

**FUNDAMENTALS IN SOFTWARE DEVELOPMENT**

ASSIGNMENT

PROGRAM DOCUMENTATION

BY:

ABIYYU TAJ MAHASIN BAGINDO  
TP058652

APU1F2002CS(IS)

Table of Contents

[Introduction and Assumptions 3](#_Toc42548324)

[Design 3](#_Toc42548325)

[Pseudocode 3](#_Toc42548326)

[Flowchart 25](#_Toc42548327)

[Source Code with Explanation 45](#_Toc42548328)

[Variables 45](#_Toc42548329)

[String 45](#_Toc42548330)

[Integers 45](#_Toc42548331)

[Control Structure 46](#_Toc42548332)

[If 46](#_Toc42548333)

[Elif 46](#_Toc42548334)

[Nested Elif 47](#_Toc42548335)

[Loops 48](#_Toc42548336)

[For loops 48](#_Toc42548337)

[List 49](#_Toc42548338)

[Functions 50](#_Toc42548339)

[Functions without parameters 51](#_Toc42548340)

[Functions with parameters 51](#_Toc42548341)

[Functions with returns 52](#_Toc42548342)

[Screenshots 53](#_Toc42548343)

[Conclusion 57](#_Toc42548344)

Github repository : <https://github.com/yuto-dev/fsd-corona-new>

# Introduction and Assumptions

## Design

Below are the program’s pseudocode and flowchart.

### Pseudocode

PROGRAM CORONA

BEGIN

FUNCTION menu()

PRINT “Welcome”

PRINT “1. Register new patient”

PRINT “2. Test patients”

PRINT “3. Modify patient status”

PRINT “4. Statistics”

PRINT “5. Search”

PRINT “6. Exit”

PRINT “Choose option : “

READ option

IF option == “1”

registerPatient()

ELIF option == “2”

testPatient()

ELIF option == “3”

modifyPatient()

ELIF option == “4”

getStatPatient()

ELIF option == “5”

searchPatient()

ELIF option == “6”

menu()

ELSE

PRINT “Invalid option, please try again”

menu()

FUNCTION registerPatient()

PRINT "Enter patient name: "

READ patientName

PRINT "Enter patient ID: "

READ patientID

PRINT "Enter patient email: "

READ patientMail

PRINT "== Groups List =="

PRINT "ATO: Asymptomatic Travelled Overseas"

PRINT "ACC: Asymptomatic Close Contact"

PRINT "AEO: Asymptomatic Event Outbreak"

PRINT "SID: Symptomatic Individual"

PRINT "AHS: Asymptomatic Hospital Staff"

PRINT "Enter patient group: "

READ patientGroup

PRINT "Any past medical conditions? (A for yes, B for no): "

READ patientCondition

PRINT "Enter patient zone (A, B, C, or D): "

READ patientZone

patientData = patientName+";"+patientID+";"+patientMail+";"+patientGroup+";"+patientZone+";"+"N"+";"+"N"+";"+"N"+";"+"N"+";"+"0"+";"+"N"+";"+patientCondition+";"+"X"

OPEN patient.txt

APPEND patientData

exitMenu()

FUNCTION testPatient()

OPEN patient.txt AS my\_file

content = READLINES my\_file

PRINT “Enter patient ID: “

READ patientID

counter = 0

FOR element IN content

newContent = SPLIT “;” content[counter]

DELETE newContent[-1]

IF newContent[1] == patientID

PRINT “Testing patient “ + newContent[0]

PRINT “Enter test result: “

PRINT “1. Positive”

PRINT “2. Negative”

PRINT “Choose option: “

patientTest, testSolution, testResult, positive = testLaboratory(newContent[3], newContent[5], testResult)

IF testSolution == “QHNF” AND newContent[11] == “1”

newSolution = “quarantine in ICU, (No follow-up test required)”

ELIF testSolution == “QHNF” AND newContent[11] == “2”

newSolution = “quarantine in normal ward, (No follow-up test required)”

ELIF testSolution == “HQNF”

newSolution = “home quarantine, (No follow-up test required)”

