THE UNIVERSITY OF THE WEST INDIES

Department of Computing

ASSIGNMENT

SEMESTER I, 2023/2024

Code and Name of Course: **COMP1127 – Introduction to Computing II**

Group Assignment	
AREA	**DESCRIPTION**
Objectives	To have students apply the knowledge garnered during weeks 1 to 6 To have students work within a group setting with the allocation of work but still ensuring that the knowledge is spread within the group
Title	Group Assignment
Deliverable	The solution for the questions which follow the given instructions.
Instructions	1. Review Lectures of weeks 1 to 6 of the COMP1127 course, and any other related Introduction to Computing II or Python material 2. Read the Assignment Sheet thoroughly 3. As a group, in week 1 complete and test the assignment code in IDLE or another Interactive Development Environment 4. As a group, in week 2 implement and test the assignment code via Hackerrank 5. Submit the final solution by using the COMP1127 Assignment - Group located within the **Assignment** Section of the VLE COMP1127 Course Environment.

Group Restrictions	This is a group assignment. Each group must have a maximum of two (2) members.
Late Assignments	Late assignments are accepted. These are however graded then 10% deducted for each day of late submission.
Expectation	It is expected that students will discuss means to a solution within their group. The actual work written is expected to reflect the group's decision. Where replication of work is identified between groups, each solution will be graded. The allocated grade to each group's piece of work for where this anomaly is identified will be the grade divided by the number of replications discovered.
Submission	I. For testing purposes, each group member will submit a copy of the same program on hackerrank using the link https://www.hackerrank.com/comp1127-assign-2023-24-01 The Hackerrank link is available one week prior to the submission date. II. For final submission, **each group member (using his/her own hackerrank id) will submit a copy of the same program on hackerrank using the links for problems 1 to 8 that scores each problem separately**: Patient Severity – Part 1 Patient Severity – Part 2 Patient Severity – Part 3 . . Patient Severity – Part 8 III. For VLE submission, **each group member will submit well documented, original code for "Patient Severity – Part 8" using the container on VLE**. Name your file by concatenating the last four (4) digits of each member's ID#, separated by underscore and prefix the concatenated digits with the characters "COMP1127_". For example: COMP1127_1234_5678.py

	The following code must be incorporated into your own. Insert your own ID number
	
	`"""`
	`Group Information:`
	
	`Member 1: IDNUMBER`
	`Member 2: IDNUMBER`
	
	`"""`

Upload Constraint	The assignment should be uploaded in the relevant space provided in VLE (See "Instructions" section above). A message indicating "File uploaded successfully" will acknowledge that the file has been sent successfully. Do **NOT** assume your project has been received if you do not get this acknowledgement.
Coding Constraint	The use of Python 3.8 (or higher) environment is expected. ***N.B.*** Throughout this assignment, no abstractions are to be violated. **Have fun!**
Scoring Rubric	Your electronic submission will be evaluated on the Hackerrank submissions: Patient Severity – Part 1 – 10 marks Patient Severity – Part 2 – 10 marks Patient Severity – Part 3 – 10 marks Patient Severity – Part 4 – 10 marks Patient Severity – Part 5 – 10 marks Patient Severity – Part 6 – 10 marks Patient Severity – Part 7 – 20 marks Patient Severity – Part 8 – 10 marks The actual grade of 100 marks will be displayed. The actual grade allocated is the percentage of the maximum marks (15 points)
Due Date and Time	**Sunday, December 3, 2023** **11:59 PM**

Department of Computing

COMP1127 – Introduction to Computing

Heart Disease Patient Severity Order System

Overview

World Health Organization (WHO) states that cardiovascular diseases (CVDs) are the leading cause of death globally. An estimated 17.9 million people died from CVDs in 2019, representing 32% of all global deaths. Of these deaths, 85% were due to heart attack and stroke. It is important to detect cardiovascular disease as early as possible so that management with counselling and medicines can begin.

Cardiovascular diseases are a group of disorders of the heart and blood vessels. Heart disease refers to several types of heart conditions. The most common type of heart disease is coronary artery disease (CAD), which affects the blood flow to the heart. Decreased blood flow can cause a heart attack.

High blood pressure, high blood cholesterol, and smoking are key risk factors for heart disease. Jamaica had 6,457 deaths from CVD in 2019. For age-standardized mortality from CVD, it ranks in the bottom 40% of countries when ordering from highest mortality to lowest mortality rates. Jamaica has relatively high rates of diabetes in females and Non-HDL cholesterol in females when compared to other countries globally.

