Project 1 Question 3 Version 2 Documentation

1. Aim of this Version of the Program

The aim of this version of the program is to add the concept of structure and function on the original version to improve code structure, organization, and maintainability of the program. This newer version also includes dynamic arrays to enhance flexibility in handling any number of runners.

2. Process:

- ✓ The code prompts the user to enter the number of runners.

```
Please enter the number of runners: 2
Please enter the name of the runner: Abebe Haile
Please enter the number of miles run by Abebe Haile on Monday: 1
Please enter the number of miles run by Abebe Haile on Tuesday: 2
Please enter the number of miles run by Abebe Haile on Wednesday: 3
Please enter the number of miles run by Abebe Haile on Thursday: 4
Please enter the number of miles run by Abebe Haile on Friday: 5
Please enter the number of miles run by Abebe Haile on Saturday: 6
Please enter the number of miles run by Abebe Haile on Sunday: 7
Please enter the name of the runner: Ayele Alemu
Please enter the number of miles run by Ayele Alemu on Monday: 7
Please enter the number of miles run by Ayele Alemu on Tuesday: 6
Please enter the number of miles run by Ayele Alemu on Wednesday: 5
Please enter the number of miles run by Ayele Alemu on Thursday: 4
Please enter the number of miles run by Ayele Alemu on Friday: 3
Please enter the number of miles run by Ayele Alemu on Saturday: 3
Please enter the number of miles run by Ayele Alemu on Sunday: 2
```

Fig 2.1: Shows how the program accepts the input

- ✓ It creates an array of runner structures based on the number of runners.

```
struct runner {
    string name;
    double mile[num_d];
    double total_m;
    double average_m;
};
```

Fig 2.2: Declare structure

```
int num;
cout << "Please enter the number of runners: ";
cin >> num;

runner* person = new runner[num];
```

Fig 2.3: Created array of runner structure

- ✓ The “accept” function is called for each runner to accept their name and the miles run on each day of the week.

```
void accept(runner& person, const string names[]) {
    cin.ignore();
    cout << "Please enter the name of the runner: ";
    getline(cin, person.name);
    for (int j = 0; j < 7; j++) {
        cout << "Please enter the number of miles run by " << person.name << " on " << names[j + 1] << ": ";
        cin >> person.mile[j];
    }
}
```

Fig 2.4: “accept” function

- ✓ The “cal” function is called for each runner to calculate their total miles and average miles.

```
void cal(runner& person) {
    for (int j = 0; j < 7; j++) {
        person.total_m += person.mile[j];
        person.mile[7] = person.total_m;
    }
    person.average_m = person.mile[7] / (num_d - 2);
    person.mile[8] = person.average_m;
}
```

Fig 2.5: “cal” fuction

- ✓ The choice function allows the user to choose between displaying data for all runners or a specific runner.
- ✓ Based on the user's choice, the appropriate data is printed using the printer function.
- ✓ The program continues to loop until the user chooses to exit.

```
Enter:
 0. to exit,
 1. to display data for all runners, or
 2. to display data for a specific runner: 1
names                Monday      Tuesday     Wednesday   Thursday    Friday      Saturday    Sunday      total miles average miles
Abebe Haile          1           2           3           4           5           6           7           28          4
Ayele Alemu          7           6           5           4           3           3           2           30          4.28571
Enter:
 0. to exit,
 1. to display data for all runners, or
 2. to display data for a specific runner: 0
THANK YOU FOR USING THIS PROGRAM!!
Process returned 0 (0x0)   execution time : 1077.251 s
Press any key to continue.
```

Fig 2.6: Shows how the information is displayed on screen

- ✓ Finally, the dynamic array is deleted, and a thank you message is displayed.

```
delete[] person;
cout << "THANK YOU FOR USING THIS PROGRAM!!";
return 0;
```

Fig 2.7: The code that deletes the dynamic array and displays the thank you message

3. New Features:

3.1 Structure:

The code introduces a structure called runner to store the data for each runner. It includes their name, miles run on each day, total miles, and average miles. Using a structure allows for better organization and encapsulation of related data.

3.2 Dynamic Array:

The code uses a dynamic array of runner structures to accommodate the variable number of runners specified by the user. This allows for flexibility and scalability.

3.3 Functions with Structures:

The functions (accept, cal, printer, and choice) are designed to work with the runner structure. They accept and process data specific to each runner, allowing for code reuse and better modularity.

4. Advantages of this Version:

- ✓ **Readability and Maintainability**: By using structures and separating tasks into functions, the code becomes more readable and maintainable. Each function focuses on a specific task, making it easier to understand and modify.
- ✓ **Flexibility:** The dynamic array and user input allow for flexibility in handling any number of runners. The code can adapt to different scenarios without the need for code changes.
- ✓ **Code Reusability**: With modular functions and structures, the code can be reused in other parts of the program or in different programs. For example, the printer function can be used to print data for other similar scenarios involving structures.
- ✓ **Encapsulation:** The structure encapsulates the data related to each runner, promoting data integrity and better organization. It provides a logical and cohesive way to group related data elements.
- ✓ **Scalability**: The code can handle a large number of runners without sacrificing performance or readability. The dynamic array allows for efficient memory allocation based on user input.