ELIF testSolution == “QDFR”

newSolution = “quarantine in designated centres, (Follow up test required)”

ELIF testSolution == “HQFR”

newSolution = “home quarantine, (Follow up test required)”

ELIF testSolution == “RU”

newSolution = “go home and reunite with family, (No follow-up test required)”

ELIF testSolution == “CW”

newSolution = “continue working, (No follow-up test required)”

PRINT “Patient should do “ + newSolution

IF patientTest == “N”

IF testResult == “1”

newContent[5] = “F”

newContent[6] = “1”

PRINT “Enter case ID: “

READ newContent[9]

newContent[10] = “A”

ELIF testResult == “0”

newContent[5] = “t1”

newContent[6] = “0”

ELSE

PRINT “Incorrect test result, returning to main menu”

menu()

ELIF patientTest == “t1”

IF testResult == “1”

newContent[5] = “F”

newContent[7] = “1”

PRINT “Enter case ID:”

READ newContent[9]

newContent[10] = “A”

ELIF testResult == “0”

newContent[5] = “t2”

newContent[7] = “0”

ELSE

PRINT “Incorrect test result, returning to main menu”

menu()

ELIF patientTest == “t2”

IF testResult == “1”

newContent[5] = “F”

newContent[8] = “1”

PRINT “Enter case ID:”

READ newContent[9]

newContent[10] = “A”

ELIF testResult == “0”

newContent[5] = “t2”

newContent[8] = “0”

ELSE

PRINT “Incorrect test result, returning to main menu”

menu()

newPatientData = patientBuilder(newContent)

newIndex = int(newContent[1]) – 1

content[newIndex] = newPatientData

OPEN patient.txt AS f METHOD w

counterIn = 0

FOR items IN content

f WRITELINES (content[counterIn])

counterIn = counterIn + 1

CLOSE f

BREAK

counter = counter + 1

CLOSE my\_file

exitMenu()

FUNCTION modifyPatient()

PRINT “Available options”

PRINT “1. Set all non-active positive patients’ status to active”

PRINT “2. Modify patient status”

PRINT “3. Return to main menu”

PRINT “Choose option: “

READ option

IF option == “1”

modifyFeatureA()

ELIF option == “2”

modifyFeatureB()

ELIF option == “3”

menu()

ELSE

PRINT “Wrong option, please choose from the”

modifyPatient()

FUNCTION getStatPatient()

OPEN patient.txt AS my\_file METHOD r

content = READLINES my\_file

counter = 0

totalTest = 0

patientTested = 0

patientRecovered = 0

patientPositiveGroupATO = 0

patientPositiveGroupACC = 0

patientPositiveGroupAEO = 0

patientPositiveGroupSID = 0

patientPositiveGroupAHS = 0

ActiveCaseZoneA = 0

ActiveCaseZoneB = 0

ActiveCaseZoneC = 0

ActiveCaseZoneD = 0

FOR element IN content

newContent = SPLIT “;” content[counter]

DELETE newContent[-1]

IF newContent[5] != “N”

patientTested = patientTested + 1

IF newContent[6] != “N”

totalTest = totalTest + 1

IF newContent[7] != “N”

totalTest = totalTest + 1

IF newContent[8] != “N”

totalTest = totalTest + 1

IF newContent[10] == “R”

patientRecovered = patientRecovered + 1

IF newContent[3] == “ATO” AND newContent[6] == “1”

patientPositiveGroupATO = patientPositiveGroupATO + 1

IF newContent[3] == “ATO” AND newContent[7] == “1”

patientPositiveGroupATO = patientPositiveGroupATO + 1

IF newContent[3] == “ATO” AND newContent[8] == “1”

patientPositiveGroupATO = patientPositiveGroupATO + 1

IF newContent[3] == “ACC” AND newContent[6] == “1”

patientPositiveGroupACC = patientPositiveGroupACC + 1

IF newContent[3] == “ACC” AND newContent[7] == “1”

patientPositiveGroupACC = patientPositiveGroupACC + 1

IF newContent[3] == “ACC” AND newContent[8] == “1”

