

Swinburne University of Technology

Faculty of Science, Engineering, and Technology

COVID Recommendation Group Report

Date of Submission	31/05/2021
Semester	Semester 1, 2021
Unit	SWE20004
Lab Tutor	Anika Kanwal

Design Project

By Design Project Group 4

Student Name	Student ID	Lab day/Time/Room	Unit code
Henil Shah	102064095	Monday 10:30 EN409	SWE20004
Ahmed Elmi	102575597	Monday 10:30 EN409	SWE20004
Dejan Poznan	102579094	Monday 10:30 EN409	SWE20004

Table of Contents

1) Project Description	3
2) Project Objectives and Problem Statement	3
3) Design and Development.....	4
4) Specifications.....	6
5) Data Display	8
6) Code Explanation	16
7) Project Outcomes.....	39
8) Executive summary of project solution/outcome	39

Consultation Phase

1) Project Description

The pandemic known as Corona Virus or COVID has placed the world under threat. In order to counter the spread of the Virus, individuals who have been feeling ill were informed to stay isolated in their homes. However, many are unaware of the symptoms of COVID which might seem like a simple cough or cold to be in fact could be the virus attempting to take over their body. This is why it is essential for people to have an easy and effective way to diagnose themselves without putting others and themselves in danger by going into public areas. A brief overview of the entire project was the creation of a program that had the capabilities to alert a user to go and take a COVID test based upon the symptoms the user is experiencing and if they have had visited a high-risk location at all. This was done in order to lift the burden of health workers across Australia, because in turn less and less people will go and take a test in person because they can just take an online test instead. This method of testing also decreases the chances of spreading COVID as people who are not infected would not go the hospital to do the test and hence would not interact with someone who might have the virus.

2) Project Objectives and Problem Statement

Problem Statement: During this COVID pandemic, people need to be able to check if they have got the COVID so that they can get treatment quickly. But many people are unsure when they should go to take the test. Because COVID is not easy to detect, most people unaware of the symptoms who do take the COVID test do not need it which delays treatment to patients who actually have COVID. Our objective is to create a C++ program with primary database of at least 10 sample datasets consisting of information like name, date of birth, address, symptoms, etc. and allows users to enter their details that are saved into the database. Based on their details users are recommended whether they should take the COVID test or isolate themselves at home.

Preliminary design concept

3) Design and Development

In the home screen, we will provide the user with 5 options, the 6th allowing them to exit the program. Users will use integers to maneuver through the program. This was done to make the program as user friendly as possible, each section was created to allow users that are all sorts of ages to be able to diagnose themselves.

The program will allow the user to enter details for the COVID. It will ask to enter an ID and then personal details such as names, DOB, address and if the user has been overseas. Then it will display the common symptoms and COVID high risk areas across Victoria. The symptoms will be ordered by how risky it is to a person and how likely based on their symptoms that they might have COVID. We will show them in ascending order from least risky to most risky to an individual. This allows the people who are undergoing this portable test to grasp a better understanding on the severity of their symptom. The program will determine whether the user should isolate themselves at home or go to take the COVID test at the nearest facility. Some areas in the program will only be accessed if the user is recorded as a COVID positive patient. This is done for security reasons. The program also has a feature that allows the user to update locations they have recently been as COVID positive patients. Which keeps the program dynamic as it is constantly being updated by several patients. Through this, the government, health workers and the public can be aware of areas they should avoid in the future.

Implementing information:

The symptoms we will employ are as follows: the low risks Headache and Nasal Congestion, the medium risks Fever and Dry coughs, the high risks are Shortness of breath. These symptoms have been researched and taken from multiple health related websites. The COVID high risk areas that we will include in the program are Doncaster (Shopping Centre), Wantirna South (Hospital), Dandenong (Rebel Sport), Epping (7/11), Broadmeadows (Car Park), Sunshine (Hotel), Craigieburn (Petrol Station), Auburn (Krispy Kreme), Sandringham (Train Station). However, these areas may increase as users will be able to update them.

The bulk of our code will be consisting of arrays to store the data from database and making changes to them. After making all changes the data will be written back to the database using for loops and arrays. Fstream library which contains ofstream and ifstream classes would be essential for reading and writing to the database files. While loops will be used to loop until desired input is entered by the user. We will use pointers to change and access value of variables by accessing their address to use them in various functions.

The factors that dictate whether the user should take the COVID test as per requirements include what symptoms they have and whether they visited to high-risk location. There are 3 degrees of symptoms – low risk, medium risk and high risk. And depending upon whether an

individual went to high risk or not and what symptoms they have, we have 6 possible combinations.

	Low risk	Medium risk	High risk
Went to high-risk location	Take COVID test	Take COVID test	Take COVID test
Did not went to high-risk location	Isolate themselves at home	Isolate themselves at home	Take COVID test

Table 1 Combinations of symptoms and high-risk locations

As shown in Table 1, if the user went to high-risk location, then regardless of their degree of symptoms, they should take the COVID test. If the user did not visit high-risk location and they have high-risk symptoms, then they should also take the COVID test. If the user did not visit high-risk location and happen to have low risk or medium risk symptoms, then they should isolate themselves at home.

Compatibility of Design

4) Specifications

Our specified project had very strict instructions and guidelines that we had to follow to deliver the best possible solution. Probably the most straight-forward requirement was to ensure that everything we have written was written in C++ and no other coding language, this was obviously done to test our knowledge and skills up until this point. Not only this, but we also had to ensure that we wrote appropriate and relevant comments in our code, alongside a brief explanation of our variables and explanation for any type of code we may have, this was important to ensure that our code ran to the specifications and made debugging much easier on a development point-of-view.

We had to think carefully on how we wanted to tackle the specifications given to use and how we were going to overcome the challenges laid before us. We had to sure that with our program, that when the user selects the option 1, then they should automatically be prompted to enter their ID, First Name, Date of Birth, Address, Overseas travel, which symptoms they are suffering from and select a high-risk area they may have visited. But from this we had to ensure to only recommend a COVID test only to people who have more severe symptoms and have also visited a high-risk location, if they have not then they should be told to self-isolate at home.

However, if the symptoms database is empty then a message along the lines of “Unable to recommend COVID test – missing data.”

Moving on to option 2 of our program, we add to ensure than we a user were to select option two, they will be given a prompt that asks them to enter their ID, the program would them the user to enter their current COVID status (being either positive or negative) and then get recorded in a database alongside the location they have visited, which will ultimately be added to the list of high-risk COVID locations. This ties in with our option 3 also which displays all updated locations (included those which are high-risk.)

Option 3 demanded of us, to display a list of multiple locations that are known to be hotspots for COVID and are ultimately quite dangerous to be in. However, it should also display the location the user input if the select option 2 to submit their COVID status.

All of these options tie in perfectly with option 4, as option 4 is the one that allows us to change COVID patient details that will ultimately appear in option 5 which allows us to

