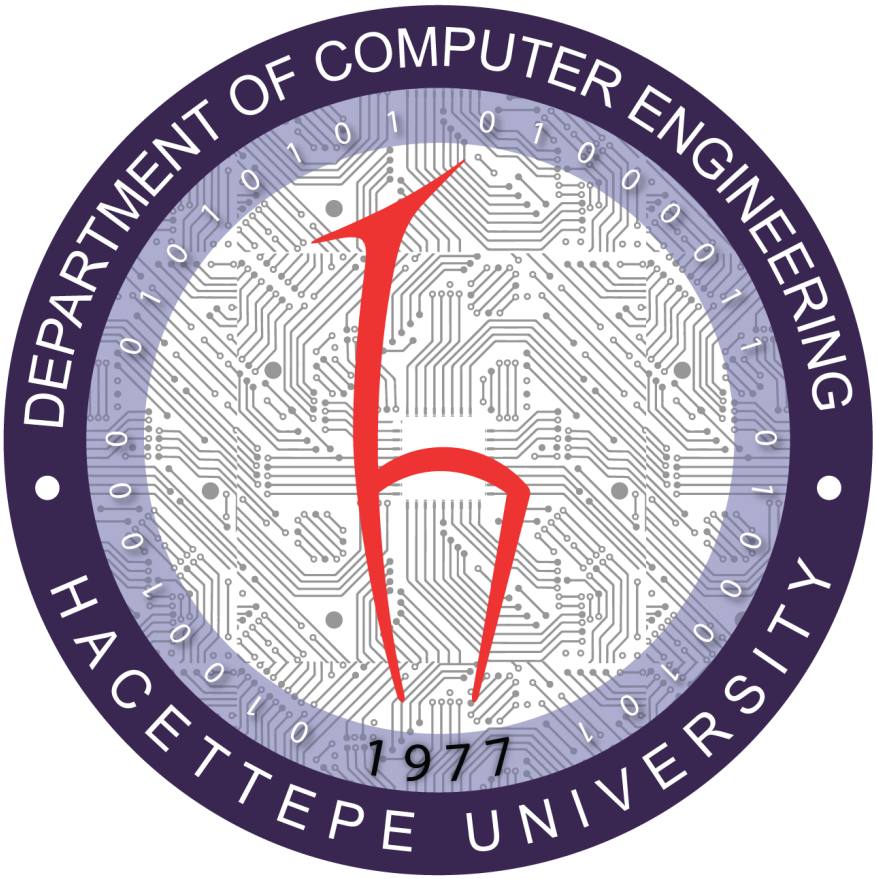
Hacettepe University

Department of Computer Science

BBM-101 Assignment 2 Report

Gökdeniz Şimşek – 2210356067

26.11.2022



ANALYSIS

In this section, I discuss the problem of the assignment and explain the aims of the project.

In this assignment, we are facing a problem in which we need to create functions for different tasks. To go into detail, we should create a patient assistance system that gives different results depending on the information entered and the desired function. The system will have basic information about patients, their diseases, and levels of disease, and will also calculate disease risks. The system will make recommendations for patients.

The primary goal of the assignment we were given was to use file operation commands. We must take all of the operations from a previously noted file and evaluate them in relation to the relevant data in the line. Then it needs to give an output with the desired result to a different file.

The most important problem in this assignment is to parse the lines inside the input file. We need to determine which function is the desired function in the line and give the correct output according to the data we have or that is given to us.

DESIGN

Find a Solution

Firstly, the problem should be considered in general, and necessary solutions should be sought. The fundamental functions required for the solution should be created by evaluating all possible errors. When the input file is examined, the main functions to be written are "create", "remove", "probability", "recommendation" and "list"

Getting Input from File

In order to get input from the file, the file should be opened with the "r" key, and the file should be evaluated separately for each line. In this case, the "readline" function can be used. The first thing to do when examining a line is to determine which function is on the line. Then the data in the line is directed to the relevant function, and output is created accordingly.

I selected the first three letters of each line to identify the function and compared them to the first three letters of the function names. If they are the same, the function in the line is the same. I also checked for a hidden "\n" at the end of the line so that the data can be used correctly. Because "\ n" should not be included in the data.