patientPositiveGroupACC = patientPositiveGroupACC + 1

IF newContent[3] == “AEO” AND newContent[6] == “1”

patientPositiveGroupAEO = patientPositiveGroupAEO + 1

IF newContent[3] == “AEO” AND newContent[7] == “1”

patientPositiveGroupAEO = patientPositiveGroupAEO + 1

IF newContent[3] == “AEO” AND newContent[8] == “1”

patientPositiveGroupAEO = patientPositiveGroupAEO + 1

IF newContent[3] == “SID” AND newContent[6] == “1”

patientPositiveGroupSID = patientPositiveGroupSID + 1

IF newContent[3] == “SID” AND newContent[7] == “1”

patientPositiveGroupSID = patientPositiveGroupSID + 1

IF newContent[3] == “SID” AND newContent[8] == “1”

patientPositiveGroupSID = patientPositiveGroupSID + 1

IF newContent[3] == “AHS” AND newContent[6] == “1”

patientPositiveGroupAHS = patientPositiveGroupAHS + 1

IF newContent[3] == “AHS” AND newContent[7] == “1”

patientPositiveGroupAHS = patientPositiveGroupAHS + 1

IF newContent[3] == “AHS” AND newContent[8] == “1”

patientPositiveGroupAHS = patientPositiveGroupAHS + 1

IF newContent[4] == “A” AND newContent[10] == “A”

ActiveCaseZoneA = ActiveCaseZoneA + 1

IF newContent[4] == “B” AND newContent[10] == “A”

ActiveCaseZoneB = ActiveCaseZoneB + 1

IF newContent[4] == “C” AND newContent[10] == “A”

ActiveCaseZoneC = ActiveCaseZoneC + 1

IF newContent[4] == “D” AND newContent[10] == “A”

ActiveCaseZoneD = ActiveCaseZoneD + 1

counter = counter + 1

totalTest = str(totalTest)

patientTested = str(patientTested)

patientRecovered = str(patientRecovered)

patientPositiveGroupATO = str(patientPositiveGroupATO)

patientPositiveGroupACC = str(patientPositiveGroupACC)

patientPositiveGroupAEO = str(patientPositiveGroupAEO)

patientPositiveGroupSID = str(patientPositiveGroupSID)

patientPositiveGroupAHS = str(patientPositiveGroupAHS)

ActiveCaseZoneA = str(ActiveCaseZoneA)

ActiveCaseZoneB = str(ActiveCaseZoneB)

ActiveCaseZoneC = str(ActiveCaseZoneC)

ActiveCaseZoneD = str(ActiveCaseZoneD)

PRINT “”

PRINT “Total tests: “ + totalTest

PRINT “”

PRINT “Tested patients: “ + patientTested

PRINT “”

PRINT “Patients recovered: “ + patientRecovered

PRINT “”

PRINT “Asymptomatic Travelled Overseas tested positive: “ + patientPositiveGroupATO

PRINT “Asymptomatic Close Contact tested positive: “ + patientPositiveGroupACC

PRINT “Asymptomatic Event Outbreak tested positive: “ + patientPositiveGroupAEO

PRINT “Symptomatic Individuals tested positive: “ + patientPositiveGroupSID

PRINT “Asymptomatic Hospital Staffs tested positive: “ + patientPositiveGroupAHS

PRINT “”

PRINT “Active cases in Zone A: “ + ActiveCaseZoneA

PRINT “Active cases in Zone B: “ + ActiveCaseZoneB

PRINT “Active cases in Zone C: “ + ActiveCaseZoneC

PRINT “Active cases in Zone D: “ + ActiveCaseZoneD

PRINT “”

CLOSE my\_file

exitMenu()

FUNCTION searchPatient()

PRINT “Available options”

PRINT “1. Patient records”

PRINT “2. Case status”

PRINT “3. Deceased patient records”

PRINT “4. Return to main menu”

PRINT “Choose option: “

READ option

IF option == “1”

searchFeatureA()

ELIF option == “2”

searchFeatureB()

ELIF option == “3”

searchFeatureC()

ELIF option == “4”

menu()

ELSE

PRINT “Wrong option, please choose from the”

modifyPatient()

FUNCTION exitMenu()

PRINT “Return to menu?”