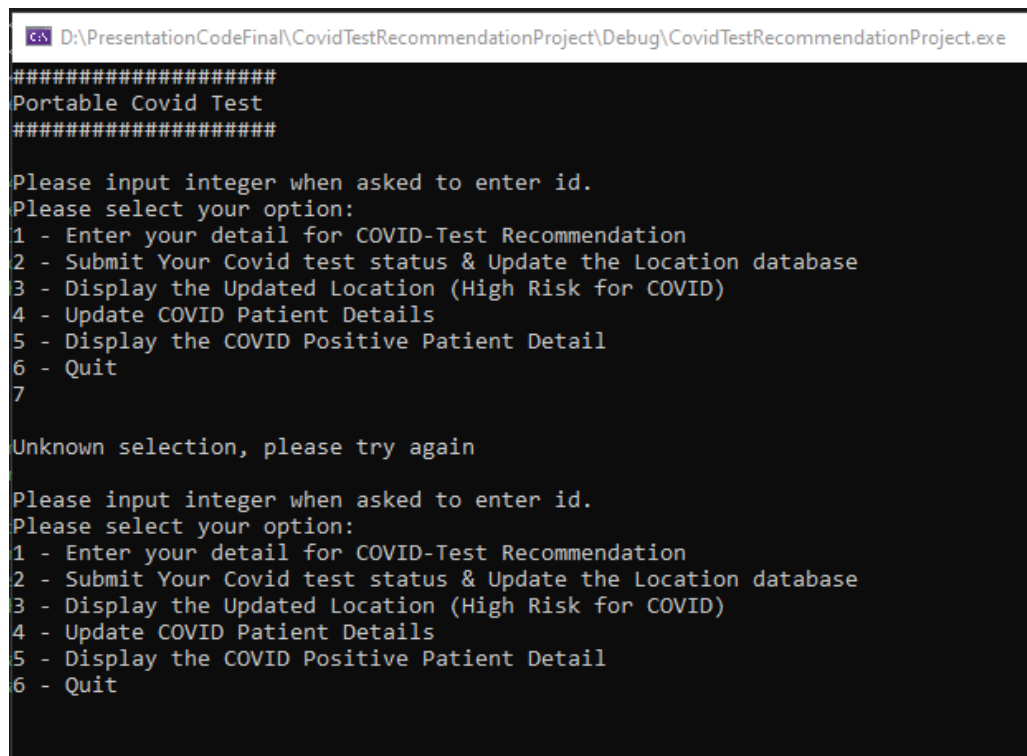
preview all COVID patient details, so we would be able to see the change if someone goes from being Cured to Dead.

Option 5 is our final option and lastly displays all our patients' details such as Name, Date of Birth, their address, visited location and last time they travelled overseas etc... This option works hand-in-hand with the rest of the program as it the one the allows use to see the changes made by the user. Lastly option 6 will terminate the program.

5) Data Display

When it came down to actually writing the code, we had to make sure in the case an invalid option was selected by the user, a message should appear with something along the lines of “Unknown selection, please try again” that would then bring the user back to the main menu with the original options displayed once again. For example, the message will appear when the user selects an incorrect menu option.

Option 1:



```
D:\PresentationCodeFinal\CovidTestRecommendationProject\Debug\CovidTestRecommendationProject.exe
#####
Portable Covid Test
#####

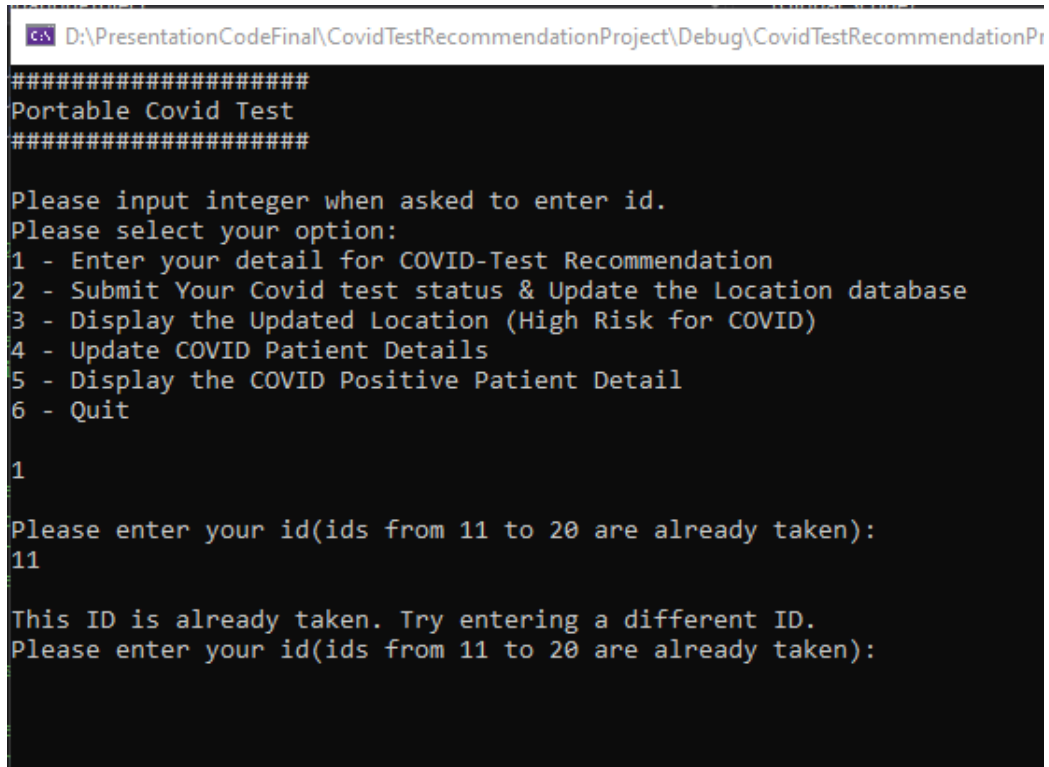
Please input integer when asked to enter id.
Please select your option:
1 - Enter your detail for COVID-Test Recommendation
2 - Submit Your Covid test status & Update the Location database
3 - Display the Updated Location (High Risk for COVID)
4 - Update COVID Patient Details
5 - Display the COVID Positive Patient Detail
6 - Quit
7

Unknown selection, please try again

Please input integer when asked to enter id.
Please select your option:
1 - Enter your detail for COVID-Test Recommendation
2 - Submit Your Covid test status & Update the Location database
3 - Display the Updated Location (High Risk for COVID)
4 - Update COVID Patient Details
5 - Display the COVID Positive Patient Detail
6 - Quit
```

Figure 5.1

Option 1: When invalid ID is entered this message is shown to let the user know to re-enter the ID:



```
C:\D:\PresentationCodeFinal\CovidTestRecommendationProject\Debug\CovidTestRecommendationPr

#####
Portable Covid Test
#####

Please input integer when asked to enter id.
Please select your option:
1 - Enter your detail for COVID-Test Recommendation
2 - Submit Your Covid test status & Update the Location database
3 - Display the Updated Location (High Risk for COVID)
4 - Update COVID Patient Details
5 - Display the COVID Positive Patient Detail
6 - Quit

1

Please enter your id(ids from 11 to 20 are already taken):
11

This ID is already taken. Try entering a different ID.
Please enter your id(ids from 11 to 20 are already taken):
```

Figure 5.2

Option 1: When the user successfully enters their details, they are recommended to take the COVID test

```
Please enter your id(ids from 11 to 20 are already taken):
147

Please enter your first name:
John

Please enter your family/last name:
Doe

Please enter your date of birth(dd/mm/yyyy):
15/02/1982

Please enter your Address:
65 John St Hawthorn

Have you travelled overseas recently? (Yes/No):
Yes

Please select symptoms from options given:
1 Headache
2 Nasal congestion
3 Fever
4 Dry cough
5 Shortness of breath
6 Chest pain or pressure
7 Loss of speech or movement
3

Please select visited location from High Risk COVID area from options given:
0 None
1 Doncaster (Shopping Centre)
2 Wantirna South (Hospital)
3 Dandenong (Rebel Sport)
4 Epping (7/11)
5 Broadmedows (Car Park)
6 Sunshine (Hotel)
7 Craigeburn (Petrol Station)
8 Auburn (Krispy Kreme)
9 Sandringham (Train Station)
2

It is recommended that you should take the COVID test.
```

Figure 5.3

Option 2: When the user enters id that does not exists message in Figure 5.4 is shown, and they are asked to enter the ID again

```
Please input integer when asked to enter id.
Please select your option:
1 - Enter your detail for COVID-Test Recommendation
2 - Submit Your Covid test status & Update the Location database
3 - Display the Updated Location (High Risk for COVID)
4 - Update COVID Patient Details
5 - Display the COVID Positive Patient Detail
6 - Quit
2

Please enter your id:
157
This id does not exist. Please enter an existing id.

Please enter your id:
```