Create Function

The “create” command will add a new patient to the system. If it is determined that the function in the line entering the input command is "create", the part of the line after the create text should be taken as data. This data should be correctly separated and converted into a patient-specific list. Subsequently, each patient list created should be combined into a common list. However, it should be noted whether the patient entered in the "create" function has been registered before.

To get the data correctly, I checked to see if there was an "n" in the line. If there is, I have listed the information except for the function name and the last index in the line, separated by commas. Then I checked whether the resulting patient list was in the list of all patients. If not, I added it to the list and printed the output accordingly.

Remove Function

The “remove” command will remove an existing patient from the system. If it is determined that the function in the line entering the input command is "remove", the part of the line after the remove text should be taken as data. The data must be the patient whose information wants to be deleted. Then, it is checked whether the patient's data to be deleted is in the list of all patients. If any, everything related to the patient is deleted, and an output is given accordingly. If there is no patient data, an output is given accordingly.

To get the data correctly, I checked to see if there was an "n" in the line. If there is, I got the text on the line as data except for the function name and the last index in the line. I created a “for loop” to compare the values in the list of all patients with the data I received. It compares the 0th index of each individual patient list in the list with the patient name in the data I received, and if it matches one, it deletes the patient list with that data from the list of all patients. I set a different variable to "False" to evaluate whether the patient in the data is on the all-patients list. It gets the value "True" if the patient's name matches someone in the list. If the variable still has the value "False" at the end of the code, it will output accordingly.

Calculate-Probability Function

There are some processes that need to be done for the "probability" and "recommendation" commands. To make these two codes easier to understand, I created a separate code that includes these operations. The code calculates the probability of being sick based on the patient's known data (Disease Incidence and Diagnosis Accuracy).

Probability Function

The “probability” command will be used to calculate and show/help the actual fatality probability of a patient. If it is determined that the function in the line entering the input command is "probability", the part of the line after the probability text should be taken as data. The data must be the patient’s name whose fatality probability wants to be calculated. Then, it is checked whether the patient data for which the fatality probability is to be calculated is in the list of all patients. If the received data does not match the information in the list of all patients, the output is given accordingly. If there is matching information, the value of the " **Calculate-Probability** " function is found over the information of the matched patient, and the result is given as a percentage.

To get the data correctly, I checked to see if there was an "n" in the line. If there is, I got the text on the line as data except for the function name and the last index in the line. I created a “for loop” to compare the values in the list of all patients with the data I received. It compares the 0th index of each individual patient list in the list with the patient name in the data I received, and if it matches one, it performs the operations in the " **Calculate-Probability** " function using other data belonging to this patient. Also, in order to give the output as a percentage, I converted the result to a "string" value and chose the first 2 (ex. 80%) or 5 (ex. 33.33%) items according to whether or not it was a decimal number. I set a different variable to "False" to evaluate whether the patient in the data is on the all-patients list. It gets the value "True" if the patient's name matches someone in the list. If the variable still has the value "False" at the end of the code, it will output accordingly.

Recommendation Function

The “recommendation” command will be used to give a system recommendation to a patient about whether to have the treatment. If it is determined that the function in the line entering the input command is "recommendation", the part of the line after the recommendation text should be taken as data. The data should be the name of the patient to be advised on whether to seek treatment. Then it is checked whether the data of this patient is in the list of all patients. If the received data does not match the information in the list of all patients, the output is given accordingly. If there is matching information, the value of the " Calculate-Probability " function is found over the information of the matched patient. This value varies according to the patient's disease and disease level. The risk of treatment that the patient should receive is also important when the system is making recommendations. The risk of treatment is contained in the information given about the patient. If the probability of fatality calculated by the " **Calculate-Probability** " function is greater than the treatment risk, it is recommended that the patient be treated.

To get the data correctly, I checked to see if there was an "n" in the line. If there is, I got the text on the line as data except for the function name and the last index in the line. I created a “for loop” to compare the values in the list of all patients with the data I received. It compares the 0th index of each individual patient list in the list with the patient name in the data I received, and if it matches one patient, it applies the function with using other data belonging to this patient. Because the treatment risk data were unsuitable for comparison, I multiplied it by 100 to convert it to percentage data. Then, compared it to the fatality probability. If the risk of treatment is greater, it indicates that treatment should not be done, and the function gives output according to that. If not, it will give output about that treatment should be done. If not, it will give output for that treatment should be done. Also, I set a different variable to "False" to evaluate whether the patient in the data is on the all-patients list. It gets the value "True" if the patient's name matches someone in the list. If the variable still has the value "False" at the end of the code, it will output accordingly.