PRINT “1. Return to menu”

PRINT “2. Exit program”

PRINT “Choose option: “

READ exitOption

IF exitOption == “1”

menu()

ELIF exitOption == “2”

PRINT “Goodbye and have a nice day.”

ELSE

PRINT “You entered the wrong option but goodbye and have a nice day ;)”

FUNCTION patientBuilder(patient)

patientData = patient[0] +";"+ patient[1] +";"+ patient[2] +";"+ patient[3] +";"+ patient[4]+";"+patient[5]+";"+patient[6]+";"+patient[7]+";"+patient[8]+";"+patient[9]+";"+patient[10]+";"+patient[11]+";"+"X\n"

RETURN patientData

FUNCTION testLaboratory(patientGroup, patientTest, testResult)

IF testResult == “2”

testResult = “0”

IF patientTest == “N”

IF testResult == “1”

IF patientGroup == “ATO”

testSolution = “QHNF”

ELIF patientGroup == “ACC”

testSolution = “QHNF”

ELIF patientGroup == “AEO”

testSolution = “QHNF”

ELIF patientGroup == “SID”

testSolution = “QHNF”

ELIF patientGroup == “AHS”

testSolution = “HQNF”

ELSE

PRINT “Invalid Group”

positive = 1

ELIF testResult == “0”

IF patientGroup == “ATO”

testSolution = “QDFR”

ELIF patientGroup == “ACC”

testSolution = “QDFR”

ELIF patientGroup == “AEO”

testSolution = “QDFR”

ELIF patientGroup == “SID”

testSolution = “HQFR”

ELIF patientGroup == “AHS”

testSolution = “CWFR”

ELSE

PRINT “Invalid Group”

positive = 0

ELIF patientTest == “t1”

IF testResult == “1”

IF patientGroup == “ATO”

testSolution = “QHNF”

ELIF patientGroup == “ACC”

testSolution = “QHNF”

ELIF patientGroup == “AEO”

testSolution = “QHNF”

ELIF patientGroup == “SID”

testSolution = “QHNF”

ELIF patientGroup == “AHS”

testSolution = “HQNF”

ELSE

PRINT “Invalid Group”

positive = 1

ELIF testResult == “0”

IF patientGroup == “ATO”

testSolution = “QDFR”

ELIF patientGroup == “ACC”

testSolution = “QDFR”

ELIF patientGroup == “AEO”

testSolution = “QDFR”

ELIF patientGroup == “SID”

testSolution = “HQFR”

ELIF patientGroup == “AHS”

testSolution = “CWFR”

ELSE

PRINT “Invalid Group”

positive = 0

ELIF patientTest == “t2”

IF testResult == “1”

IF patientGroup == “ATO”

testSolution = “QHNF”

ELIF patientGroup == “ACC”

testSolution = “QHNF”

ELIF patientGroup == “AEO”

testSolution = “QHNF”

ELIF patientGroup == “SID”

testSolution = “QHNF”

ELIF patientGroup == “AHS”

testSolution = “HQNF”

ELSE

PRINT “Invalid Group”

positive = 1

ELIF testResult == “0”

IF patientGroup == “ATO”

testSolution = “RU”

ELIF patientGroup == “ACC”

testSolution = “RU”

ELIF patientGroup == “AEO”

testSolution = “RU”

ELIF patientGroup == “SID”

testSolution = “RU”

ELIF patientGroup == “AHS”

testSolution = “CW”

ELSE

PRINT “Invalid Group”

positive = 0

ELSE

PRINT “No tests required for this patient, returning to main menu”

testSolution = 0

positive = 0

menu()

RETURN patientTest, testSolution, testResult, positive

FUNCTION modifyFeatureA()

OPEN patient.txt AS my\_file METHOD r

CONTENT = READLINES my\_file

counter = 0

FOR element IN content

newContent = SPLIT “;” content[counter]