The symptoms of various heart diseases have variances. For heart attack, the symptoms are chest pain or discomfort, upper back or neck pain, indigestion, heartburn, nausea or vomiting, extreme fatigue, upper body discomfort, dizziness, and shortness of breath. For arrhythmia, the symptom is fluttering feelings in the chest (palpitations).

The importance of assessing disease severity in people with heart disease is well recognised, but validated CVD severity measures derived from routinely collected health records are lacking, as are applications of such measures in primary care settings. Most current CVD risk stratification tools are for people without CVD, but very few are for prevalent CVD.

The Requirement

In this programming assignment, you will develop a system to order Heart Disease patients based on the severity of their condition. The severity assessment will be determined by various factors related to the patient's health.

Your system will consider the following factors for assessing the severity of Heart Disease:

- The patient's age (age)
- The patient's gender (sex)
- The patient's resting blood pressure systolic value (trestbps)
- The patient's cholesterol level (chol)
- The patient's fasting blood sugar (fbs)
- The maximum heart rate achieved by the patient (thalach)
- ST depression induced by exercise relative to rest (oldpeak)

Part 1 – Create a Patient ADT

You are required to create a Patient Abstract Data Type (ADT) with the following format:

- It will be a tagged tuple with the tag "HDPT" (Heart Disease Patient) placed at index 0 of the tuple.
- At index 1, there will be a dictionary holding all the factors listed above.
- Each factor will be represented by the following keys:
    - The patient's age: 'age'
    - The patient's gender: 'sex'
    - The patient's resting blood pressure systolic value: 'trestbps'
    - The patient's cholesterol level: 'chol'
    - The patient's fasting blood sugar: 'fbs'
    - The maximum heart rate achieved by the patient: 'thalach'
    - ST depression induced by exercise relative to rest: 'oldpeak'

The constructor has the following interface:

```
makePatient : <Heart Dieses Factors> -> Patient
```

A Patient has the following format:

```
("HDPT", { 'age':27,'sex': "M", 'trestbps':145, 'chol':233,
'fbs':0, 'thalach':150, 'oldpeak':2.3 })
```

A tuple with a "**HDPT**" tag in the first position and a list of the heart dieses factors in index position one.

Implement the Patient ADT by completing the following functions.

`makePatient(age, sex, trestbps, chol, fbs, thalach, oldpeak)`	Constructor	Takes Patient information and returns a patient as a tuple, where the first part of the tuple is a tag 'HDPT'.
`getPatientAge(pt)`	Selector	Takes a Patient as input and returns the age of the patient.
`getPatientSex(pt)`	Selector	Takes a Patient as input and returns the sex of the patient.
`getPatientTres(pt)`	Selector	Takes a Patient as input and returns the patient's resting blood pressure.
`getPatientChol(pt)`	Selector	Takes a Patient as input and returns the patient's cholesterol level
`getPatientFbs(pt)`	Selector	Takes a Patient as input and returns the patient's fasting blood sugar.
`getPatientTlach(pt)`	Selector	Takes a Patient as input and returns the maximum heart rate achieved by the patient.
`getPatientST(pt)`	Selector	Takes a Patient as input and returns the ST depression induced by exercise relative to rest for the patient.
`isPatient(pt)`	Predicate	Checks to see if a given object, is a valid Patient.
`isEmptyPt(pkt)`	Predicate	Checks to see if a given Patient is empty.

Part 2 – Create the Functions to Analyse a Patient to determine their status.

You are tasked with creating functions to analyse the health status of a patient based on various factors. The function should consider thresholds for making decisions related to high blood pressure (hypertension), high cholesterol levels, and physical inactivity.

Create the following functions:

Function Description: `isHypertensive(age, sex, trestbps)`

This function takes three parameters, **`age, sex, and trestbps,`** representing the age, gender, and resting blood pressure of a patient, respectively. It is designed to determine whether the patient is hypertensive based on predefined thresholds for age, gender, and resting blood pressure.

Input Parameters:

- age (integer): Represents the age of the patient.
- sex (string): Represents the gender of the patient ('male' or 'female').
- trestbps (integer): Represents the resting blood pressure of the patient.

Output:

- Returns a boolean value:
- True if the patient is determined to be hypertensive based on the specified thresholds.
- False if the patient does not meet the criteria for hypertension.

Thresholds: The function evaluates the input parameters against predefined thresholds for age, gender, and resting blood pressure. The specific thresholds are set based on established medical guidelines for hypertension. Use the following thresholds seen in the example below.