List Function

The “list” command will be used to list the complete information about all patients in the system. If it is determined that the function in the line entering the input command is "list", the list function works without receiving any data. While listing, some data must be written as a percentage; therefore, necessary actions are taken. Since tab spaces will be taken as a basis while listing, it should be checked whether there is any contrary situation, and action should be taken accordingly. After the necessary actions are taken for the errors, the function outputs the created list.

I created a for loop to list the people in the list of all the patients in order. In this way, I evaluated the lengths of all patient data separately. The length of the data affected the number of tab spaces that had to be used. For example, there should be an interval of 8 units from the starting point of the patient's name to the starting point of the data "Diagnosis Accuracy". For patient "Su" this is provided with two tab spaces, while for patient "Hayriye" one tab space should be used.

Output Function

The only task of the output function is to correctly print the results returned by the other functions into the previously created file. The output of each new function is written on a new line. All functions should give an output after running. Instead of writing output code for all these functions separately, I created a single function that generates all output.

Codes Outside of Define Function Blocks

Here I have defined the global values of a few variables to be used in different functions. In addition, here are the codes required to create the file to be output and to read the input file required to receive input.

Programmer Catalog

Codes Outside of Define Function Blocks

```
patients = []
ask_n = True     #checking for is sentence include \n
doc_aid_outs = open("doctors_aid_outputs.txt","w",encoding="utf-8")     #for creating output.txt document
doc_aid_outs.close()
doc_aid_file = open("doctors_aid_inputs.txt","r",encoding="utf-8")
line = doc_aid_file.readline()
input_doc()
doc_aid_file.close()
```

Since most of the code consists of functions, this is the part where the global values used in different functions are defined and the functions are called to make the program run.

Calling the input function is sufficient for all code to work because all the code inside the program is interdependent. In addition, the file where the input will be taken is read, and the file where the output will be written is created.

Input Function

```
def input_doc():
    global line, ask_n
    while True:
        first_three_letter = line[0:3]
        if "\n" in line:
            ask_n = True
        else:
            ask_n = False
        if first_three_letter == "cre":
            output_doc(create())
            line = doc_aid_file.readline()
        elif first_three_letter == "rem":
            output_doc(remove())
            line = doc_aid_file.readline()
        elif first_three_letter == "lis":
            output_doc(list())
            line = doc_aid_file.readline()
        elif first_three_letter == "pro":
            output_doc(probability())
            line = doc_aid_file.readline()
        elif first_three_letter == "rec":
            output_doc(recommendation())
            line = doc_aid_file.readline()
        else:
            break
```

The input function is the main place where we parse each line of the input file. Here, the function written on the line is determined, and accordingly, that function and the output function are called one after the other. Then, the "readline()" function runs and the new line is read. Here it also checks if there is "\n" at the end of the line and returns "True" to "ask_n" if it exists. The "ask_n" value will be used in other functions. When the input function cannot find a readable value in the line, the input function and all other processes are terminated.

Output Function

```
if first_three_letter == "cre":
    output_doc(create())
```
(From input function)

```
def output_doc(returning_value):
    with open("doctors_aid_outputs.txt", "a", encoding="utf-8") as doc_aid_outs:
        doc_aid_outs.write(returning_value)
```

As seen in the part of the input function above, the output function takes the outputs of other functions as values. Then it prints this output into the output file.

Create Function

```
def create():
    global create_done, creat_not_done
    if ask_n:
        infos = line[7:-1].split(", ")
    else:
        infos = line[7:].split(", ")
    if infos not in patients:
        patients.append(infos)
        create_done = "Patient {} is recorded.\n".format(infos[0])
        return create_done
    else:
        creat_not_done = "Patient {} cannot be recorded due to duplication.\n".format(infos[0])
        return creat_not_done
```

Here, firstly, the part to be taken as data is determined according to whether there is "\n" in the line or not. (This code will also be used in other functions.) Afterward, it is checked whether the patient-related data is in the system and if it is not registered in the system, it is recorded in the "patients" list. Finally, the function outputs according to the status.

