NATIONAL JUNIOR COLLEGE
Mathematics Department

General Certificate of Education Advanced Level
Higher 2

COMPUTING 9569/02

Paper 2 Lab-based **13 September 2022**

3 hours

Additional Materials: Electronic version of data file HEX_SCORE.TXT
Electronic version of data file COUNTRY.TXT

Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions**.**

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

Note that up to **4** marks out of 100 will be awarded for the use of common coding standards for programming style.

The number of marks is given in the brackets [ ] at the end of each question or part question.
The total number of marks for this paper is 100.

This document consists of **x** printed pages and x blank pages.

Instructions to candidates:

- Copy the folder from the thumb drive to the PC's desktop and rename the folder on the desktop to `<your name>`. For example, TanKengHan).
- All the resource files are found in the folder and you should work on the folder in the desktop.
- Your program code and output for each of Task 1 to 3 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as `Task1_<your name>.ipynb`.
- You should have a total of **three** `.ipynb` files to submit at the end of the paper.
- The answer files for Task 4 should be saved in a **single folder** `Task4_<your name>`
- At the end of the exam, copy the working folder on your desktop to the thumb drive.

Task 1

In mathematics, a matrix is a two dimensional object made of m rows and n columns, having $m \times n$ values or elements. For example, a 2 × 3 matrix `A`, with two rows and three columns, can be written as

$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$, where the value in the $\boldsymbol{i}$th row and $\boldsymbol{j}$th column is denoted as a_{ij}.

row 1 can be represented as a vector $\begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \end{pmatrix}$.

row 2 can be represented as a vector $\begin{pmatrix} a_{21} \\ a_{22} \\ a_{23} \end{pmatrix}$.

column 1 can be represented as a vector $\begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix}$.

column 2 can be represented as a vector $\begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix}$.

column 3 can be represented as a vector $\begin{pmatrix} a_{13} \\ a_{23} \end{pmatrix}$.

A UML class diagram that can be use to model a matrix abstract data type(ADT) is given as follows:

Matrix
-buffer: ARRAY[0:m-1,0:n-1] of INTEGER -row_count: INTEGER -column_count: INTEGER
+constructor(m:INTEGER, n:INTEGER) +get_dimension(): ARRAY[0:1] OF INTEGER +get_value(row:INTEGER,column:INTEGER):INTEGER +get_row(row:INTEGER): ARRAY[0:n-1] OF INTEGER +get_column(column): ARRAY[0:m-1] OF INTEGER +set_value(row:INTEGER, column: INTEGER, value: INTEGER) +set_row(row:INTEGER,value: ARRAY[0:n-1] OF INTEGER) +set_column(column:INTEGER, value: ARRAY[0:m-1] OF INTEGER) +__repr__(): RETURNS STRING

Attributes	**Description**
`buffer`	- Two dimension array to store the values of the matrix
`row_count`	- Number of rows in the matrix
`column_count`	- Number of columns in the matrix

Methods	**Description**
`constructor(m,n)`	- `m` and `n` are the number of rows and columns respectively in the matrix. - initialises the `buffer` attribute which is a two dimensional array with `None` values.
`get_value(row,column)`	- row and column number starts at 1. - returns the value in row number `row` and column number `column`.
`get_row(row)`	- row number starts at 1. - returns the row as an array in row number `row`.
`get_column(column)`	- column number starts at 1. - returns the column as an array in column number `column`.
`get_dimension()`	- return an array of 2 values containing the row_count and column_count. You may use a Python tuple as the return data type .

Methods	Description
`set_value(column,row,value)`	- row and column number starts at 1. update the value in row number `row` and column number `column` with `value`
`set_row(row ,vector)`	- row number starts at 1. - update the values in row number `row` with `vector` which is an array of n elements
`set_column(column,vector)`	- column number starts at 1. - update the values in column number `column` with `vector` which is an array of `m` elements
`__repr__()`	- returns the string representation of a Matrix instance. - the string returned should look like this when printed [a_{11} a_{12} a_{13} a_{21} a_{22} a_{23}] where a_{11}, a_{12}, a_{13} are the values in row 1 and a_{21}, a_{22}, a_{23} are the values in row 2.