Criteria:
The function evaluates the combination of age, gender, and resting blood pressure of the patient to determine if the patient is hypertensive. The specific criteria are set based on established medical guidelines for physical activity.

- A toddler (age 1 to 3), male or female, with resting blood pressure greater than or equal to 98 would be considered hypertensive.
- A child or adult (age 4 or greater), male or female, with resting blood pressure greater than or equal to 140 would be considered hypertensive.

Example:

```
age = 45
sex = 'male'
trestbps = 140

result = isHypertensive(age, sex, trestbps)
print(result)
```

Function Description: `hasHighCholesterol(chol)`

This function takes one parameter, **`chol`**, representing the cholesterol level of a patient. It is designed to determine whether the patient has high cholesterol based on a predefined threshold for cholesterol levels.

Input Parameters:

- chol (integer): Represents the cholesterol level of the patient.

Output:

- Returns a boolean value:
- True if the patient is determined to have high cholesterol based on the cholesterol equalling or exceeding specified threshold.
- False if the patient's cholesterol level is below the threshold for high cholesterol.

Threshold: The function evaluates the input cholesterol level against a predefined threshold. The specific threshold is set based on established medical guidelines for high cholesterol. Use the following threshold seen in the example below.

Example:

```
chol = 220

result = hasHighCholesterol(chol)
print(result)
```

Function Description: `isPhysicallyInactive(thalach, sex, age)`

This function takes three parameters, **`thalach, sex, and age,`** representing the maximum heart rate achieved by a patient, the gender of the patient, and the age of the patient, respectively. It is designed to determine whether the patient is physically inactive based on predefined criteria.

Input Parameters:

- thalach (integer): Represents the maximum heart rate achieved by the patient.
- sex (string): Represents the gender of the patient ('male' or 'female').
- age (integer): Represents the age of the patient.

Output:

- Returns a boolean value:
- True if the patient is determined to be physically inactive based on the specified criteria.
- False if the patient is considered physically active.

Criteria:

The function evaluates the combination of maximum heart rate, gender, and age to determine if the patient is physically inactive. The specific criteria are set based on established medical guidelines for physical activity.

- A male older than or equal to 35 with a maximum heart rate of 220 would be considered physically inactive.
- A female older than or equal to 35 with a maximum heart rate of 226 would be considered physically inactive.
- A male younger than 35 with a maximum heart rate of 230 would be considered physically inactive.
- A female younger than 35 with a maximum heart rate of 236 would be considered physically inactive.

Example:

```
thalach = 150
sex = 'female'
age = 40

result = isPhysicallyInactive(thalach, sex, age)
print(result)
```

Part 3 – Patient score ADT

In this section we will develop our patient score ADT that accepts a list of patients and returns a list of patients with their respective scores.

The constructor has the following interface:

```
makePscore : [patients] -> ["PS",[]]
```

A Score has the following format:

```
["PS", [(Patient, score), (Patient, score)]]
```

A list with a "PS" tag in the first position and a list in the first index position that contains the Patient score tuples.

Calculate the Patient score based on the points assigned to each factor below:

1. The Patient is hypertensive – 4 Points
2. The Patient has hight cholesterol – 3 Points
3. The Patient is physically inactive – 2 Points
4. The Patient is a male – 1.5 Points
5. The Patient is female – 1 Point

```
calPScore : Patient -> Float
```

Implement the following functions to complete the Score ADT

`makePscore(ptLst)`	Constructor	Takes a List of Patients and returns a score list, where the first part of the list is a tag 'PS'
`addPatient(PSLst,pt)`	Mutator	Takes a Score List and a Patient as input and adds the patient to the score list. this function calculates the patient score before adding it to the list.
`getCritical(Score)`	Selector	Takes a Score as an input and returns a list of all critical Patients.
`getNonCrit(Score)`	Selector	Takes a Score as an input and returns a list of all non-critical patients.
`isScore(Score)`	Predicate	Checks to see if a given list, is a valid Score
`isEmptyScore(Score)`	Predicate	Checks to see if a given Score list is empty.

Part 4 – Create Patient Queue

In this section, you will create a Patient Queue ADT capable of managing three different types of queues: a critical queue, a non-critical queue, and a watch queue. The queues will be differentiated by the tags they have: C-Q for critical queue, N-Q for non-critical queue, and W-Q for the watch queue. All queues will store patient scores.

