BBM 103 Introduction to Programming Laboratory 1

Assignment 2: Doctor's Aid

2210356032

İbrahim Emek

24.11.2022

Contents:

Analysis:

In this assignment we have 3 main stage:

1- Reading input file
2- Process input file
3- Giving outputs to a file.

At the first stage in order to read input file we need to use basic I/O functions. If we read all file it would be hard to process them. We can read each line by using readline() function. By this way, we only need to process each line.

At the second stage we need to process each line. Each line’s first word is telling us to what to do, so we need to read the first word, and use that word to call the function it corresponds. Then, we can split rest sentence by split() method from commas, and it would be a list, and we can use this list’s items afterwards. In order to use these items we need to define functions that use them and do things that need to be done.

At the third stage we should use I/O functions again. We should take the results from functions at second stage, and save them to the output file.

Design:

Our purpose:

Our purpose is to create a system to help to doctor in his decisions. In this system we need a nested list to save the data, so let’s start with creating with an empty list patients_info to use it later.

Opening input file:

We need to take inputs from input file. We should open, and read the contents of the doctors_aid_inputs file. We can open it by using with open() function. We should read “r” and encoding = “utf-8” between the brackets. “r” means to read the file, encoding = “utf-8” provides us to use different language’s characters.

Reading input file:

Now, we need to read the input file. We can use read() or readline() methods. If we use read() method, it would be hard to process all the data, so we should use readline() method. We need to read every line, and process them, so we should use while loop. We can use infinite while loop which reads a new line in each cycle. If it reaches an empty line we can use break keyword, and we exit the loop. By this way, we can read every line until the blank line, and in each cycle, we can read a new line and process it.

Read_input_file() function:

1-Understanding what to do in each line:

Now, we have a line which wait us to process it. First, we need to understand what it wait us to do. In each line, first word tells us to what to do, so we need to use it. We can reach it by using substring the line. It starts at 0, and stops at first whitespace. We can find the first location of whitespace by using .find() method. It finds the first occurrence of the specified value. We can substring the line now by using string[starting: ending]. Starting point is 0, ending point is whitespace location. Now, this is our command.

2-Using commands:

Each command tells to us what to do. Now we should look what command tells us.

3-Using rest sentence:

We need to reach the rest sentence, and use them. We should substring the rest sentence from line, and separate them by commas(.split(",")). Then, we have a list of items which we will use in the functions. In order to use them easily on the other functions, lets create a global variable list_of_items and assign the list to that variable.

Creating other functions:

Now, we have commands. We should define functions, and call them if commands tells us to do. For example if command tells us to create, we should call create function after we define it.

Create() function:

If command is create, we need to call the create() function, but firstly, let's define it. Create command tells us to add a new patient to the system. To add a new patient, we need the items' list which I created at using rest sentence step. We can simply add this list to system by using patients_info.append(list_of_items) code. Before that, we should look if someone in that name is created before. If it created, we shouldn't add it again. We can use for loop in range of length of patients_info list. This provides us to loop added person number time. In each cycle, we can look to each item on the patients_info and if an item is same with which will we add, the loop can break and we don't create again. If there is no item same with which will we add, we can append it.

Delete() function

If command is remove, we need to call the delete() function,(remove() is method, so I used delete() instead of remove() to avoid confusion) but firstly, let's define it. Remove command tells us to remove an added patient from the system. To remove a patient, we should look each item in patients_info list, and if patient's name is in one of these items, we should remove this item. We can use for loop in range for look each item, and if it finds, we can remove that item. If it doesn't find, we can't delete item due to absence.

Probability() function

If command is probability, we need to call the probability() function. We call the probability function to calculate and show/help the actual fatality probability of a patient. First, we need to reach the patient's data. We can reach it by using for loop and look for everyone's name in patients_info database. If the patient isn't there, we cannot calculate due to absence. If we find the patient, now we can reach the patient's data. We need disease incidence, and diagnosis accuracy information. By using equation, we can calculate the result.

Recommendation() function

If command is probability, we need to call the probability() function, but firstly, let’s define it. We call the probability function to recommend to patient whether he/she should have the treatment. First, we need to call the probability function to calculate the actual fatality probability of a patient. We can reach it by using for loop and look for everyone’s name in patients_info database. If the patient isn’t there, we cannot calculate due to absence. If we find the patient, we can calculate probability of having disease. Also, we can reach the treatment risk from patient_info database. Now, we should compare them, and say if they should have the treatment or not.