Figure 5.4

Option 2: When the user changes the COVID test result to positive and enter the new visited location:

```
Please input integer when asked to enter id.
Please select your option:
1 - Enter your detail for COVID-Test Recommendation
2 - Submit Your Covid test status & Update the Location database
3 - Display the Updated Location (High Risk for COVID)
4 - Update COVID Patient Details
5 - Display the COVID Positive Patient Detail
6 - Quit
2

Please enter your id:
157
This id does not exist. Please enter an existing id.

Please enter your id:
147

Please enter the COVID test status (type p for Positive/ n for Negative):
p

Please enter new visited location:
Hospital
```

Figure 5.5

Option 3: When the user selects the option 3 all the high-risk locations are displayed(also hospital which was added in Figure 5.5)

```
Please input integer when asked to enter id.  
Please select your option:  
1 - Enter your detail for COVID-Test Recommendation  
2 - Submit Your Covid test status & Update the Location database  
3 - Display the Updated Location (High Risk for COVID)  
4 - Update COVID Patient Details  
5 - Display the COVID Positive Patient Detail  
6 - Quit  
3  
  
Doncaster (Shopping Centre)  
Wantirna South (Hospital)  
Dandenong (Rebel Sport)  
Epping (7/11)  
Broadmedows (Car Park)  
Sunshine (Hotel)  
Craigeburn (Petrol Station)  
Auburn (Krispy Kreme)  
Sandringham (Train Station)  
Hospital
```

Figure 5.6

Option 4: When the user inputs a non-COVID positive patient's ID

Note: ID 12 is negative COVID patient which is part of sample datasets.

The sample ids are from 11 to 20.

```
Please input integer when asked to enter id.  
Please select your option:  
1 - Enter your detail for COVID-Test Recommendation  
2 - Submit Your Covid test status & Update the Location database  
3 - Display the Updated Location (High Risk for COVID)  
4 - Update COVID Patient Details  
5 - Display the COVID Positive Patient Detail  
6 - Quit  
4  
  
Please enter id of the COVID positive patient:  
12  
The person with this id is not Covid positive.  
  
Please enter id of the COVID positive patient:
```

Figure 5.7

Option 4: When the user enters a valid ID, and the user input is saved into the status database

```
Please input integer when asked to enter id.  
Please select your option:  
1 - Enter your detail for COVID-Test Recommendation  
2 - Submit Your Covid test status & Update the Location database  
3 - Display the Updated Location (High Risk for COVID)  
4 - Update COVID Patient Details  
5 - Display the COVID Positive Patient Detail  
6 - Quit  
4  
  
Please enter id of the COVID positive patient:  
12  
The person with this id is not Covid positive.  
  
Please enter id of the COVID positive patient:  
157  
The person with this id is not Covid positive.  
  
Please enter id of the COVID positive patient:  
147  
  
Enter the new status of the patient(Cured/Dead):  
Dead
```

Figure 5.8

Option 5: When option 5 is selected, details of all positive patients are displayed.

```
Please input integer when asked to enter id.
Please select your option:
1 - Enter your detail for COVID-Test Recommendation
2 - Submit Your Covid test status & Update the Location database
3 - Display the Updated Location (High Risk for COVID)
4 - Update COVID Patient Details
5 - Display the COVID Positive Patient Detail
6 - Quit
5

Patient id: 11
Name: Wade
Date of Birth: 05/01/1980
Address: 89 Chapman Avenue
Visited Location: Federation Square
Date/Timing: 05/01/2021 04:00 PM
Last Overseas Travel: No
Covid Test: Positive
Status: Alive

Patient id: 14
Name: Ivan
Date of Birth: 05/01/1985
Address: 83 George Street
Visited Location: Southbank
Date/Timing: 20/01/2021 03:00 PM
Last Overseas Travel: No
Covid Test: Positive
Status: Alive

Patient id: 17
Name: Jorge
Date of Birth: 05/12/1980
Address: 47 Mnimbah Road
Visited Location: Melbourne Museum
Date/Timing: 17/01/2021 12:15 PM
Last Overseas Travel: No
Covid Test: Positive
Status: Alive

Patient id: 20
Name: Roberto
Date of Birth: 28/07/1990
Address: 67 Wallis Street
Visited Location: Shrine of Remembrance
Date/Timing: 01/05/2021 03:00 PM
Last Overseas Travel: Yes
Covid Test: Positive
Status: Alive

Patient id: 147
Name: John Doe
Date of Birth: 15/02/1982
Address: 65 John St Hawthorn
Visited Location: [] - the database is empty
Date/Timing: [] - the database is empty
Last Overseas Travel: Yes
Covid Test: Positive
Status: Dead
```

Figure 5.9

Option 6: When the user selects option 6, they are displayed “Goodbye” message and the program ends.

```
Please input integer when asked to enter id.  
Please select your option:  
1 - Enter your detail for COVID-Test Recommendation  
2 - Submit Your Covid test status & Update the Location database  
3 - Display the Updated Location (High Risk for COVID)  
4 - Update COVID Patient Details  
5 - Display the COVID Positive Patient Detail  
6 - Quit  
6  
  
Goodbye  
  
D:\PresentationCodeFinal\CovidTestRecommendationProject\Debug\CovidTest  
To automatically close the console when debugging stops, enable Tools->  
Press any key to close this window . . .
```

6) Code Explanation

```

2  #include <iostream>
3  #include <fstream>
4  #include <string>
5  using namespace std;
6
7  /*
8   SWE20004 - Group : Design Project 4
9   Group members :
10  Ahmed Elmi - student ID : 102575597
11  Dejan Poznan - student ID : 102579094
12  Henil Shah - student ID : 103190205
13  */
14
15  /*
16  Name of the file : ProjectCovidDPG4.cpp
17  Purpose of the program : The purpose of the program is
18  to recommend the user whether they should take the COVID Test
19  based on the details they enter and store that data.
20
21  The design solution/overview of the program : The design solution for the project is to
22  first create a primary database(a txt file) using ofstream and
23  add 10 sample datasets(randomly created based on structure mentioned in requirements) to it.
24  After that, data written to the primary database is read into arrays which allows to make
25  changes to data more easily.
26  After making necessary changes to the arrays,
27  the data inside the arrays is written back to the txt file.
28  */

```

Figure 6.1

In Figure 6.1, the libraries used in this project and namespace std are mentioned. Details mentioned as per the requirement are also added as comments – Group number, group members and their student ID, name of the file, purpose of the program, design solution.

```

30  #define SIZE 200
31  #define DEFAULTDATASIZE 10
32
33  template <class dataType>
34  void writeToFile(string fileName, dataType data);
35  void createPatientDetailTable();
36  void symptomsTable(string lowRisk[], string mediumRisk[], string highRisk[]);
37  void highRiskLocationTable(string highRiskLocationsDb[]);
38  void readFromIntFileToArray(string fileName, int savingToArray[], int* numberOfElements);
39  void readFromStringFileToArray(string fileName, string savingToArray[], int* highRiskLocationsSize);
40
41  void enterDetails(int patientId[], string name[], string dob[], string address[],
42  string lastOverseasTravel[], string symptomsRisk[], string highRiskLocations[],
43  int numberOfElements, string lowRisk[2], string mediumRisk[2], string highRisk[3],
44  int getIdIndex, int highRiskLocationsSize);
45
46  void displayPositivePatientDetails(int patientId[], string name[], string dob[], string address[],
47  string visitedLocation[], string dateTime[], string lastOverseasTravel[], string covidTest[],
48  string status[]);
49
50  bool isDuplicate(int idToFind, int arrayToFindIn[], int size, int* getIdIndex);
51  void submitAndUpdate(int patientId[], string covidTest[], int getIdIndex, string highRiskLocations[],
52  int size, int highRiskLocationsSize);
53
54  void updateCovidPositiveStatusDetail(int patientId[], int getIdIndex, string covidTest[], string status[]);
55
56  template <class dataType>
57  void writeToFileForUpdate(string fileName, dataType data[], int numberOfElements, bool sizeChanged);
58