DELETE newContent[-1]

IF newContent[6] == “1”

IF newContent[10] == “N”

newContent[10] = “A”

IF newContent[7] == “1”

IF newContent[10] == “N”

newContent[10] = “A”

IF newContent[8] == “1”

IF newContent[10] == “N”

newContent[10] = “A”

newPatientData = patientBuilder(newContent)

newIndex = int(newContent[1]) – 1

content[newIndex] = newPatientData

OPEN patient.txt AS f METHOD w

counterIn = 0

FOR items IN content

f WRITELINES content[counterIn]

counterIn = counterIn + 1

CLOSE f

counter = counter + 1

PRINT “All active patients’ status has been changed to Active!”

exitMenu()

FUNCTION modifyFeatureB

OPEN patient.txt AS my\_file METHOD r

content = READLINES my\_file

PRINT “Enter patient ID: “

READ patientID

counter = 0

FOR element IN content

newContent = SPLIT “;” content[counter]

DELETE newContent[-1]

IF newContent[1] == patientID

PRINT “Modify patient “ + newContent[0] + “’s status to?”

PRINT “1. Recovered”

PRINT “2. Deceased”

PRINT “3. Return to main menu”

PRINT “Choose option: “

READ option

IF option == “1”

newContent[10] = “R”

ELIF option == “2”

newContent[10] = “D”

ELIF option == “3”

menu()

newPatientData = patientBuilder(newContent)

newIndex = int(newContent[1]) – 1

content[newIndex] = newPatientData

OPEN patient.txt AS f METHOD w

counterIn = 0

FOR items IN content

f WRITELINES content[counterIn]

counterIn = counterIn + 1

CLOSE f

exitMenu()

counter = counter + 1

FUNCTION searchFeatureA

OPEN patient.txt AS my\_file METHOD r

content = READLINES my\_file

PRINT “Available options”

PRINT “1. Search by name”

PRINT “2. Search by ID”

PRINT “3. Go back to the main menu”

PRINT “Choose option: “

READ option

IF option == “1”

PRINT “Enter patient name (case sensitive): “

READ patientName

counter = 0

FOR element IN content

newContent = SPLIT “;” content[counter]

DELETE newContent[-1]

IF newContent[0] == patientName

IF newContent[10] == “N”

status = “Healthy”

ELIF newContent[10] == “A”

status = “Infected”

ELIF newContent[10] == “R”

status = “Recovered”

ELIF newContent[10] == “D”

status = “Deceased”

ELSE

status = “status not found”

IF newContent[11] == “1”

cond = “Yes”

ELIF newContent[11] == “2”

cond = “No”

ELSE

cond = “condition not found”

PRINT "---------------------------------"

PRINT “Patient name: “ + newContent[0]

PRINT “Patient ID: “ + newContent[1]

PRINT “Patient mail: “ + newContent[2]

PRINT “Patient group: “ + newContent[3]

PRINT “Patient zone: “ + newContent[4]

PRINT “Patient status: “ + status

PRINT “Have any past medical conditions: “ + cond

PRINT "---------------------------------"

exitMenu()

counter = counter + 1

ELIF option == “2”

PRINT “Enter patient ID: “

READ patientID

counter = 0

FOR element IN content

newContent = SPLIT “;” content[counter]

DELETE newContent[-1]

IF newContent[1] == patientID

IF newContent[10] == “N”

status = “Healthy”

ELIF newContent[10] == “A”

status = “Infected”

ELIF newContent[10] == “R”

status = “Recovered”

ELIF newContent[10] == “D”

status = “Deceased”

ELSE

status = “status not found”

IF newContent[11] == “1”

cond = “Yes”

ELIF newContent[11] == “2”

cond = “No”

ELSE

cond = “condition not found”

PRINT "---------------------------------"

PRINT “Patient name: “ + newContent[0]

PRINT “Patient ID: “ + newContent[1]

PRINT “Patient mail: “ + newContent[2]

PRINT “Patient group: “ + newContent[3]

PRINT “Patient zone: “ + newContent[4]

