NATIONAL JUNIOR COLLEGE
Mathematics Department

General Certificate of Education Advanced Level
Higher 2

COMPUTING 9597/01

Paper 1

20 August 2019

3 hours 15 minutes

Additional Materials:	Pre-printed A4 Paper Removable storage device Electronic version of `WEATHER-JUNE-2018.TXT` data file Electronic version of `WEATHER-JUNE-2019.TXT` data file Electronic version of `PHONENUMS.TXT` data file Electronic version of `EVIDENCE.DOCX` document

READ THESE INSTRUCTIONS FIRST

Type in the `EVIDENCE.DOCX` document the following:
- Candidate details
- Programming language used

They are 4 questions worth a total of 100 marks. 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.

All tasks and required evidence are numbered.

The number of marks is given in the brackets [ ] at the end of each task.

Copy and paste required evidence of program listing and screenshots into the `EVIDENCE.DOCX` document.

This document consists of **11** printed pages and **1** blank page.

1 The text files `WEATHER-JUNE-2018.TXT` and `WEATHER-JUNE-2019.TXT` contain weather information in Singapore for the month of June in both 2018 and 2019. Each line in either file contains tab delimited data corresponding to (from the leftmost to the rightmost):

Date
Daily Rainfall Total (mm)
Highest 30-min Rainfall (mm)
Highest 60-min Rainfall (mm)
Highest 120-min Rainfall (mm)
Mean Temperature (°C)
Maximum Temperature (°C)
Minimum Temperature (°C)
Mean Wind Speed (km/h)
Max Wind Speed (km/h)

The Date values are in the format: d(d) Mmm – e.g., 1 Jun, 10 Jun. All the remaining data values are stored as either integers (no decimal place) or floating-point values (one decimal place).

Task 1.1
Write the program code to read and store the data in both files. You should use appropriate data structures to store this data such that each individual value may be easily referenced. The data from each file must be stored in different instances.

Evidence 1
- The program code for **Task 1.1**. [2]

In order to make a very simplistic observation on the effect of global warming over the past year, you have been tasked to calculate the median over the daily mean temperatures for the month of June in both 2018 and 2019.

Task 1.2
Write the program code for a function that will sort each of the two sets of data retrieved in **Task 1.1**. You are to implement the Bubble Sort algorithm to perform this sorting. The Bubble Sort algorithm should be applied to sort the records based on **Mean Temperature**.

Evidence 2
- The program code for **Task 1.2**. [5]

Task 1.3
Write the program code that utilises the Bubble Sort function implemented in **Task 1.2** to calculate the **median** of the Mean Temperatures from June 2018 and June 2019. Your program code should then output the difference between the two temperatures.

Sample output:
```
Difference in Median Temperatures (June 2019 – June 2018): 1.5°C
```

Evidence 3
- The program code for **Task 1.3**. [3]

In order to better study daily weather conditions, you must now also create a simple text-based user interface that will allow the user to search and output all temperatures (from both June 2018 and June 2019) that are above a certain mean temperature.

The records that have a higher mean temperature than the specified temperature should be output such that the following values are displayed:

Date
Daily Rainfall Total (mm)
Mean Temperature (°C)
Mean Wind Speed (km/h)

The user interface should repeatedly prompt the user for a temperature (i.e., a float).

Sample input message:

```
Please input a mean temperature threshold (°C):
```

Appropriate exception handling should be performed on user input, with an appropriate warning given whenever incorrect input is specified.

Once the user has input a temperature, the program should then output all the records (from both the June 2018 and June 2019 files) that have mean temperatures that are higher than the one specified.