Remove Function

```
def remove():
    global remove_done, remove_not_done
    if ask_n:
        pat_rem = line[7:-1]
    else:
        pat_rem = line[7:]
    patient2 = False     #for checking if the patient is in the patients list
    for i in range(len(patients)):
        patients_name = patients[i][0]
        if patients_name == pat_rem_:
            patient2 = True
            patients.pop(i)
            remove_done = "Patient {} is removed.\n".format(patients_name)
            return remove_done
        else:
            pass
    if patient2 == False:
        remove_not_done = "Patient {} cannot be removed due to absence.\n".format(pat_rem)
        return remove_not_done
```

After the requested data is received, it is checked whether that person is on the "patients" list. (The variable "patient2" is used here. It takes the value "False" at the beginning of the function. If the data received by the function does not match anyone in the list, it remains as "False" and outputs accordingly. This code will also be used in other functions.) If it matches a patient in the list, it deletes that person's information from the "patients" list. Then it outputs depending on the situation.

Calculate-Probability Function

```python
def calculate_probability(ratio):
    plc_den = ratio[3].find("/")
    num_rat = float(ratio[3][:plc_den])
    den_rat = float(ratio[3][plc_den+1:])
    ratio = float(ratio[1])
    ratio2 = 1 - ratio
    tolerance = ratio2 * den_rat
    real_ratio = tolerance + num_rat
    probability = num_rat/real_ratio*100
    return probability
```

This function is for calculating the fatality probability required for the probability and recommendation functions. It is called inside of probability and recommendation functions, and outputs probability of death.

Probability Function

```python
def probability():
    global probability_done, probability_not_done
    if ask_n:
        pat_pro = line[12:-1]
    else:
        pat_pro = line[12:]
    patient = False     #for checking if the patient is on the patients list
    for i in range(len(patients)):
        patients_name = patients[i][0]
        if patients_name == pat_pro:
            patient = True
            prob = str(calculate_probability(patients[i]))
            if int(float(prob)) == float(prob[:5]):
                prob = int(prob[:2])
            else:
                prob = float(prob[:5])
            probability_done = "Patient {} has a probability of {}% of having {}.\n".format(patients_name,prob,patients[i][2])
            return probability_done
        else:
            pass
    if patient == False:
        probability_not_done = "Probability for {} cannot be calculated due to absence.\n".format(pat_pro)
        return probability_not_done
```

It is checked whether the patient in the data received by the function from the row is in the "patients" list. If available, it sends patient information to the calculate-probability function. The fatality probability obtained without taking unnecessary fractions of the result is output as a percentage. If the patient cannot be found in the "patients" list, a printout is given accordingly.

Recommendation Function

```
def recommendation():
    global treatment, not_treatment, not_exist_treatment
    if ask_n:
        pat_rec = line[15:-1]
    else:
        pat_rec = line[15:]
    patient1 = False     #for checking if the patient is on the patients list
    for i in range(len(patients)):
        patients_name = patients[i][0]
        if patients_name == pat_rec:
            patient1 = True
            tre_risk = float(patients[i][5])*100
            if calculate_probability(patients[i]) >= tre_risk:
                treatment = "System suggests {} to have the treatment.\n".format(pat_rec)
                return treatment
            else:
                not_treatment = "System suggests {} NOT to have the treatment.\n".format(pat_rec)
                return not_treatment
    if patient1 == False:
        not_exist_treatment = "Recommendation for {} cannot be calculated due to absence.\n".format(pat_rec)
        return not_exist_treatment
```

It is checked whether the patient in the data received by the function from the row is in the "patients" list. If available, it sends patient information to the calculate-probability function. The treatment risk is calculated using the patient's data. A recommendation about whether the user should take the treatment is output by comparing the treatment risk and the probability of death. If the patient cannot be found in the "patients" list, the output is given accordingly.