PRINT “Patient status: “ + status

PRINT “Have any past medical conditions: “ + cond

PRINT "---------------------------------"

exitMenu()

counter = counter + 1

ELIF option == “3”

menu()

FUNCTION searchFeatureB

OPEN patient.txt AS my\_file METHOD r

content = READLINES my\_file

PRINT “Enter case ID: “

READ caseID

counter = 0

FOR element IN content

newContent = SPLIT “;” newContent[counter]

DELETE newContent[-1]

IF newContent[9] == caseID

IF newContent[10] == “A”

status = “Active”

ELIF newConent[10] == “R”

status = “Recovered”

ELIF newContent[10] == “D”

status = “Deceased”

ELSE

status = “status not found

PRINT "---------------------------------"

PRINT “Patient status for case “ + newContent[9] + “ is “ + status

PRINT "---------------------------------"

exitMenu()

counter = counter + 1

FUNCTION searchFeatureC

OPEN patient.txt AS my\_file METHOD r

content = READLINES my\_file

counter = 0

FOR element IN content

newContent = SPLIT “;” newContent[counter]

DELETE newContent[-1]

IF newContent[10] == “D”

IF newContent[11] == “1”

cond = “Yes”

ELIF newContent[11] == “2”

cond = “No”

ELSE

cond = “condition not found”

PRINT "---------------------------------"

PRINT “Patient name: “ + newContent[0]

PRINT “Patient ID: “ + newContent[1]

PRINT “Patient mail: “ + newContent[2]

PRINT “Patient group: “ + newContent[3]

PRINT “Patient zone: “ + newContent[4]

PRINT “Patient case ID: “ + newContent[9]

PRINT “Have any past medical conditions: “ + cond

PRINT "---------------------------------"

counter = counter + 1

exitMenu()

menu()

END

### Flowchart

A picture containing clock, game, table, drawing

Description automatically generated

Figure 2.2.1 Global flowchart of the whole program

A close up of a map

Description automatically generated

Figure 2.2.2 Flowchart of the menu() function

A picture containing table, bird

Description automatically generated

Figure 2.2.3 Flowchart of the registerPatient() function

A close up of text on a black background

Description automatically generated

Figure 2.2.4 Flowchart of the testPatient() function

A close up of text on a black background

Description automatically generated

Figure 2.2.5 Flowchart of the modifyPatient() function

A close up of a map

Description automatically generated

Figure 2.2.6 Flowchart of the getStatPatient() function

A close up of a map

Description automatically generated

Figure 2.2.7 Flowchart of the searchPatient() function

A close up of a logo

Description automatically generated

Figure 2.2.8 Flowchart of the exitMenu() function

A picture containing clock, table

Description automatically generated

Figure 2.2.9 Flowchart of the patientBuilder() function

A close up of a clock

Description automatically generated

Figure 2.2.10 Flowchart of the testLaboratory() function

A close up of text on a black background

Description automatically generated

Figure 2.2.11 Flowchart of the modifyFeatureA function

A close up of a map

Description automatically generated

Figure 2.2.12 Flowchart of the modifyFeatureB function

A close up of a map

Description automatically generated

Figure 2.2.13 Flowchart of the searchFeatureA function

A close up of a map

Description automatically generated

Figure 2.2.14 Flowchart of the searchFeatureB function

A close up of text on a white background

Description automatically generated

Figure 2.2.15 Flowchart of the searchFeatureC function

# Source Code with Explanation

## Variables

Normally, there are two types of variables: strings, and integers. There are other types as well but normally they can be related into either strings or integers. Strings store alphanumeric characters while integers store numbers. Arithmetic operations can be performed on integers while not on strings.

### String

String type variables are used a lot in this program. They are typically used to store patients’ data for writing or when a patient’s data is read from the database. An example will be the variable patientGroup in the registerPatient function. This string variable is used to store a patient’s group when registering as a new patient. Then, after all patient data has been inputted while registering, all these data are concatenated into one string variable patientData for storage purposes.