Sample output:

```
YEAR      DATE      RAIN      TEMP      WIND
2018      7 Jun     0         30.3      11.5
2018      11 Jun    0         30.2      13.7
2018      6 Jun     0         30.2      12.6
2019      10 Jun    0         30        12.2
2019      11 Jun    0         29.9      11.9
2018      12 Jun    0         29.9      9.4
2018      10 Jun    0         29.9      12.2
2018      4 Jun     0.4       29.7      7.9
2019      28 Jun    0         29.6      14.4
2018      17 Jun    0.4       29.6      11.9
```

Note that this output should be sorted in descending order of (mean) temperature, and in the case of ties, in descending order of date.

Task 1.4
Write the program code for a function that performs an augmented Binary Search. This function should use the sorted data from **Task 1.1** to search for the record that is just greater than a specified temperature (i.e., the record with the lowest temperature that is just greater than the specified temperature).

Evidence 4
- The program code for **Task 1.4**. [6]

Task 1.5
Write the program code that uses the augmented Binary Search function implemented in **Task 1.4** to implement the user interface specified above.

Evidence 5
- The program code for **Task 1.5**. [4]

2 While hunting for lost treasure, Lara happened upon an ancient scroll that contains the following text (with some missing text denoted by the underlined sections).

```
FUNCTION insertionSort(L: ARRAY OF INTEGER) RETURNS ARRAY OF INTEGER
    RETURN insertSortOuter(_______________)
ENDFUNCTION

FUNCTION insertSortOuter(L: ARRAY OF INTEGER, i: INTEGER) RETURNS
ARRAY OF INTEGER
    IF _______________ THEN
        RETURN L
    ELSE
        RETURN _____________________________________________
    ENDIF
ENDFUNCTION

FUNCTION insertSortInner(L: ARRAY OF INTEGER, j: INTEGER) RETURNS
ARRAY OF INTEGER
    IF _______________ THEN
        RETURN L
    ELSE
        IF _______________ THEN
            _____________________________________________
        ENDIF
        RETURN ______________________________
    ENDIF
ENDFUNCTION
```

Assist Lara by completing the above code.

Task 2.1
Write the program code for a recursive implementation of the Insertion Sort algorithm (based on the functions defined above).

Evidence 6
- The program code for **Task 2.1**. [6]

Lara is not completely sure that your implementation really works and asks you to provide proof that it works correctly.

Task 2.2
Design a set of test cases to test the algorithm you implemented in **Task 2.1**.

Evidence 7
- The table of test cases to test the program code implemented in **Task 2.1**. [2]

On another adventure, Lara happens upon a puzzle that requires the calculation of Fibonacci Numbers.

Once again, she enlists your help, and tasks you to assist her with the computation of these numbers. However, since she abhors loops, your implementation must be fully recursive (and thus, not include the use of any loops – i.e., no `while` or `for` statements).

Note that the Fibonacci Number sequence, F_0, F_1, F_2, …, is defined as follows:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ &\vdots \\ F_i &= F_{i-1} + F_{i-2} \end{aligned}$$

More specifically, you are required to write a function that:

- Takes one positive integer value as input, `n`
- If `n` is a Fibonacci Number, then determine (recursively), the smallest integer `k`, where `n` = F_k, the `k`-th Fibonacci Number, and return the value `k`
- Or else, if `n` is not a Fibonacci Number, return the value -1

Task 2.3
Write the program code for a recursive function, where no loops are used, to implement the algorithm described above.

Evidence 8
- The program code for **Task 2.3**. [8]

Task 2.4
Design a set of test cases to test the algorithm you implemented in **Task 2.3**.

Evidence 9
- The table of test cases to test the program code implemented in **Task 2.3**. [2]

Task 2.5
Write the program code (using your implementation of the algorithm in **Task 2.3**) to determine if the output for the integers: 1346270 and 24157817.

Evidence 10
- The program code to test the implementation in **Task 2.3** (using cases in **Task 2.4**). [2]

3 Desmond wishes to implement a Hybrid Data Structure (HDS) using object-oriented programming (OOP). This HDS is meant to store telephone numbers in the following format (as a string).

