2 name: Jane
Test [1]: Math
Math score: 78
Test [2]: Physics
Physics score: 82

Enter student 3 name: Doe
Test [1]: Math
Math score: 88
Test [2]: Physics
Physics score: 84
```

Output:

```
STUDENT DETAILS...
Muchangi   [Math -> 85.00%]  [Physics -> 90.00%]  [Average -> 87.50%]
Jane    [Math -> 78.00%]  [Physics -> 82.00%]  [Average -> 80.00%]
Doe    [Math -> 88.00%]  [Physics -> 84.00%]  [Average -> 86.00%]

TOTAL CLASS AVERAGE => 84.50%
```

Data Structures Used:

- `char students_names[10][128]`: Stores the names of the students.
- `char test_names[10][2][128]`: Stores the names of the tests for each student.
- `double test_scores[10][2]`: Stores the test scores for each student.
- `double total_average`: Accumulates the class average.

Implementation

1. Command-Line Argument Parsing:

The program expects two command-line arguments:

- `argv[1]`: The number of students (`student_count`).
- `argv[2]`: The number of tests per student (`test_count`).

If these are not provided, the program prints an error message and exits:

```
if (argc < 3) {
    fprintf(stderr, "Failed to run the program.\nMake sure to include the
student_count and test_count when calling the program.\ne.g. './main 3 2' -
> 3 students each with 2 tests.\n");
    exit(EXIT_FAILURE);
}
```

2. Variable Initialization:

The program then initializes variables to store student names, test names, and test scores. Arrays of fixed size (`10` students and `2` tests) are used to store this information:

```
char students_names[10][128];
char test_names[10][2][128];
double test_scores[10][2];
double total_average = 0;
```

3. User Input:

The program uses nested loops to prompt the user for each student's details:

- **Outer loop (`i`):** Iterates over the students.
- **Inner loop (`j`):** Iterates over the tests for each student.

For each student, the program:

1. Prompts for the student's name (if it's the first test entry for that student).
2. Prompts for the name of each test.
3. Prompts for the test score and stores it in `test_scores[i][j]`.

```
for (int i = 0; i < student_count; ++i) {
    for (int j = 0; j < test_count; ++j) {
        if (j == 0) {
            printf("\nEnter student %d name : ", i + 1);
            fgets(student_name, sizeof(student_name), stdin);
            strcpy(students_names[i], student_name);
            students_names[i][strcspn(students_names[i], "\n")] = '\0';
        }

        printf("Test [%d] : ", j + 1);
        fgets(test_name, sizeof(test_name), stdin);
        strncpy(test_names[i][j], test_name, strlen(test_name));
        test_names[i][j][strcspn(test_names[i][j], "\n")] = '\0';

        printf("%s score : ", test_names[i][j]);
        scanf(" %lf", &test_scores[i][j]);

        while (getchar() != '\n');
    }
}
```

4. Calculating and Displaying Student Details:

After collecting the data, the program calculates and prints the test scores and average for each student. It also computes the class average:

```
for (int i = 0; i < student_count; ++i) {
    double sum = 0;
    for (int j = 0; j < test_count; ++j) {
        if (j == 0) {
            printf("\n%s    ", students_names[i]);
        }
        printf("  [%s -> %.2lf%%]", test_names[i][j], test_scores[i][j]);
        sum += test_scores[i][j];
    }
    printf("  [Average -> %.2lf%%]",  sum / test_count);
    total_average += sum / test_count;
}
```

5. Displaying Class Average:

Finally, the program calculates and prints the class average by dividing the total of all students' averages by the number of students:

```
printf("\nTOTAL CLASS AVERAGE => %.2lf%%\n\n", total_average /
student_count);
```