List Function

```
def list():
    out_list = """Patient\t"+"Diagnosis\t"+"Disease\t\t\t"+"Disease\t\t"+"Treatment\t\t"+"
    Treatment\n"+"Name\t"+"Accuracy\t"+"Name\t\t\t"+"Incidence\t"+"Name\t\t\t"+"Risk\n"+73*"-"+"\n"""
    for i in range(len(patients)):
        per_accuracy = str(float(patients[i][1])*100)
        if len(per_accuracy) == 4:
            per_accuracy = per_accuracy + "0"
        per_accuracy = per_accuracy + "%"
        per_risk = str(int(float(patients[i][5])*100))+"%"
        if len(patients[i][0]) < 4:
            out_list = out_list + patients[i][0]+"\t\t"+per_accuracy+"\t\t"
            if len(patients[i][2]) < 12:
                out_list = out_list + patients[i][2]+"\t\t"+patients[i][3]+"\t"
                if len(patients[i][4]) < 8:
                    out_list = out_list + patients[i][4]+"\t\t\t"+per_risk+"\n"
                elif 8 <= len(patients[i][4]) < 12:
                    out_list = out_list + patients[i][4] + "\t\t" + per_risk + "\n"
                elif 12 <= len(patients[i][4]) < 16:
                    out_list = out_list + patients[i][4] + "\t" + per_risk + "\n"
                else:
                    out_list = out_list + patients[i][4] + per_risk + "\n"
            else:
                out_list = out_list + patients[i][2]+"\t"+patients[i][3]+"\t"
                if len(patients[i][4]) < 8:
                    out_list = out_list + patients[i][4] + "\t\t\t" + per_risk + "\n"
                elif 8 <= len(patients[i][4]) < 12:
                    out_list = out_list + patients[i][4] + "\t\t" + per_risk + "\n"
                elif 12 <= len(patients[i][4]) < 16:
                    out_list = out_list + patients[i][4] + "\t" + per_risk + "\n"
                else:
                    out_list = out_list + patients[i][4] + per_risk + "\n"
```

```
        else:
            out_list = out_list + patients[i][0]+"\t"+per_accuracy+"\t\t"
            if len(patients[i][2]) < 12:
                out_list = out_list + patients[i][2] + "\t\t" + patients[i][3] + "\t"
                if len(patients[i][4]) < 8:
                    out_list = out_list + patients[i][4] + "\t\t\t" + per_risk + "\n"
                elif 8 <= len(patients[i][4]) < 12:
                    out_list = out_list + patients[i][4] + "\t\t" + per_risk + "\n"
                elif 12 <= len(patients[i][4]) < 16:
                    out_list = out_list + patients[i][4] + "\t" + per_risk + "\n"
                else:
                    out_list = out_list + patients[i][4] + per_risk + "\n"
            else:
                out_list = out_list + patients[i][2] + "\t" + patients[i][3] + "\t"
                if len(patients[i][4]) < 8:
                    out_list = out_list + patients[i][4] + "\t\t\t" + per_risk + "\n"
                elif 8 <= len(patients[i][4]) < 12:
                    out_list = out_list + patients[i][4] + "\t\t" + per_risk + "\n"
                elif 12 <= len(patients[i][4]) < 16:
                    out_list = out_list + patients[i][4] + "\t" + per_risk + "\n"
                else:
                    out_list = out_list + patients[i][4] + per_risk + "\n"
    return out_list
```

The function ensures that all patients in the "patients" list are listed with their data. The important point here is the tab spaces that need to be set in order to output a list regularly. Depending on the length of the patient data, the required number of "\t" should be used. After considering all the possibilities and setting the tab spaces, the list is sent to the output function as a string.

Time Spent

For analyzing: 1 and half hours

For designing: 2 hours

For implementing: 4 and half hours

For testing: 1 hour

For reporting: 5 hours

User Catalog

The program generally consists of functions. And since the purpose of the function is to take information from one file, edit it, and print it to another file, there is not much that the user can change or enter input while using the program.

```
patients = []
ask_n = True     #checking for is sentence include \n
doc_aid_outs = open("doctors_aid_outputs.txt","w",encoding="utf-8")     #for creating output.txt document
doc_aid_outs.close()
doc_aid_file = open("doctors_aid_inputs.txt","r",encoding="utf-8")
line = doc_aid_file.readline()
input_doc()
doc_aid_file.close()
```

Here, in codes outside of define function blocks, a few can change data. These are the name of the input file and the name of the output file.

Evaluation	Points	Evaluate Yourself / Guess Grading
Indented and Readable Codes	5	5
Using Meaningful Naming	5	5
Using Explanatory Comments	5	4
Efficiency (avoiding unnecessary actions)	5	5
Function Usage	25	25
Correctness	35	35
Report	20	18
There are several negative evaluations	...	...