```

Figure 6.2

Figure 6.2 lists all constants and function prototypes needed for functions.


```
58
59 int main() {
60
61     /*
62     Arrays are declared here which will be used to store and manage data read from database files.
63     */
64     int patientId[SIZE], option, getIdIndex = 0, numberOfElements = 10, highRiskLocationsSize = 0, arraySize = 0;
65     string name[SIZE], dob[SIZE], address[SIZE], visitedLocation[SIZE], dateTime[SIZE], lastOverseasTravel[SIZE],
66         covidTest[SIZE], status[SIZE], symptomsRisk[SIZE], highRiskLocations[SIZE];
67
68     string lowRisk[2] = { "Headache", "Nasal congestion" };
69     string mediumRisk[2] = { "Fever", "Dry cough" };
70     string highRisk[3] = { "Shortness of breath", "Chest pain or pressure", "Loss of speech or movement" };
71
72     string highRiskLocationsDb[DEFAULTDATASIZE] = { "None", "Doncaster (Shopping Centre)",
73         "Wantirna South (Hospital)", "Dandenong (Rebel Sport)", "Epping (7/11)", "Broadmedows (Car Park)",
74         "Sunshine (Hotel)", "Craigieburn (Petrol Station)", "Auburn (Krispy Kreme)", "Sandringham (Train Station)" };
75 }
```

Figure 6.3

In Figure 6.3, main function is created and inside it, variables and arrays needed to manage the data and run the program are declared. Some arrays are populated because they are needed to make the primary database – lowRisk, mediumRisk, highRisk and highRiskLocationsDb.

```
75
76 // Task 1: Creating a Primary Database
77 createPatientDetailTable();
78 symptomsTable(lowRisk, mediumRisk, highRisk);
79 highRiskLocationTable(highRiskLocationsDb);
80
81 // Task 2: Recommend the COVID Test
82 cout << "#####" << endl;
83 cout << "Portable Covid Test " << endl;
84 cout << "#####" << endl;
85
```

Figure 6.4

In Figure 6.4, functions named createPatientDetailTable, symptomsTable and highRiskLocationTable are called. Function createPatientDetailTable creates 10 sample datasets for patient details - patient ID, name, date of birth, address, visited locations, date and time, last overseas travel, COVID test results and status of the patient. Function symptomsTable takes arrays lowRisk, mediumRisk and highRisk as arguments and creates a database file for symptoms. Similarly, highRiskLocationTable takes array highRiskLocationsDb as an argument and creates a database file for high-risk locations.

```
85
86  /* The options(1 to 6) should be display until user enters 6 to quit the program
87  |hence bool isRunning is false when 6 is selected.*/
88  bool isRunning = true;
89  while (isRunning) {
90
91      // Load the data from existing sample database
92      readFromFileToArray("patientIdTableDatabase", patientId, &numberOfElements);
93      readFromStringFileToArray("nameTableDatabase", name, &numberOfElements);
94      readFromStringFileToArray("dobDatabase", dob, &numberOfElements);
95      readFromStringFileToArray("addressDatabase", address, &numberOfElements);
96      readFromStringFileToArray("lastOverseasTravelDatabase", lastOverseasTravel, &numberOfElements);
97
98      readFromStringFileToArray("visitedLocationDatabase", visitedLocation, &arraySize);
99      readFromStringFileToArray("dateTimeDatabase", dateTime, &arraySize);
100     readFromStringFileToArray("covidTestDatabase", covidTest, &arraySize);
101     readFromStringFileToArray("statusDatabase", status, &arraySize);
102     readFromStringFileToArray("symptomsRiskDatabase", symptomsRisk, &arraySize);
103
104     readFromStringFileToArray("highRiskLocationsDatabase", highRiskLocations, &highRiskLocationsSize);
```

Figure 6.5

In Figure 6.5, a boolean named `isRunning` with while loop is used to loop through the options presented in Figure 6.6 until a valid input is entered. Function `readFromFileToArray` reads integer values from file consisting of integers (patient IDs). Reference of `numberOfElements`, `arraySize` and `highRiskLocationsSize` are passed as arguments which keeps track of how many elements exist inside that particular array. Similarly, function `readFromStringFileToArray` reads string data from file consisting of strings.

```
105
106     cout << "\nPlease input integer when asked to enter id." << endl;
107     cout << "Please select your option: " << endl;
108     cout << "1 - Enter your detail for COVID-Test Recommendation" << endl;
109     cout << "2 - Submit Your Covid test status & Update the Location database" << endl;
110     cout << "3 - Display the Updated Location (High Risk for COVID)" << endl;
111     cout << "4 - Update COVID Patient Details" << endl;
112     cout << "5 - Display the COVID Positive Patient Detail" << endl;
113     cout << "6 - Quit" << endl;
114     cin >> option;
115
```

Figure 6.6

In Figure 6.6, options are displayed with numbers beside them as per the requirements. After the user enters their choice, it is saved into a variable called `option`.

```
116 switch (option) {
117     case 1:
118
119         // Function enterDetails collects user data and recommends COVID test based on input data.
120         enterDetails(patientId, name, dob, address, lastOverseasTravel, symptomsRisk, highRiskLocations,
121             numberOfElements, lowRisk, mediumRisk, highRisk, getIdIndex, highRiskLocationsSize);
122
123         /* After collecting data from user and giving recommendation, the data is written to the database so when
124            it is accessed again the data is updated.*/
125         writeToFileForUpdate("patientIdTableDatabase", patientId, numberOfElements, true);
126         writeToFileForUpdate("nameTableDatabase", name, numberOfElements, true);
127         writeToFileForUpdate("dobDatabase", dob, numberOfElements, true);
128         writeToFileForUpdate("addressDatabase", address, numberOfElements, true);
129         writeToFileForUpdate("lastOverseasTravelDatabase", lastOverseasTravel, numberOfElements, true);
130         break;
131
```

Figure 6.7

In Figure 6.7, a switch case is used which takes an option selected by the user as an input and executes relevant functions inside that case statement. So, when option 1 is selected the functions `enterDetails` and `writeToFileForUpdate` execute. The function `enterDetails` adds the data entered by the user into the relevant arrays (patient ID in `patientId` array, etc.) and then recommends whether they should take the COVID test or isolate themselves at home. Function `writeToFileForUpdate` writes the changes made to the arrays by the function `enterDetails` to the database files.

```
131
132     case 2:
133
134         /* Function submitAndUpdate allows users to make changes to their COVID test result(Positive/Negative)
135            and asks for their visited location if the result is positive.*/
136         submitAndUpdate(patientId, covidTest, getIdIndex, highRiskLocations, numberOfElements, highRiskLocationsSize);
137
138         /* After collecting data from user and updating their COVID test result, the data
139            is written to the database so when it is accessed again the data is updated.*/
140         writeToFileForUpdate("covidTestDatabase", covidTest, numberOfElements, false);
141         break;
142
```

Figure 6.8

In Figure 6.8, when the user selects option 2 a function named `submitAndUpdate` is executed. This function allows the user to update their COVID test results and asks them to enter a new location if their test result is positive. The location entered is then saved and written into the high-risk locations database file. Function `writeToFileForUpdate` updates the COVID test results database as per the newly entered results.

```
142
143     case 3:
144
145         // Prints all updated high risk locations
146         cout << endl;
147         for (int i = 1; i < highRiskLocationsSize; i++) {
148             cout << highRiskLocations[i] << endl;
149         }
150         break;
151
```

Figure 6.9

In Figure 6.9, when the user selects option 3 the updated high-risk locations are displayed.

```
151
152     case 4:
153
154         /* Function updateCovidPositiveStatusDetail allows user to make changes to their
155          * status(from positive to Cured/Dead) using their id.*/
156         updateCovidPositiveStatusDetail(patientId, getIdIndex, covidTest, status);
157
158         /* After collecting data from user and updating their status, the data
159          * is written to the database so when it is accessed again the data is updated.*/
160         writeToFileForUpdate("covidTestDatabase", covidTest, numberOfElements, false);
161         writeToFileForUpdate("statusDatabase", status, numberOfElements, false);
162         break;
163
```