A screen shot of a social media post

Description automatically generated

Figure 3.1.1 String type variables used to store data

### Integers

Integer type variables are not used as much as string type variables however they play an important role in adding patient data entries into the main database list. Here, the string type variable newContent[1] is converted into an integer type variable so it can be used for an arithmetic operation. The result is then stored into the integer type variable newIndex. The variable newIndex is then used to locate the actual index of the patient in the list content[] for storage purposes. This is because newContent[1] is the patient’s ID, and to locate a patient’s index in the content[] list, the patient’s ID can simply be subtracted by 1 to obtain the patient’s index in the list content[].



Figure 3.1.2 The variable newContent[1] is converted to an integer for an arithmetic operation

## Control Structure

Control structures such as conditional statements are used to steer the program to the direction the programmer wants it to. For example, with an if statement, a programmer can execute certain lines of codes or functions when a certain condition is met. This is a very important part of most programs including this one.

### If

In this example, the If statement is used to filter out patient ID’s. Since this conditional statement is used in a loop. When the ID of the wanted patient is reached, the loop course of the program will shift into the conditional statement. In this example, when the condition of newContent[5] is not equal to “N”, the program will execute the lines inside the conditional statement. In this case, it increments the variable patientTested by 1.



Figure 3.2.1 An if statement is used to filter out patients with tests done on them.

### Elif

Elif, or commonly known as elseif statements in other programming languages, are basically if statements but with more conditions and branches for the program to execute. An example of the usage of elif statements in this program is when navigating through the menus in this program. In the main menu of the program, 6 options are given, but only one can be taken at a time. An if and 5 elif statements can solve this problem. The program will run different functions depending on the conditions met. When none of the conditions are met, the program can run a backup lines of code. In this example, it starts again and prompts the user to enter an appropriate option.

A picture containing table

Description automatically generated

Figure 3.2.2 Elif statements used to navigate the user through the program

### Nested Elif

It is also possible to write an elif statement inside another elif statement. This multiplies the possibilities of conditional statements even more. In this example, it is used in the testLaboratory function which is used to give out test solutions depending on the amount tests done on patients, patients’ group, and test result. In this case, it first checks on the amount of tests done on the patient to see whether it should do the first, second, or third test. If a patient did all 3 tests, the function will reject the patient and not test the patient.

A picture containing table, sitting, screen, holding

Description automatically generated

Figure 3.2.3 A nested elif statement that is used to check a patient’s test number and result.

## Loops

Loops are very useful when a certain line of code is needed to be ran multiple times. Be it print statements, increments, or for loops that browse through lists. There are two main types of loops: For loops, and while loops. The difference is for loops loop for a specific amount of time while while loops loop until its running condition is no longer met. This program uses for loops only due to them normally being used to browse through lists for a specific data.

### For loops

For loops are loops that loop for a specific amount of time. Be it the amount is inputted by the user, or the amount of elements in a specific list. For loops are really useful and in this program its role is very significant. This is due to the nature of the data storage method, where the data is stored in lists. For loops can be used to go through the list and find the data that we need to proceed. It can be a patient’s name, a patient’s ID, test results, or whether that patient has a preexisting medical condition or not. Below it is used to browse through the list content

A screenshot of a cell phone

Description automatically generated

Figure 3.3.1 A for loop is used to browse through the indexes of the list content[].

## List

Lists are used to store multiple items under one name for processing. In this program, lists play a very significant role due it being the main method of patient data storage. There are two main lists in this program: content which stores all patient data, and newContent which is basically the indexes of content that is split into multiple pieces for processing. Below is an example of the content’s of the patient.txt file being assigned to the list content. And then the indexes of the list content being assigned to the variable newContent. And then the variable is split by the semicolons, turning it into processable patient data. Where all the data including ID, and test results can be obtained by simply calling the list index.

Screen of a cell phone

Description automatically generated

Figure 3.4.1 Lists being used in the program to store patient data.