For each sub-task, add a comment statement, at the beginning of the code using the hash symbol "#", to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 1.1
        Program code
        Output:
```

Task 1.1

Implement the Matrix class using Python code based on the UML and description above. You must ensure that the operations that update the Matrix object do not violate the integrity of the Matrix object and print an error message when there is a violation. [14]

Task 1.2

Write Python code to test your code in Task 1.1 by creating and printing the following matrices:

- $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$
- $(5 \quad 6 \quad 7)$
- $\begin{pmatrix}9\\8\\7\end{pmatrix}$
- ( ) , This is a matrix with no dimension (0 row and 0 column).
- $\begin{pmatrix}1 & 2\\3 & \end{pmatrix}$, This is a invalid matrix, row 2 is invalid [5]

The dot product is an algebraic operation that takes two equal-length sequences of numbers which can be represented by $n \times 1$ $matrices$ and returns a single number. The dot product of two such matrices is deined as:

$$\begin{pmatrix}a_1\\a_2\\a_3\end{pmatrix} \cdot \begin{pmatrix}b_1\\b_2\\b_3\end{pmatrix} = a_1b_1 + a_2b_2 + a_3b_3.$$

For example, $\begin{pmatrix}1\\3\\5\end{pmatrix} \cdot \begin{pmatrix}2\\4\\6\end{pmatrix} = (1 \times 2) + (3 \times 4) + (5 \times 6) = 44$

Note that the dimensions of the two vectors must be the same.

Task 1.3

Implement the dot product operation as a Python function as follows:

```
FUNCTION dot_product(A:ARRAY[0:n]OF INTEGER, B:ARRAY[0:n]OF
INTEGER)
RETURNS INTEGER
```
[3]

Task 1.4

Write Python code to test your code in Task 1.3 by performing the following dot product operation:

$$\begin{pmatrix}1\\3\\5\end{pmatrix} \cdot \begin{pmatrix}2\\4\\6\end{pmatrix}$$
[1]

The transpose of a matrix is an operator which flips a matrix over its diagonal; that is, it switches the row and column indices of the matrix ***A*** by producing another matrix, often denoted by $\mathbf{A}^{\mathbf{T}}$. In other words, the rows of the matrix becomes its columns and its columns becomes its rows. Thus, the transpose of a $m \times n$ matrix is a $n \times m$ matrix. For example, given a matrix

$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, the transpose of A,

$A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$

Task 1.5

Implement the transpose operation as a Python function as follows:

`FUNCTION transpose(A: Matrix ) RETURNS Matrix.`

where `Matrix` is the class that you implement in Task 1.1 [3]

The multiplication of two matrices ***A*** and ***B*** can be defined as follow:

If ***A*** is a $m \times n$ matrix and ***B*** is a $n \times p$ matrix,

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \dots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{pmatrix}$$

the matrix product **C** = **AB** can be written as

$$C = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \dots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \dots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \dots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \dots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Thus, the product **AB** is defined if and only if the number of columns in **A** equals the number of rows in **B**, in this case n. The resultant matrix `C`, will have `m` rows and `p` columns.

Task 1.6

The following **incomplete** algorithm can be used to implement the matrix multiplication operation using the Matrix class and the function/s that you implement in the previous tasks:

```
FUNCTION M_multiply(A:Matrix, B:Matrix) RETURNS Matrix
// A is a m × n matrix, B is a n × p matrix