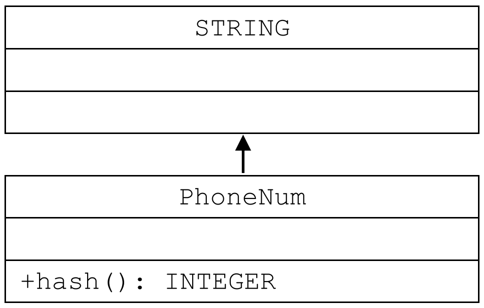
<Country Code String><Space Character><Number in Country String>

The Country Code String may include:

- Any digit – i.e., 0 to 9
- A dash – i.e., "-"

The Number in Country String may only include digits. It also includes any city codes, etc.

Consequently, you are to implement the following `PhoneNum` class.



Attribute/Method	**Description**
`PhoneNum.hash()`	This overwrites the default hash method and is calculated based on: ((ASCII value of char at index i) * i), for each element i in the word stored. Note that the value of i is based on 1-indexed indexing.

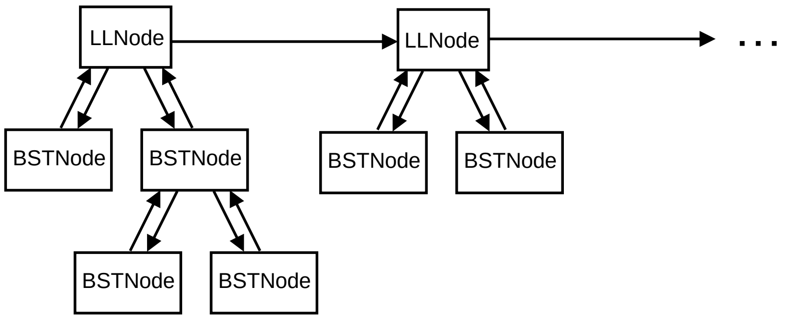
Task 3.1

Write the program code to implement the `PhoneNum` class.

Evidence 11

- The program code for **Task 3.1**. [3]

The HDS specifically corresponds to a Linked List (LL), where each node in this LL is also the root of a Binary Search Tree (BST).



Essentially, each LLNode stores a telephone number from a unique country, as defined by the <Country Code String> part of the string within each `PhoneNum` instance. Thus, among all the `PhoneNum` instances stored in LLNodes, there cannot be any instances that share the same Country Code String.

The BSTNodes are sorted based on the Hash Value of each `PhoneNum` instance.

It should also be noted that all `PhoneNum` instances are to be stored within an array in the HDS, and as such, all links stored in nodes are integers corresponding to indices within this array.

Consequently, you are to implement the following classes for the HDS.

Node
-data: PhoneNum -link1: INTEGER -link2: INTEGER -link3: INTEGER

+constructor()
+getData(): STRING
+setData(STRING)
+getLink1(): INTEGER
+setLink1(INTEGER)
+getLink2(): INTEGER
+setLink2(INTEGER)
+getLink3(): INTEGER
+setLink3(INTEGER)
+print()

HDS
-nodes: ARRAY OF NODES -free: INTEGER -first: INTEGER

+constructor(INTEGER)
+insert(PhoneNum)
+contains(PhoneNum)
+print()

Since each LLNode is also the root node of its corresponding BST, the `Node` class defined above is used to represent both LLNodes and BSTNodes.

The `link2` and `link3` attributes always correspond to the left and right nodes of the BST, whereas `link1` corresponds to either: (a) the next link in the LL when the node in question is a LLNode; or (b) the previous link in the BST when the node in question is a BSTNode.

The following are the descriptions of the various attributes and methods within the above-mentioned Node and HDS classes.

Attribute/Method	**Description**
`Node.constructor()`	Initialisation of a `Node` instance assigns the value `-1` to `link1`, `link2`, and `link3` attributes, and a `NULL` value to the `data` attribute.
`Node.data`	This attribute is used to store an instance of `PhoneNum`.