Figure 6.10

In Figure 6.10, when the user selects option 4 the function `updateCovidPositiveStatusDetail` updates the status of the COVID positive patient to the input entered by the user with the help of their ID. Similarly, as before, function `writeToFileForUpdate` updates the COVID test results database and status database.

```
163
164     case 5:
165
166         // Displays all patients who are COVID positive
167         displayPositivePatientDetails(patientId, name, dob, address, visitedLocation, dateTime, lastOverseasTravel, covidTest, status);
168         break;
169
170     case 6:
171
172         cout << "\nGoodbye" << endl;
173         isRunning = false;
174         break;
175
176     default:
177
178         cout << "\nUnknown selection, please try again" << endl;
179         break;
180     }
181 }
182
183 return 0;
184 }
```

Figure 6.11

In Figure 6.11, when the user selects option 5, all the details of the COVID positive patients are displayed. When the user selects option 6, “Goodbye” is printed to the console and the while loop terminates because `isRunning` (mentioned in Figure 6.5) is set to false. If the user inputs an invalid choice (other than 1-6) then the default case is executed which notifies users that an unknown selection is made. After that message, the options are shown again, and the program waits for the user input. On line 183, `return 0` is used to successfully end the main function.

```
185
186 // Function isThere checks if the file already exists - returns true if it does.
187 bool isThere(string fileName) {
188     ifstream myfile;
189     myfile.open(fileName + ".txt");
190     if (myfile) {
191         return true;
192     }
193     return false;
194 }
```

Figure 6.12

In Figure 6.12, a function named `isThere` is defined that takes a string as a parameter and returns boolean. This function checks if the text file (whose name is the parameter) already exists. It returns true if the file exists else returns false.

```
196 void createPatientDetailTable() {
197
198     // Creating file for storing patient ids and adding 10 sample data
199     if (!isThere("patientIdTableDatabase")) {
200         int patientIdDb[DEFAULTDATASIZE] = { 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
201         for (int i = 0; i < DEFAULTDATASIZE; i++) {
202             writeToFile<int>("patientIdTableDatabase", patientIdDb[i]);
203         }
204     }
205
206     // Creating file for storing names and adding 10 sample data
207     if (!isThere("nameTableDatabase")) {
208         string nameDb[DEFAULTDATASIZE] = { "Wade", "Dave", "Seth", "Ivan", "Riley", "Gilbert", "Jorge", "Dan", "Brian", "Roberto" };
209         for (int i = 0; i < DEFAULTDATASIZE; i++) {
210             writeToFile<string>("nameTableDatabase", nameDb[i]);
211         }
212     }
213
214     // Creating file for storing date of birth and adding 10 sample data
215     if (!isThere("dobDatabase")) {
216         string dobDb[DEFAULTDATASIZE] = { "05/01/1980", "05/05/1980", "01/03/1980", "05/01/1985",
217             "05/10/1980", "20/01/1980", "05/12/1980", "15/01/1983", "25/08/1984", "28/07/1990" };
218
219         for (int i = 0; i < DEFAULTDATASIZE; i++) {
220             writeToFile<string>("dobDatabase", dobDb[i]);
221         }
222     }
223
224     // Creating file for storing address and adding 10 sample data
225     if (!isThere("addressDatabase")) {
226         string addressDb[DEFAULTDATASIZE] = { "89 Chapman Avenue", "53 Capper Street", "24 Butler Crescent",
227             "83 George Street", "83 Hereford Avenue", "27 Fitzroy Street", "47 Mnimbah Road", "79 Zipfs Road",
228             "59 Whitehaven Crescent", "67 Wallis Street" };
229
230         for (int i = 0; i < DEFAULTDATASIZE; i++) {
231             writeToFile<string>("addressDatabase", addressDb[i]);
232         }
233     }
234 }
```

Figure 6.13

In Figure 6.13, a function named `createPatientDetailTable` is defined. Inside that function, the function `isThere` is used to check whether the patient ID, name, date of birth, and address databases already exists. If it does not exist, it is created using their respective arrays: `patientIdDb`, `nameDb`, `dobDB` and `addressDb` that has sample data. Function `writeToFile` is then used write to the file.

```
235 // Creating file for storing visited Locations and adding 10 sample data
236 if (!isThere("visitedLocationDatabase")) {
237     string visitedLocationDb[DEFAULTDATASIZE] = { "Federation Square", "Royal Botanic Gardens",
238     "Melbourne Cricket Ground", "Southbank", "National Gallery of Victoria", "Eureka Tower",
239     "Melbourne Museum", "Melbourne Zoo", "Docklands", "Shrine of Remembrance" };
240
241     for (int i = 0; i < DEFAULTDATASIZE; i++) {
242         writeToFile<string>("visitedLocationDatabase", visitedLocationDb[i]);
243     }
244 }
245
246 // Creating file for storing DateTime and adding 10 sample data
247 if (!isThere("dateTimeDatabase")) {
248     string dateTimeDb[DEFAULTDATASIZE] = { "05/01/2021 04:00 PM", "09/03/2021 12:00 PM",
249     "15/02/2021 01:00 PM", "20/01/2021 03:00 PM", "13/01/2021 05:15 PM", "28/02/2021 03:20 PM",
250     "17/01/2021 12:15 PM", "14/04/2021 02:45 PM", "28/03/2021 11:00 AM", "01/05/2021 03:00 PM" };
251
252     for (int i = 0; i < DEFAULTDATASIZE; i++) {
253         writeToFile<string>("dateTimeDatabase", dateTimeDb[i]);
254     }
255 }
256
257 // Creating file for storing Last Overseas Travel and adding 10 sample data
258 if (!isThere("lastOverseasTravelDatabase")) {
259     string lastOverseasTravelDb[DEFAULTDATASIZE] = { "No", "No", "Yes", "No", "No", "Yes", "No", "Yes", "No", "Yes" };
260     for (int i = 0; i < DEFAULTDATASIZE; i++) {
261         writeToFile<string>("lastOverseasTravelDatabase", lastOverseasTravelDb[i]);
262     }
263 }
264
265 // Creating file for storing Covid Test results and adding 10 sample data
266 if (!isThere("covidTestDatabase")) {
267     string covidTestDb[DEFAULTDATASIZE] = { "Positive", "Negative", "Negative", "Positive",
268     "Negative", "Negative", "Positive", "Negative", "Negative", "Positive" };
269
270     for (int i = 0; i < DEFAULTDATASIZE; i++) {
271         writeToFile<string>("covidTestDatabase", covidTestDb[i]);
272     }
273 }
```

Figure 6.14

In Figure 6.14, function `isThere` is used to check whether the visited locations, date and timing, last overseas travel and COVID test result databases already exist. If they do not exist, they are created using the arrays: `visitedLocation`, `dateTimeDb`, `lastOverseasTravelDatabase` and `covidTestDb`.

Each of these arrays have different sample data.

`visitedLocation`: Sample visited locations data.

`dateTimeDb`: Sample dates and timings for visited locations.

`covidTestDb`: Sample data consisting of "Positive" and "Negative".