List() function

We should give the output which is taken as input. Firstly, we need to reach the data.

Write_output_file() function

We should write the output to “doctors_aid_outputs.txt”. The output changes if the command changes, so we should look what the command is, and give the output what command corresponds. We should add every corresponded result to each function’s end as a result, and call the function in write_output_file function. When these functions returns as the results of functions, we should write these results to “doctors_aid_outputs.txt” by using .write() method.

Finishing the code

Now, we need to call the read_input_file() function, and write_output_file() function. Before these, we need to open the files to read, and write. After opening them, we need to read the input. We should call the read_ input_file() function. After reading the input file, we need to write the output which input file line corresponds, so we can add end of the read_input_file.

Programmer’s catalogue:

The time I spent for:

Analyzing: 30 minutes.

Designing: 2 hours.

Implementing: 4 hours

Testing: 5 hours

Reporting: 3 hours

```
def read_input_file():
    while True:
        each_line = file.readline()
        if each_line == "" or each_line == "\n":  # Reads until an empty line
            break
        blank_location = each_line.find(" ")
        global command
        command = each_line[0: blank_location]  # First word until first
whitespace is command
        rest_sentence = each_line[blank_location + 1: -1]
        global list_of_items  # we will use this list everywhere, so I made
it global
        list_of_items = rest_sentence.split(",")
        write_output_file()
```

It reads every line, if the line is empty, it stops. It reads first word until first blank, so every programmer can use it to read first word. Then, it reads the rest sentence after the blanc location, and it splits from commas. It is valid for each line, so it makes the code easier to read, and easy to process.

```
patient_name = list_of_items[0]
diagnosis_accuracy = list_of_items[1]
disease_name = list_of_items[2]
disease_incidence = list_of_items[3]
treatment_name = list_of_items[4]
treatment_risk = list_of_items[5]
```

It makes easier to read, and use them later.

Description of read_input_file() function: The code reads until the end of input file, assignes first read as command to use it later, converts the rest sentence to a list to use it later, and calls the write_output_file.

```
def read_input_file():
    while True:
        each_line = file.readline()
        if each_line == "" or each_line == "\n":  # Reads until an empty line
            break
        blank_location = each_line.find(" ")
        global command
        command = each_line[0: blank_location]  # First word until first
whitespace is command
        rest_sentence = each_line[blank_location + 1: -1]
        global list_of_items  # we will use this list everywhere, so I made
it global
        list_of_items = rest_sentence.split(",")
        write_output_file()
```

Description of write_output_file() function: It uses command, and calls the corresponded function, and writes to output file the return result.

```
def write_output_file():
    if command == "create":  # Command is first word which tells us what to
do.
        create_result = create()
        output_file.write(create_result)

    elif command == "remove":
        delete_result = delete()
        output_file.write(delete_result)

    elif command == "list":
        list_result = list_of_data()  # list is built-in name, so I changed
it.
        output_file.write(list_result)

    elif command == "probability":
        probability_result = probability()
        output_file.write(probability_result)

    elif command == "recommendation":
        recommendation_result = recommendation()
        output_file.write(recommendation_result)
```

Description of create() function: It looks to the database, if there isn't same person, it adds to database.

```
def create():
    patient_name = list_of_items[0]
    diagnosis_accuracy = list_of_items[1]
    disease_name = list_of_items[2]
    disease_incidence = list_of_items[3]
    treatment_name = list_of_items[4]
    treatment_risk = list_of_items[5]
    person = [patient_name, diagnosis_accuracy, disease_name,
disease_incidence, treatment_name, treatment_risk]
    # I assigned them to variables in order to make the code more readable.
    if (len(patients_info)) > 1:
        for x in range(len(patients_info)):
            name = patients_info[x][0]
            if name == patient_name:
                return f"Patient {patient_name} cannot be recorded due to
duplication.\n"
        patients_info.append(person)
        return f"Patient {patient_name} is recorded.\n"
    else:
        patients_info.append(person)
        return f"Patient {patient_name} is recorded.\n"
```

Description of delete() function: It looks to the database, if there is a same person in database, it deletes it.

```
def delete():
    name_to_delete = list_of_items[0]
    for x in range(len(patients_info)):
        a = patients_info[x][0]
        if a == name_to_delete:
            del patients_info[x]
            return f"Patient {name_to_delete} is removed.\n"
    return f"Patient {name_to_delete} cannot be removed due to absence.\n"
```