`Node.link1`
`Node.link2`
`Node.link3`
	Each of these attributes references an index (i.e., integer) from `HDS.nodes`.
	To indicate an empty reference, the value `-1` is used.
	When the `Node` instance in question corresponds to a LLNode, `link1` corresponds to the index of the next LLNode.
	When the `Node` instance in question corresponds to a BSTNode, `link1` corresponds to the index of the previous BSTNode (of LLNode).
	The `link2` and `link3` attributes always correspond to the indices of the left and right nodes in the BST.

`Node.print()`	This method should output the integer values stored in `link1`, `link2`, and `link3`, as well as the value in `data` the following format: `DATA: <STRING>; HASH: <INTEGER>; LINK1: <INTEGER>; LINK2: <INTEGER>; LINK3: <INTEGER>`

`HDS.nodes`
`HDS.free`
`HDS.first`
	The attribute `HDS.nodes` corresponds to an array of 25 `Node` instances. This array is to be pre-initialised with 25 unused `Node` instances. This value should be specified on creation of the new instance (i.e., as an argument).
	The `HDS.free` attribute references the index of the head (i.e., first node) of a singly-linked linked list of empty `Node` instances. The `Node.link1` values are used to reference each subsequent node index in this linked list. When initialising the `HDS.nodes` attribute, you must ensure that this linked list of free nodes is initialised as well.
	The `HDS.first` attribute references index of the head (i.e., first node) of the actual HDS.

`HDS.insert(PhoneNum)`	The `HDS.insert` method takes a `PhoneNum` instance as input and inserts it into the HDS. Note that each `Node` instance inserted into the actual HDS must correspond to a `Node` instance from the LL of free nodes, which is referenced by the head of the free node LL at the index stored in the attribute `HDS.free`. Insertion requires you to first remove the node from the LL of free nodes before inserting it into the actual HDS. When inserting into the HDS, you must first check the Country Code String part of `PhoneNum` and insert it into the LL part of the HDS only if that country code does not already exist in the LL part of HDS. If the Country Code String part of the `PhoneNum` instance in question does exist (in the LL part of the HDS), then it is instead to be inserted into the BST whose root shares the same Country Code String. When inserting into a BST within the HDS, do recall that within each BST, nodes are sorted based on Hash Value.
`HDS.contains(PhoneNum)`	This method will return True if the given `PhoneNum` instance exists within the `HDS`, or else returns False.
`HDS.print()`	Prints the contents of `HDS.nodes` in index order. Note that to print each `Node` instance within `HDS.nodes`, `Node.print()` should be called.

Task 3.2

Write the program code to implement the `Node` class.

Evidence 12

- The program code for **Task 3.2**. [4]

Task 3.3

Write the program code to implement the `HDS` class, **excluding** the `insert` and `contains` methods.

Evidence 13

- The program code for **Task 3.3**. [5]

Task 3.4

Write the program code to implement the `insert` method for the `HDS` class.

Evidence 14

- The program code for **Task 3.4**. [13]

Task 3.5

Write the program code to implement the `contains` method for the `HDS` class.

Evidence 15

- The program code for **Task 3.5**. [7]

Task 3.6

Write the program code to initialise a `HDS` and store the contents of `PHONENUMS.TXT`. as `PhoneNum` instances within the initialised `HDS`. Then print the contents of the `HDS`.

Evidence 16

- The program code for **Task 3.6**. [3]

Evidence 17

- The screenshot output corresponding to **Task 3.6**. [1]

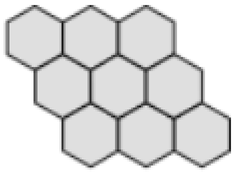
Task 3.7

Write the program code for the `HDS.orderedPrint` method, which uses in-order traversal on each BST within the HDS.