`lastOverseasTravelDatabase`: Sample data consisting of "Yes" and "No".


```
274
275 // Creating file for storing status of patients and adding 10 sample data
276 if (!isThere("statusDatabase")) {
277     string statusDb[DEFAULTDATASIZE] = { "Alive", "Alive", "Alive", "Alive", "Alive", "Alive",
278     "Alive", "Alive", "Alive", "Alive" };
279
280     for (int i = 0; i < DEFAULTDATASIZE; i++) {
281         writeToFile<string>("statusDatabase", statusDb[i]);
282     }
283 }
284
285
286 // Function symptomsTable create symptoms database with all symptoms defined earlier
287 // with the use of writeToFile function.
288 void symptomsTable(string lowRisk[], string mediumRisk[], string highRisk[]) {
289
290     if (!isThere("symptomsRiskDatabase")) {
291
292         writeToFile<string>("symptomsRiskDatabase", lowRisk[0]);
293         writeToFile<string>("symptomsRiskDatabase", lowRisk[1]);
294         writeToFile<string>("symptomsRiskDatabase", mediumRisk[0]);
295         writeToFile<string>("symptomsRiskDatabase", mediumRisk[1]);
296         writeToFile<string>("symptomsRiskDatabase", highRisk[0]);
297         writeToFile<string>("symptomsRiskDatabase", highRisk[1]);
298         writeToFile<string>("symptomsRiskDatabase", highRisk[2]);
299     }
300 }
301
302 // Function highRiskLocationTable create high risk locations database with all locations defined earlier
303 // with the use of writeToFile function.
304 void highRiskLocationTable(string highRiskLocationsDb[]) {
305
306     if (!isThere("highRiskLocationsDatabase")) {
307
308         for (int i = 0; i < DEFAULTDATASIZE; i++) {
309             writeToFile<string>("highRiskLocationsDatabase", highRiskLocationsDb[i]);
310         }
311     }
312 }
```

Figure 6.15

In Figure 6.15, function `isThere` is used to check if status database of the patients already exists. If the database does not exist, then it is created using the data from array `statusDb` that has sample data consisting of “Alive” and function `writeToFile` is used to write to the file.

On line 288, function `symptomsTable` is defined which takes 3 parameters. These parameters are arrays which contains the symptoms as shown in Figure 6.3. Then if the file of symptoms database does not exist then it is created, and the arrays passed as parameters are written to the database.

On line 304, function `highRiskLocationTable` is defined which takes `highRiskLocationsDb` array as a parameter and if the file of high-risk locations database does not exist then it is created and filled with data from `highRiskLocationsDb` array.


```
314  /*
315  * Function writeToFile writes data that is provided in argument data to the file whose name is provided
316  * in argument fileName.
317  * Since different datatypes are being written(int for ids, string for everything else), template was used to
318  * write generic function which accepts both datatype.
319  */
320  template <class dataType>
321  void writeToFile(string fileName, dataType data) {
322      ofstream myfile;
323      myfile.open(fileName + ".txt", ios::app);
324      if (!myfile) {
325          cout << fileName << " file not created!";
326      }
327      else {
328          myfile << data << endl;
329      }
330      myfile.close();
331  }
```

Figure 6.16

In Figure 6.16, a function named `writeToFile` is defined which takes a string `fileName` and datatype `data` defined by template as parameters. Template is used so that it can be used for integers(IDs) and strings (all other data). Using the `ofstream` class, a file whose name is taken from `fileName` parameter is opened in append mode. If the file is successfully created, then that data from `datatype data` parameter is written to the file or else a message is printed to the console to let the user know that file was not created. After that file is closed using `close()` method.

```
332
333  /*
334  * Function readFromIntFileToArray reads data from an int file(for ids) into an array.
335  * It also keeps track of numbers of elements within the array(hence giving array size) since sizeof function
336  * was giving wrong output(error code: C6384) when getting size of an array.
337  */
338  void readFromIntFileToArray(string fileName, int savingToArray[], int* numberOfElements) {
339      int index = 0, temp;
340      ifstream myfile;
341      myfile.open(fileName + ".txt");
342
343      if (myfile.is_open()) {
344          while (myfile >> temp) {
345              savingToArray[index] = temp;
346              // variable index keeps track of number of elements encountered hence giving array size
347              index++;
348          }
349      }
350      else {
351          cout << "Unable to open file " << fileName << endl;
352      }
353
354      *numberOfElements = index;
355      myfile.close();
356  }
357
```

Figure 6.17

In Figure 6.17, a function named `readFromIntFileToArray` is defined which takes string `fileName` - to get file name to read from, integer `savingToArray` - save the data to a file and integer pointer `numberOfElements` - to keep track of how many elements are there in the file.

Variable `index` is defined as 0 since at the start there will be no elements, so 1st element is 0th index. Using `ifstream` class and `fileName` passed as parameter, the file is opened to read.

While there is data inside the file it is read and saved into the `temp` variable and then from `temp` to array `savingToArray` using the `index` variable. `Index` is incremented by 1 after saving so for next time when loop runs the data is not overridden. If the file was not opened due to some error, then users are informed using an error message. The value of `index` is then saved using the pointer variable `numberOfElements` to know number of elements inside the file and the file is closed using `close()` method.

```
357
358  /*
359  * Similar to function readFromIntFileToArray but for datatype string
360  */
361  void readFromStringFileToArray(string fileName, string savingToArray[], int* highRiskLocationsSize) {
362      int index = 0;
363      string line;
364      ifstream myfile;
365      myfile.open(fileName + ".txt");
366
367      if (myfile.is_open()) {
368          while (getline(myfile, line)) {
369              savingToArray[index] = line;
370              index++;
371          }
372      }
373      else {
374          cout << "Unable to open file " << fileName << endl;
375      }
376
377      *highRiskLocationsSize = index;
378      myfile.close();
379  }
```

Figure 6.18

Similar to function `readFromIntFileToArray` defined in Figure 6.17, in Figure 6.18 the function `readFromStringFileToArray` is defined and achieves the same results but it is used to read from files containing string data.

```
381 void enterDetails(int patientId[], string name[], string dob[], string address[],
382 string lastOverseasTravel[], string symptomsRisk[], string highRiskLocations[],
383 int numberOfElements, string lowRisk[2], string mediumRisk[2], string highRisk[3], int getIdIndex,
384 int highRiskLocationsSize) {
385
386     bool inputId = true;
387     int patientIDInput;
388
389     // Loops until user inputs a valid/new id
390     while (inputId) {
391         cout << "\nPlease enter your id(ids from 11 to 20 are already taken): " << endl;
392         cin >> patientIDInput;
393
394         // Checks if the id already exists
395         if (isDuplicate(patientIDInput, patientId, SIZE, &getIdIndex)) {
396             cout << "\nThis ID is already taken. Try entering a different ID.";
397         }
398         else {
399             inputId = false;
400         }
401     }
402
403     // Adds new id to the array patientId which is later written to database.
404     patientId[numberOfElements] = patientIDInput;
405 }
```

Figure 6.19

In Figure 6.19, a function named `enterDetails` is defined which is executed when user wants to enter their details to get recommended for a COVID test recommendation. On line 390, a while loop is used to ask for user input until a valid ID is provided. Duplicate ID check is done by the function `isDuplicate` on line 395 and is defined later within the code. If the ID is duplicate the program lets the user know and asks for their input again. But if a valid ID is entered then it is stored inside the `patientId` array to later write that ID to the patient ID database file.

```
405
406     cout << "\nPlease enter your first name: " << endl;
407     string firstName;
408     cin.ignore();
409     getline(cin, firstName);
410     cout << "\nPlease enter your family/last name: " << endl;
411     string lastName;
412     getline(cin, lastName);
413     // Adds new name to the array name which is later written to database.
414     name[numberOfElements] = firstName + " " + lastName;
415
416     cout << "\nPlease enter your date of birth(dd/mm/yyyy): " << endl;
417     // Adds new date of birth to the array dob which is later written to database.
418     getline(cin, dob[numberOfElements]);
419
420     cout << "\nPlease enter your Address: " << endl;
421     // Adds new address to the array address which is later written to database.
422     getline(cin, address[numberOfElements]);
423
424     cout << "\nHave you travelled overseas recently? (Yes/No): " << endl;
425     // Adds the input(Yes/No) by user to the array lastOverseasTravel which is later written to database.
426     getline(cin, lastOverseasTravel[numberOfElements]);
427
```