    m ← ___
    n ← ___
    p ← ___

    new_matrix ← ___ // Create a ADT Matrix
    FOR row = 1 to ___ DO
        new_row ← List() //List is a dynamically sized array
        row_from_A ← ___ //
        FOR col = 1 to ___ DO
            col_from_B ← ___
            result ← ___
            APPEND result to new_row
        ENDFOR
        new_matrix.set_row(___)
    ENDFOR
    RETURN new_matrix
ENDFUNCTION
```

Write Python code to implement the `M_multiply()` function as described above. [7]

Task 2

Bowling is a sport in which a player throws a bowling ball down a synthetic lane and towards ten pins positioned at the end of the lane. The objective is to score points by knocking down as many pins as possible. The following paragraphs describe the basic rules and scoring for the game:

A bowling game

One game of bowling consists of ten frames. Each frame consists of two chances for the bowler to knock down ten pins.

Strikes and spares

Knocking down all ten pins on the first throw of any frame is called a strike, denoted by 'X' on the score sheet. If a bowler takes two throws to knock down all ten pins, it is called a spare.

Scoring and bonus scoring

Each pin that is knocked down is worth 1 point.

A strike is worth 10 points plus the number of pins hit on the next two throws.

A spare is worth 10 points plus the number of pins hit on the next throw.

The total score for a ten frame game ranges from 0 to 300 points.

The tenth frame

A bowler who strikes on the tenth frame will be given two extra throws.

A bowler who spares on the tenth frame will be given one extra throw. The number of pins hit on these extra throws will be added to the bowler's score.

Sample Scores

Player 1's score sheet:

Frame	1	2	3	4	5	6	7	8	9	10
Pins Hit	X	7 | 3	7 | 2	9 | 1	X	X	X	2 | 3	6 | 4	7 | 3 | 3
Frame Score	20	17	9	20	30	22	15	5	17	13
Cumulative Score	20	37	46	66	96	118	133	138	155	168

Player 2's score sheet:

Frame	1	2	3	4	5	6	7	8	9	10
Pins Hit	0 | 5	8 | 0	X	0 | 5	X	6 | 4	0 | 5	8 | 1	9 | 1	5 | 0
Frame Score	5	8	15	5	20	10	5	9	15	5
Cumulative Score	5	13	28	33	53	63	68	77	92	97

Player 3's score sheet:

Frame	1	2	3	4	5	6	7	8	9	10
Pins hit	X	X	X	X	9 | 1	X	0 | 0	2 | 2	8 | 2	X | X | X
Frame Score	30	30	29	20	20	10	0	4	20	30
Cumulative Score	30	60	89	109	129	139	139	143	163	193

An algorithm below describes the calculation of bowling scores.

The input, `throws`, is a string containing the sequence of pins knocked down in each throw, a strike is indicated as an 'X' For example, accoding to Player 1's score sheet shown above, the value for `throws` will be

`"X737291XXX2364733".`

```
FUNCTION throw2score(throw: STRING) RETURNS INTEGER
  IF throw = "X" THEN
    RETURN 10
  ELSE:
    RETURN INTEGER(throw) // Convert to integer
  ENDIF
ENDFUNCTION

FUNCTION calc_score(throws: STRING) RETURNS INTEGER
// Assume that the input is valid

  DECLARE total_score: INTEGER
  Initialise total_score
  FOR frame = 1 TO 10 DO
    DECLARE frame_score: INTEGER
    IF throws[1] = "X" THEN // index of throws starts at 1
      frame_score ← 10 + Sum of scores for throws[2] and throws[3]
      Remove the first element from throws
    ELSE
      frame_score ← Sum of scores for throws[1] and throws[2]
      IF frame_score = 10 THEN
        frame_score ← frame_score + score throws[3]
      END IF
      Remove the first two elements from throws
    ENDIF
    total_score ← total_score + frame_score
  ENDFOR
  RETURN total_score
ENDFUNCTION
```

For each sub-task, add a comment statement, at the beginning of the code using the hash symbol "#", to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 1.1
        Program code
        Output:
```

Task 2.1

Write Python code to implement an algorithm for calculating bowling scores. [5]

Task 2.2

Modify the code implemented in Task 2.1 to include code for writing the score sheet for a player into a file named `SCORE_<player_name>.TXT`, where `<player_name>` is the name of the player.The contents of the file should show the number of pins hit in each frame, the score for that frame and the cumulative score computation.The format of the output should look like this (Player 1 is the name of the player):

```
Player 1 Scoresheet
Frame          1    2    3    4    5    6    7    8    9   10
Pins hit    [ X ][7|3][7|2][9|1][ X ][ X ][ X ][2|3][6|4][7|3|3]
FrameScore    20   17    9   20   30   22   15    5   17   13
Cum. Score    20   37   46   66   96  118  133  138  155  168
```

The code for Task 2.2 should have a different function name from Task 2.1. [10]

Task 2.3

Write Python code to test the code implemented in Task 2.2 by using three test cases for Player 1, Player 2 and Player 3 score sheets as show above. You should generate 3 output files. [3]

Task 2.4

Write Python code to implement a **recursive** function, `validate(throws)`that will return `True`, if the value of `throws` is valid and `False` , if invalid. The value of `throws` is described above.

The value of `throws` is valid if it contains a sequence of values that allows a score to be computed for a ten frame bowling game.