Each index of the list newContent, stores a specific data. For example, the index 0 always stores the patient’s name, while the index 7 will always store the result of the second test. Below is the format of a patient data

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Name | Name | ID | Email | Group | Zone | Tests | T1 Result | T2 Result | T3 Result | Case ID | Case Status | Preexisting condition | Extra index |
| Example | Reimu | 17 | [reimu@mail.com](mailto:reimu@mail.com) | ATO | A | F | 0 | 1 | N | 8 | R | 2 | X |

Test number will be N by default. When a test is done this will change to t1, t2, or F in that specific order. Patients with the value F can not be tested. Test results will be N by default as well. This can change to 1 or 0 depending on the test result. Case status will be N by default. When a patient tests positive, the status will automatically change to A. This can later be modified to R or D depending on the patient’s status. The extra index 12 is there only to be deleted to remove the \n that appears naturally in a new line.

## Functions

Functions are there to make a programmer’s life easier. They are basically snippets of code that can be called and used multiple times in other parts of the program without having to write the code again. This is really useful when a specific action has to be executed multiple times or just to easily navigate through the program. Below is an example of the function menu. Note that the whole program is just functions and one line of function call at the end. And that function will then proceed to call other functions.

A screenshot of a cell phone

Description automatically generated

Figure 3.5.1 The function menu.

## Functions without parameters

Functions can be executed by simply calling it. An example will be the function menu which is the backbone of the program. It allows the user to call other functions and navigate through the program with ease.



Figure 3.5.2 The function menu being called at the bottom of the program, below all the function definitions.

### Functions with parameters

Functions are not just a static line of code that can be called anytime. Some functions have parameters. They take in arguments, and process those arguments they take in. This kind of functions are used in the patient testing part of the program due to the function needing patient information to be able to test a patient. Below is the function testLaboratory taking 3 arguments: the patient’s group, tests done, and test result.

A screenshot of a cell phone

Description automatically generated

Figure 3.5.3 The function testLaboratory taking 3 arguments to be processed.

### Functions with returns

After processing, some functions might need to return some values to the line it was called. In this program the testLaboratory takes in arguments, process those arguments, and then return multiple values such as the patient test number, solution for the patient based on test result, and the test result itself. Below is the line in which the function testLaboratory returns four values.



Figure 3.5.4 The function testLaboratory returning four variables of various types.

# Screenshots

A screen shot of a social media post

Description automatically generated

Figure 4.1 Main menu.

A screen shot of a computer

Description automatically generated

Figure 4.2 Patient registration.

A close up of a logo

Description automatically generated

Figure 4.3 A patient tests positive for Corona

A screenshot of a cell phone

Description automatically generated

Figure 4.4 A patient with pre-existing condition tests positive for Corona

A screenshot of a cell phone

Description automatically generated

Figure 4.5 Modify patient menu



Figure 4.6 Changing all infected patient’s status to active

A screen shot of a social media post

Description automatically generated

Figure 4.7 A patient’s status modified from active to recovered

A screen shot of a computer

Description automatically generated

Figure 4.8 Statistics

A picture containing black, sitting, holding, table

Description automatically generated

Figure 4.9 Search menu

A picture containing black, holding, street, red

Description automatically generated

Figure 4.10 Patient records search menu.

A screenshot of a cell phone

Description automatically generated

Figure 4.11 Searching a patient’s records by name

A screenshot of a cell phone

Description automatically generated

Figure 4.12 Searching a patient’s records by ID

A close up of a screen

Description automatically generated

Figure 4.13 Searching for a case’s status by case ID

A screen shot of a smart phone

Description automatically generated

Figure 4.14 All deceased patients’ records

A close up of a screen

Description automatically generated

Figure 4.15 The exitMenu function

Screen of a cell phone screen with text

Description automatically generated

Figure 4.16 Returning to the main menu from the exitMenu function

A close up of a screen

Description automatically generated

Figure 4.17 Entering the wrong option in the exitMenu function

Screen of a cell phone

Description automatically generated

Figure 4.18 Exiting the program from the main menu

# Conclusion

In conclusion, just like most programs out there, this program will work as intended when used correctly. The program fulfills all the required features in the assignment question, and does it properly. The program’s graphics may be dull and plain however it is not part of the marking scheme therefore is not a focus during development.