Figure 6.20

In Figure 6.20, the user is asked for first and last name. The input is saved into temporary variables `firstName` and `lastName`. `Getline` is used to store full first name and last name inputs to the array named `name` to later write to the name database file. Similarly, date of birth, address, last overseas travel of patient is stored into the arrays `dob`, `address`, `lastOverseasTravel` respectively.

```
428     bool invalidSymptomSelected = true;
429     // Loops until user selects a valid option
430     while (invalidSymptomSelected) {
431         cout << "\nPlease select symptoms from options given: " << endl;
432         cout << "1 " << lowRisk[0] << endl;
433         cout << "2 " << lowRisk[1] << endl;
434         cout << "3 " << mediumRisk[0] << endl;
435         cout << "4 " << mediumRisk[1] << endl;
436         cout << "5 " << highRisk[0] << endl;
437         cout << "6 " << highRisk[1] << endl;
438         cout << "7 " << highRisk[2] << endl;
439
440         int selectedOption;
441         cin >> selectedOption;
442         switch (selectedOption)
443         {
444             // Option selected is added to array symptomsRisk.
445             case 1:
446                 symptomsRisk[numberOfElements] = lowRisk[0];
447                 invalidSymptomSelected = false;
448                 break;
449
450             case 2:
451                 symptomsRisk[numberOfElements] = lowRisk[1];
452                 invalidSymptomSelected = false;
453                 break;
454
455             case 3:
456                 symptomsRisk[numberOfElements] = mediumRisk[0];
457                 invalidSymptomSelected = false;
458                 break;
```

Figure 6.21

In Figure 6.21, while loop runs until a valid input is entered. Inside that loop, options from 1 to 7 are displayed in severity from low to high. The input is temporarily stored and used in switch case. If the input is 1 then the relevant option displayed earlier which is lowRisk[0] gets stored into the symptomsRisk array. And the boolean invalidSymptomSelected is set to false which causes the while loop to terminate. Similarly, if option 2 is selected then lowRisk[1] is saved into the symptomsRisk array and rest of the options are saved likewise to symptomsRisk array.

```
459
460     case 4:
461         symptomsRisk[numberOfElements] = mediumRisk[1];
462         invalidSymptomSelected = false;
463         break;
464
465     case 5:
466         symptomsRisk[numberOfElements] = highRisk[0];
467         invalidSymptomSelected = false;
468         break;
469
470     case 6:
471         symptomsRisk[numberOfElements] = highRisk[1];
472         invalidSymptomSelected = false;
473         break;
474
475     case 7:
476         symptomsRisk[numberOfElements] = highRisk[2];
477         invalidSymptomSelected = false;
478         break;
479
480     default:
481         cout << "Invalid option selected!" << endl;
482         break;
483     }
484 }
485
```

Figure 6.22

Much like in Figure 6.21, the option selected is saved into the symptomsRisk array. If invalid input is entered then the default case lets the user know that invalid option is selected, and the program asks the user for input again (since invalidSymptomSelected is not set to false here).

```
486 string locationSelected;
487 bool invalidLocationSelected = true;
488 // Loops until user selects a valid option
489 while (invalidLocationSelected) {
490     cout << "\nPlease select visited location from High Risk COVID area from options given: " << endl;
491
492     // Prints options from the array highRiskLocations
493     for (int i = 0; i < highRiskLocationsSize; i++)
494     {
495         cout << i << " " << highRiskLocations[i] << endl;
496     }
497
498     int selectedOption;
499     cin >> selectedOption;
500
501     // Checks if the user input a valid option and saves selected option in variable locationSelected
502     if (selectedOption >= 0 && selectedOption < highRiskLocationsSize) {
503         locationSelected = highRiskLocations[selectedOption];
504         invalidLocationSelected = false;
505     }
506     else {
507         cout << "Invalid option selected!" << endl;
508     }
509
510     /* All 6 combinations
511     (1) Went to high risk location and has low risk symptom.
512     (2) Went to high risk location and has medium risk symptom.
513     (3) Went to high risk location and has high risk symptom.
514     (4) Did not went to high risk location and has low risk symptom.
515     (5) Did not went to high risk location and has medium risk symptom.
516     (6) Did not went to high risk location and has high risk symptom.*/
```

Figure 6.23

In Figure 6.23, like Figure 6.21 and 6.22, the user is asked to select a high-risk location they have visited from the options displayed using for loop and array `highRiskLocations` on line 493. If valid input is entered the location is saved into the variable `locationSelected` for further use otherwise the user is asked for the input again. On line 510, all six combinations (discussed in Design and Development) are written as comments to let the person reading the code better understand the code presented in Figure 6.24.


```
517
518     /* If the option selected is not "None"(which means high risk location is selected) from options
519     then recommend them to take the test. This takes care of combination - (1), (2) and (3).*/
520     if (!(locationSelected == highRiskLocations[0])) {
521         cout << "\nIt is recommended that you should take the COVID test." << endl;
522     }
523     else {
524
525         /* If the option selected is "None"(which means they did not went to high risk location) from options
526         and have high risk then recommend them to take the test. This takes care of combination (6).*/
527         if (symptomsRisk[numberOfElements] == highRisk[0] || symptomsRisk[numberOfElements] == highRisk[1]
528             || symptomsRisk[numberOfElements] == highRisk[2]) {
529             cout << "\nIt is recommended that you should take the COVID test." << endl;
530         }
531
532         /* If the option selected is "None"(which means they did not went to high risk location) from options
533         and have low or medium risk then recommend them to isolate themselves at home.
534         This takes care of combinations - (4) and (5).*/
535         else {
536             cout << "\nIsolate yourself at home." << endl;
537         }
538     }
539 }
540
541 }
```

Figure 6.24

In Figure 6.24, on line 520 if the location selected is not `highRiskLocations[0]`("None") then that means user went to a high-risk location and therefore they are recommended to take the COVID test. Otherwise, if they have a high-risk symptom then they are also recommended to take the COVID test. If the user did not go to a high-risk location and has no high-risk symptoms, then they are advised to self-isolate themselves at home.

```

543 void displayPositivePatientDetails(int patientId[], string name[], string dob[], string address[],
544 string visitedLocation[], string dateTime[], string lastOverseasTravel[], string covidTest[],
545 string status[]) {
546
547     int index = 0;
548     string line;
549     ifstream myfile;
550     myfile.open("covidTestDatabase.txt");
551
552     if (myfile.is_open()) {
553         while (getline(myfile, line)) {
554
555             /* Finds the index where the COVID positive patient is and then uses that index
556             to get corresponding values of other fields(patient id, name, etc).*/
557             if (!line.compare("positive") || !line.compare("POSITIVE") || !line.compare("Positive")) {
558                 cout << "\nPatient id: " << patientId[index] << endl;
559                 cout << "Name: " << name[index] << endl;
560                 cout << "Date of Birth: " << dob[index] << endl;
561                 cout << "Address: " << address[index] << endl;
562
563                 if (visitedLocation[index].empty()) {
564                     cout << "Visited Location: " << "[] - the database is empty" << endl;
565                 }
566                 else {
567                     cout << "Visited Location: " << visitedLocation[index] << endl;
568                 }
569
570                 if (dateTime[index].empty()) {
571                     cout << "Date/Timing: " << "[] - the database is empty" << endl;
572                 }
573                 else {
574                     cout << "Date/Timing: " << dateTime[index] << endl;
575                 }
576
577                 cout << "Last Overseas Travel: " << lastOverseasTravel[index] << endl;
578
579                 if (covidTest[index].empty()) {
580                     cout << "Covid Test: " << "[] - the database is empty" << endl;
581                 }
582                 else {
583                     cout << "Covid Test: " << covidTest[index] << endl;
584                 }
585
586                 if (status[index].empty()) {
587                     cout << "Status: " << "[] - the database is empty" << endl;
588                 }
589                 else {
590                     cout << "Status: " << status[index] << endl;
591                 }
592             }
593             index++;
594         }
595     }
596     else {
597         cout << "Unable to open file " << "covidTestDatabase.txt" << endl;
598     }
599
600     myfile.close();
601 }