Evidence 18

- The program code for **Task 3.7**. [4]

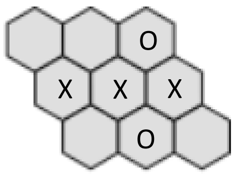
4 The Hex game involves an n × n hexagonal board. An example of a 3 × 3 Hex board is thus as follows.



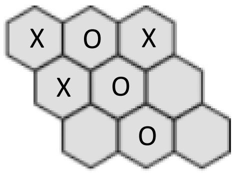
Hex is a two-player game, where one player must build a bridge that extends from left to right, and the other player must build a bridge that extends from top to bottom.

Each player takes turns to play, and may place a piece in any empty cell.

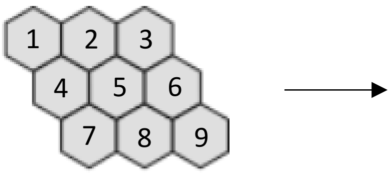
The following is an example board where the X player (going from left to right) has won the game.



The following is an example board where the O player (going from top to bottom) has won the game.



The representation for a Hex Board may be based on a standard 2-dimensional Array. Essentially, the Hex Board may be referenced as a 2-dimensional Array as follows.



1	2	3
4	5	6
7	8	9

You are tasked to design an object oriented programming class to store the Hex Board. This class, `HexBoard`, should be implemented as follows.

`HexBoard`

`-board: ARRAY OF ARRAY OF STRING`
`-turn: INTEGER`

`+constructor(INTEGER)`
`+playX(INTEGER, INTEGER)`
`+playO(INTEGER, INTEGER)`
`+checkWinX(): BOOLEAN`
`+checkWinO(): BOOLEAN`
`+printBoard()`

Attribute/Method	Description
`HexBoard.constructor (INTEGER)`	Initialises the `board` attribute as a 2D Array of Strings. The size of each array (both outer and inner arrays) are based on the specified integer value. The `turn` attribute is initialised as 0.
`HexBoard.playX (INTEGER, INTEGER)`	This method allows the X player to make a move by specifying the coordinates where he or she wished to place an X piece.
`HexBoard.playO (INTEGER, INTEGER)`	This method allows the O player to make a move by specifying the coordinates where he or she wished to place an O piece.
`HexBoard.checkWinX(): BOOLEAN`	This method checks the board and returns True if the X player has won the game, or else returns False.
`HexBoard.checkWinO(): BOOLEAN`	This method checks the board and returns True if the O player has won the game, or else returns False.
`HexBoard.printBoard()`	This method prints the contents of the board using the 2D Array representation.

Task 4.1

Write the program code to implement the `HexBoard` class, excluding the `checkWinX` and `checkWinO` methods. Your solution **must work for any board size**.

Evidence 19

- The program code for **Task 4.1**. [4]

Task 4.2

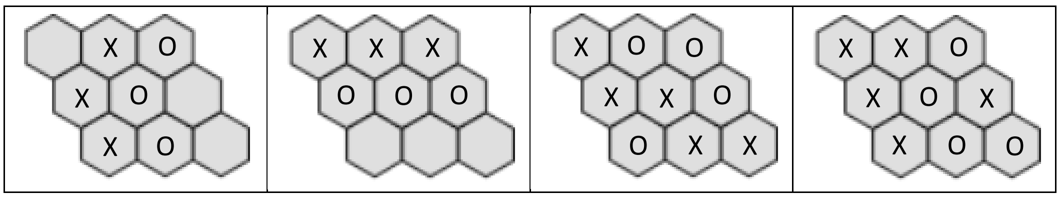
Write the program code to implement the `checkWinX` and `checkWinO` methods.

Evidence 20

- The program code for **Task 4.2**. [12]

Task 4.3

Write the program code to test the following 4 test cases for both X and O.



Note that X wins by forming a bridge from left to right, while O wins by forming a bridge from top to bottom.

Evidence 21

- The screenshots of the inputs and outputs to test each of the above cases. [4]

BLANK PAGE