Description of probability() function: Calculates the probability of patient's fatality probability.

```
def probability():
    name_to_calculate = list_of_items[0]
    for x in range(len(patients_info)):
        a = patients_info[x][0]
        if a == name_to_calculate:
            disease_name = patients_info[x][2]
            disease_incidence = patients_info[x][3]
            diagnosis_accuracy = patients_info[x][1]
            divide_location = disease_incidence.find("/")
            numerator = float(disease_incidence[0: divide_location])
            denominator = float(disease_incidence[divide_location + 1:])
            cancer_probability = 100 * numerator / ((denominator - numerator)
* (1 - float(diagnosis_accuracy)) + numerator)
            cancer_probability = round(cancer_probability, 2)
            try:  # I called this function in recommendation function, so
```

```
 I need to look why I called this function.
               if probability_for_recommendation:
                   return cancer_probability
               else:
                   return f"Patient {name_to_calculate} has a probability of
{cancer_probability}% of having {disease_name}.\n"
           except:  # If it gives an error it means that
probability_for_recommendation is undefined, so this function is not called
for recommendation function.
               return f"Patient {name_to_calculate} has a probability of
{cancer_probability}% of having {disease_name}.\n"
    return f"Probability for {name_to_calculate} cannot be calculated due to
absence.\n"
```

Description of recommendation() function: It calls the probability() function, and uses the result to suggest if the patient should have the treatment or not.

```
def recommendation():
    global probability_for_recommendation
    probability_for_recommendation = True  # If I call the probability
function for recommendation function, it only returns to cancer probability.
    disease_probability = probability()
    probability_for_recommendation = False
    name_to_recommend = list_of_items[0]
    for q in range(len(patients_info)):
        if name_to_recommend == patients_info[q][0]:
            treatment_risk = float(patients_info[q][5]) * 100
            name_to_recommend = patients_info[q][0]
            if float(treatment_risk) > float(disease_probability):
                return f"System suggests {name_to_recommend} NOT to have the
treatment.\n"
            elif float(treatment_risk) < float(disease_probability):
                return f"System suggests {name_to_recommend} to have the
treatment.\n"
    return f"Recommendation for {name_to_recommend} cannot be calculated due
to absence.\n"
```

Description of list_of_data() function: It lists every data entered the system.

```
def list_of_data():
    first_line = f'{"Patient": <8}{"Diagnosis": <16}{"Disease":
<16}{"Disease": <12}{"Treatment": <16}{"Treatment"}'
    second_line = f'{"Name": <8}{"Accuracy": <16}{"Name": <16}{"Incidence":
<12}{"Name": <16}{"Risk"}'
    third_line = "-" * 77
    all_person_list = ""
    for person_number in range(len(patients_info)):
        patient_name = patients_info[person_number][0]
        diagnosis_accuracy = float(patients_info[person_number][1]) * 100
        diagnosis_accuracy = "{:.2f}".format(diagnosis_accuracy)
        diagnosis_accuracy = str(diagnosis_accuracy) + "%"
        disease_name = patients_info[person_number][2]w
        disease_incidence = patients_info[person_number][3]
        treatment_name = patients_info[person_number][4]
        treatment_risk = float(patients_info[person_number][5]) * 100
        if treatment_risk == int(treatment_risk):
            treatment_risk = int(treatment_risk)
        treatment_risk = str(treatment_risk) + "%"
        each_person_list = f"{patient_name: <8}{diagnosis_accuracy:
<16}{disease_name: <16}{disease_incidence: <12}{treatment_name:
<16}{treatment_risk}\n"
        all_person_list += each_person_list
    return f"{first_line}\n{second_line}\n{third_line}\n{all_person_list}"
```

User Catalogue

If you want to add a new patient enter like this: “create” patient name,_diagnosis accuracy, disease name, disease incidence, treatment name, treatment risk. Be careful: You can’t add a patient who added before.

If you want to remove a patient enter like this: “remove” patient name. Be careful: You can’t remove the patient if the patient is not in the database.

If you want to calculate the probability of patient’s fatality probability enter like this: “probability” patient name. Be careful: If the patient isn’t in the database, we cannot calculate the probability.

If you want to system’s recommendation enter like this: “recommendation” patient name. Be careful: If the patient isn’t in the database, the system cannot suggest.

If you want to list all the data enter like this: “list”