Given a sore list, we will now filter the patients based on the score. If a patient has an overall sore greater than 7.00 then this patient should be added to the critical queue. If the patient's score is less that 5.00 they should be added to the non-critical queue. If the patient's score is between 5.00 and 7.00 they should be placed in the watch queue.

The constructor for the Patient queue has the following interface:

```
makePtQueue: <Queue Type a an Int> -> PtQueue
```

A Patient queue has the following format:

```
("C-Q", [])
("N-Q", [])
("W-Q", [])
```

a tuple with a tag in the first position and a list of patient scores in the second position in the order of highest to the lowest overall score.

Implement the Patient Queue ADT by completing the following functions.

`makePtQueue(Qtype)`	Constructor	Returns an empty Patient Queue as a tuple, where the first part of the tuple is a tag and the second part of the tuple is an empty list. The tag is determined by the Qtype. 1 for C-Q, 2 for N-Q and 3 for W-Q.
`contentsQ(q)`	Selector	Takes a Patient Queue as input and returns the list of Patients in the Patient Queue.
`frontPtQ(q)`	Selector	Takes a Queue as an input and returns the element in the front of the list.
`addToPtQ(pt,q)`	Mutator	Takes a Patient and a Patient Queue as input and adds the given patient to the **appropriate** position in the queue.

`removeFromPtQ(q)`	Mutator	Takes a Patient Queue as input and removes the front element from the Queue.
`isPatientQ(q)`	Predicate	Checks to see if a given Queue is a valid Patient Queue.
`isEmptPtQ(q)`	Predicate	Checks to see if a given Patient Queue is empty.

Part 5 – Create Patient Priority Stack

The Stack will hold patient records with a high suspicion of heart disease. Records that indicate a high risk, based on a suspicion score greater than 7.00, will be pushed onto the stack.

The constructor for the Patient Stack has the following interface:

```
makePatientStack: void -> PatientStack
```

A patient stack has the following format:

```
("HDS", [])
```

A tuple with an "HDS" tag in the first position and a list of patient records in the second position.

Implement the Patient Stack ADT by completing the following functions:

`makePatientStack()`	Constructor	Returns an empty Patient Stack as a tuple, where the first part of the tuple is a tag 'HDS' and the second part of the tuple is an empty list.
`contentsStack(stk)`	Selector	Takes a Patient Stack as input and returns the list of patient records in the Stack.
`topPtStack(stk)`	Selector	Takes a Patient Stack as an input and returns the element on the top of the stack.
`pushPtStack(pkt,stk)`	Mutator	Takes a Patient Record and a Patient Stack as input and adds the record to the top of the stack.
`popPtStack(stk)`	Mutator	Takes a Patient Stack as input and removes the top element from the stack.
`isPtStack(stk)`	Predicate	Checks to see if a given Stack is indeed a valid Patient Stack.
`isEmptyPtStack(stk)`	Predicate	Checks to see if a given Patient Stack is empty.

Part 6 – Sort Patients Based on Heart Disease Risk

Write a function **`sortPatients`** that sorts patients into the priority queue or the Patient Priority Stack based on the risk score for heart disease. This function accepts a patient list and modifies the queue and stack accordingly.

This function has the following interface:

```
sortPatients: patientList, patientStack, patientQueue -> void
```

Part 7 – Main Driver Function

Write the main driver function **`analyzePatients()`** having the following interface:

```
analyzePatients: patient_lst -> void
```

Each patient in the stream of patients will be in the following format:

```
('age', 'sex', 'trestbps', 'chol', 'fbs', 'thalach', 'oldpeak')
```

A tuple with the patient information.

The **`analyzePatients`** function accepts a list of patients in the above format.

This function accepts a stream of patient info as a list of patients (not in ADT format) and creates a patient (as described in our patient ADT). It **`analyzes`** each patient using the functions we created in **Part 2** and registers a risk score. The function then adds each patient to a score ADT. With that, the function separates the patients into the patient queue and patient stack based on the requirements stated in **Parts 4** and **5** using the function created in **part 6**. Finally, the function returns the queue of all the patients ready to be treated in the order of highest to lowest overall risk score.

Part 8 – Deliver the Heart Disease Patient Severity Order System

Include all functions and program code written for Parts 1 to 7

Part 1 – Create a Patient ADT

Part 2 – Create the Functions to Analyse a Patient to determine their status

Part 3 – Patient score ADT

Part 4 – Create Patient Queue

Part 5 – Create Patient Priority Stack

Part 6 – Sort Patients Based on Heart Disease Risk

Part 7 – Main Driver Function