```

Figure 6.25

In Figure 6.25, the function `displayPositivePatientDetails` is defined which uses `ifstream` class to read data from COVID test results database. If the patient's COVID test results are positive then all the details of that patient are shown using the corresponding index of arrays `patientId`, `name`, `dob`, `address`, `visitedLocation`, `dateTime`, `lastOverseasTravel`, `covidTest` and `status`. If the database is empty, then the user is informed as shown on line 564 in Figure 6.25. The index is incremented by 1 to go to the next element. If due to some reason program fails to open the file, then the message on line 597 is shown. At the end, the file is close using `close()` method.

```
602
603 bool isDuplicate(int idToFind, int arrayToFindIn[], int size, int* getIdIndex) {
604
605     /*Checks if idToFind exists inside the array arrayToFindIn and if it does then getIdIndex
606     gets the index of that so it can be used for other operations.*/
607     for (int i = 0; i < size; i++) {
608         if (idToFind == arrayToFindIn[i]) {
609             *getIdIndex = i;
610             return true;
611         }
612     }
613     return false;
614 }
```

Figure 6.26

In Figure 6.26, function `isDuplicate` is defined which checks if the integer `idToFind` is present within the array `arrayToFindIn`. Function `isDuplicate` returns `true` if it finds the `idToFind` within the array `arrayToFindIn` and saves the index of that ID using a pointer variable `getIdIndex`.

```
615
616 void submitAndUpdate(int patientId[], string covidTest[], int getIdIndex, string highRiskLocations[],
617 int size, int highRiskLocationsSize) {
618
619     int id;
620     bool checkForId = true;
621     // Loops until user enters a valid input
622     while (checkForId) {
623         cout << "\nPlease enter your id: " << endl;
624         cin >> id;
625
626         // Checks if the id already exists
627         if (isDuplicate(id, patientId, size, &getIdIndex)) {
628             cout << "\nPlease enter the COVID test status (type p for Positive/ n for Negative): " << endl;
629
630             // Uses getIdIndex to store the input
631             cin >> covidTest[getIdIndex];
632
633             // If the input is positive and ask and store the new location.
634             if (!covidTest[getIdIndex].compare("p") || !covidTest[getIdIndex].compare("P")) {
635                 covidTest[getIdIndex] = "Positive";
636
637                 cout << "\nPlease enter new visited location: " << endl;
638
639                 cin.ignore();
640                 getline(cin, highRiskLocations[highRiskLocationsSize]);
641                 writeToDatabaseForUpdate("highRiskLocationsDatabase", highRiskLocations, highRiskLocationsSize, true);
642             }
643
644             // If the input is negative, store it.
645             else {
646                 covidTest[getIdIndex] = "Negative";
647             }
648             checkForId = false;
649         }
650         else {
651             cout << "This id does not exist. Please enter an existing id." << endl;
652         }
653     }
654 }
655
656
```

Figure 6.27

In Figure 6.27, function `submitAndUpdate` is defined which allows the user to submit and update their COVID test results. The user is first asked to enter their ID and if the ID is valid then they are asked to enter their COVID test result as shown on line 628 in Figure 6.27. If their result is positive, the user is asked to enter their new visited location which is then added to high-risk locations database as shown on line 637, 640 and 641. If their ID is not valid then the user is asked again for their input.

```
656
657 void updateCovidPositiveStatusDetail(int patientId[], int getIdIndex, string covidTest[], string status[]) {
658
659     int id;
660     bool checkForId = true;
661     // Loops until user enters a valid input
662     while (checkForId) {
663         cout << "\nPlease enter id of the COVID positive patient: " << endl;
664         cin >> id;
665
666         // Checks if the id already exists
667         if (isDuplicate(id, patientId, SIZE, &getIdIndex)) {
668             if (!covidTest[getIdIndex].compare("Positive") || !covidTest[getIdIndex].compare("POSITIVE")
669                 || !covidTest[getIdIndex].compare("positive")) {
670
671                 cout << "\nEnter the new status of the patient(Cured/Dead): " << endl;
672                 cin >> status[getIdIndex];
673
674                 /* After storing the status, if the status entered is cured then the patient is no longer positive
675                  hence change their COVID result to negative.*/
676                 if (!status[getIdIndex].compare("Cured") || !status[getIdIndex].compare("cured")
677                     || !status[getIdIndex].compare("CURED")) {
678
679                     covidTest[getIdIndex] = "Negative";
680                 }
681                 checkForId = false;
682             }
683             else {
684                 cout << "The person with this id is not Covid positive." << endl;
685             }
686         }
687         else {
688             cout << "\nThis id does not exist. Please enter a existing id." << endl;
689         }
690     }
691 }
692
```

Figure 6.28

In Figure 6.28, function `updateCovidPositiveStatusDetail` is defined which allows users to change status of a COVID positive patient. First the user is asked to enter their ID, if the ID is valid then they are asked to enter the status of the patient. The user input is saved into the status array and if the status they enter is cured then the COVID test results of that patient is changed to negative. If the ID user enters is invalid, then the program asks for user input again.

```
692
693     template <class dataType>
694     void writeToFileForUpdate(string fileName, dataType data[], int numberOfElements, bool sizeChanged) {
695         ofstream myfile;
696         myfile.open(fileName + ".txt");
697         if (!myfile) {
698             cout << fileName << " file not created!";
699         }
700         else {
701             /* Sometimes the data is added and sometimes data is updated, so data is written
702             to the file for different amount of times hence bool sizeChanged is used indicate which
703             loop to execute.*/
704             if (sizeChanged) {
705                 for (int i = 0; i < numberOfElements + 1; i++) {
706                     myfile << data[i] << endl;
707                 }
708             }
709             else {
710                 for (int i = 0; i < numberOfElements; i++) {
711                     myfile << data[i] << endl;
712                 }
713             }
714         }
715     }
716     myfile.close();
717 }
718
719
```

Figure 6.29

In Figure 6.29, the function `writeToFileForUpdate` is defined which writes data to the file. The data and file name are passed as parameters `dataType data` and `string fileName` respectively. Template was used to write different datatypes to file. Using `ofstream` class, a file is created and if boolean parameter `sizeChanged` is false the for loop writes the data for same amount of time as the elements in the array `data` before any changes – this means the database is updating since the number of elements written to the database are same. But if the `sizeChanged` is true then the for loop runs for one more time compared to updating the database – since the loop runs for one more time it means the new data has been added to the database hence one extra iteration of the for loop. After the writing to the file, the file is closed using `close()` method.

Project Outcomes

7) Project Outcomes

The project outcome is to prevent the spread of the COVID virus, has allowed health workers around the world to save both their time and resources. Rather than going to the hospital whenever an individual is feeling ill. We have successfully created a faster way to indicate whether someone should take a COVID test or isolate themselves. This also saves recourses for individuals who have a higher risk of contracting the virus such as the senior citizens or people immune disorders. Additionally, this program protects people who may have the virus but are unaware that they do, as it depicts the symptoms as well. Ultimately, this project has become a benefit for both medical workers and citizens globally minimising the threat of COVID. We have been able to do that successfully by making a database and saving details entered by the users. Based on user details they are recommended to take the COVID test and isolate themselves at home.

8) Executive summary of project solution/outcome

This report was commissioned to determine how a C++ program can be ultimately used for the greater good to battle COVID-19 as it decreases the number of workers required to administer the tests to people in person. Our findings indicate that is far simpler and cost efficient to have a COVID test in this format as demonstrated by the criteria. Our objective originally starting out was to create a solution for people to take the COVID test and this goes perfectly with our goal which is to reduce the burden on health workers across the world. Our final solution was a resounding success according to our original scope and specifications as it clearly helps target the people who need to take the COVID test because of their symptoms and the people who should go self-isolate so that they in turn do not end up spreading the virus even more. Although we may have few blunders during the development of our solution, we were able to satisfy all requirements given to us successfully.