For example in the following games played:

Frame	1	2	3	4	5	6	7	8	9	10
Pins hit	X	X	X	X	9 | 1	X	0 | 0	2 | 2	8 | 2	9 | 1 | X
Frame Score	30	30	29	20	20	10	0	4	19	20
Cumulative Score	30	60	89	109	129	139	139	143	162	182

- `validate("XXXX91X00228291X")` should return `True.`
- `validate("XXXX91X00228290")` should return `True.`
- `validate("XXXX91X00228291")` should return `False,` as the tenth frame is a spare, there should be 1 more value.
- `validate("XXXX91")` should return `False,` as there are not enough values to compute a score for ten frames.

Frame	1	2	3	4	5	6	7	8	9	10
Pins hit	X	X	9 | 0	X	X	X	9 | 0	X	X	X | 8 | 0
Frame Score	29	19	9	30	29	19	9	30	28	18
Cumulative Score	29	48	57	87	116	135	144	174	202	220

- `validate("XX90XXX90XXX80")` should return `True.`
- `validate("XX90XXX90XXX")` should return `False,` as the tenth frame is a strike, there should be two more values.
- `validate("XX9XXX90XXX80")` should return `False,` as the values for the third frame is incorrect.
- `validate("XX95XXX90XXX80")` should return `False,` as the values for the third frame is incorrect.
- `validate("Xz90XXX90XXX80")` should return `False,` as "z" is an invalid value [9]

Task 2.5

Write Python code to test the `validate` function in Task 2.4, using one valid, one boundary and one invalid test cases. [3]

Task 3

The result of an international bowling tournament is stored in a file named `HEX_SCORE.TXT`. Due to a glitch in the scoring system, the values of the score are all in hexadecimal numbers. The file contains the player ID, the country code for the country the player is representing and the scores for the six games played in the tournament.

Task 3.1

Write Python code to convert a string containing hexadecimal number to denary numbers.You are not allowed to use any built-in Python function. [5]

Task 3.2

Each player's total score for the six games he/she played will be computed and thus determine his ranking in the tournament.

Write Python code to

- read the data from the file `HEX_SCORE.TXT`.
- compute the total score for each player
- use a bubble sort algorithm to sort the player's total score in descending order and then print the ranking of the players as follows in a similar format as given below:

```
Rank     ID    Score
 1   Player175  1734
 2   Player173  1723
 3   Player168  1678
 4   Player166  1670
 5   Player189  1519
 :       :        :
```

[6]

Task 3.3

Each country can be represented by one to five players. The total score for a country is computed by taking the average score of all the players who represented the country. The score should be rounded to an integer value. Print the results of each country as follows:

```
Country Num.Players Score
  SIN        5       1533
  MAS        3       1438
  IND        1       1723
  KOR        1       1519
  TPE        2       1409
  MAC        1       1400
```

[5]

Task 4

A bowling competition management system needs to implement a relational database to store information for its players and results during the competition. A web application is also used to display the results and to provide data entry operations.

Task 4.1

Create a bowling competition database named `Task4.db` with three tables. Write SQL statements to create the three tables Player, Country and Game described by the following:

- The Player table contains information of players in the competition. It has a PlayerID, Name, CountryCode, Gender and DateOfBirth.
- The Country table contains the information of the country represented in the competition. It has a Country Code, CountryName, PlayerID of Team captain.
- The Game table contains the scores obtained by the players during the competition. It has GameID, PlayerID, DatePlayed, StartTime, and Score.

PlayerID is the primary key for the PlayerID table. It should auto-increment. A player plays a maximum of six games in the competition.

GameID is the primary key for the Game table. It should auto-increment.

CountryCode is the primary key for the Country table. The team captain for the country must be a player.

You should include appropriate constraints in your CREATE statements.

Save your SQL code as `Task4_1_<your name>.sql`

[4]

Task 4.2

Write Python code to import the data from `COUNTRY.TXT` into the Country table. [4]

Task 4.3

Write a SQL query that shows :

- The player name
- The mean score for each player and the number of games played in the competition. [4]

Save your SQL code as `Task4_2_<your name>.sql`

Task 4.4

Create a web application using Python code and the necessary files that allows players information to be entered into the Players table and stored in the database. [6]

END OF PAPER