INNOVA JUNIOR COLLEGE
JC 2 PRELIM EXAMINATION
in preparation for General Certificate of Education Advanced Level
Higher 2

CANDIDATE NAME	
CENTRE NUMBER	
INDEX NUMBER	

H2 COMPUTING **9597/01**

Paper 1 **12 September 2018**

3 hour 15 minutes

Additional Materials:	Pre-printed A4 Paper
	Removable storage device
	Electronic version of `DRINKS.TXT` data file
	Electronic version of `KEYS.TXT` data file
	Electronic version of `KEYS2.TXT` data file
	Electronic version of `PSEUDOCODE_TASK_4_1.TXT` file
	Electronic version of `SCORES.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

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 brackets [ ] at the end of each task.

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

At the end of the examination, print out your `EVIDENCE.DOCX` and fasten your printed copy securely together.

This document consists of **10** printed pages.

[Turn over

1 Customers of coffee shops in Singapore use ‘codewords’ to order their drinks. `DRINKS.TXT` is a text file containing the codewords.

Brewed coffee drinks begin with the word ‘Kopi’, while brewed tea drinks begin with the word ‘Teh’. All other drinks are classified as ‘Other drinks’.

The task is to read the codewords from the file and display a list according one of the following search criterion:

1. Brewed coffee
2. Brewed tea
3. Other drinks

Task 1.1

Design and write program code to:

- read the entire contents of `DRINKS.TXT` to an appropriate data structure called `Drinklist`
- display a menu with the following options:

```
Menu

1. Brewed coffee
2. Brewed tea
3. Other drinks
```

- generate the list of drinks and the total number of items according the user’s selection.

Example run of the program:

DrinkList
Kopi
Kopi Siew Dai
Teh O
Milo Dinosaur
Clementi

The output generated from the selection of option 1 would be:

```
Brewed coffee

Kopi
Kopi Siew Dai

Total items: 2
```

Evidence 1

Your program code.
Screenshot of the output for the selection of **Menu Option 3**. [15]

2 In a school, students are identified by student numbers. These numbers are stored in a hash table which uses the hashing function

```
Address <- StudentNumber MOD X
```

The hash table is implemented as a one-dimensional array with elements indexed 0 to (`X-1`).

Task 2.1

Write program code to:

- Read student numbers from a text file and store them in a hash table. For the purpose of testing the program, `X` is to be set to the value of 12.
  Assume different student numbers will hash to different addresses (no collisions).
- Print out the contents of the hash table in the order in which the elements are stored in the array.

Use `KEYS.TXT` to test your program code.

Evidence 2

Your program code.
Screenshot of the program output. [7]

Task 2.2

Linear probing is a method to handle collisions. This means that a collision is resolved by searching sequentially from the hashed address for an empty location and storing the student number at this empty location. If the end of the table is reached, the search for an empty location is continued from the start of the table.

Use `KEYS2.TXT` to test your amended program code.

Evidence 3

Your amended program code which performs linear probing.
Screenshot of the program output. [4]

Task 2.3

Add code to your Task 2.2 program.
The program is to:

- Take as input a student number
- Search the hash table and output the address (index number) of the hash table where the student number was found.

Use `KEYS2.TXT` to test your program code.
Run the program three times. Use the following inputs: 15, 23, 88.

Evidence 4

Your program code.
Screenshot of the program output. [7]

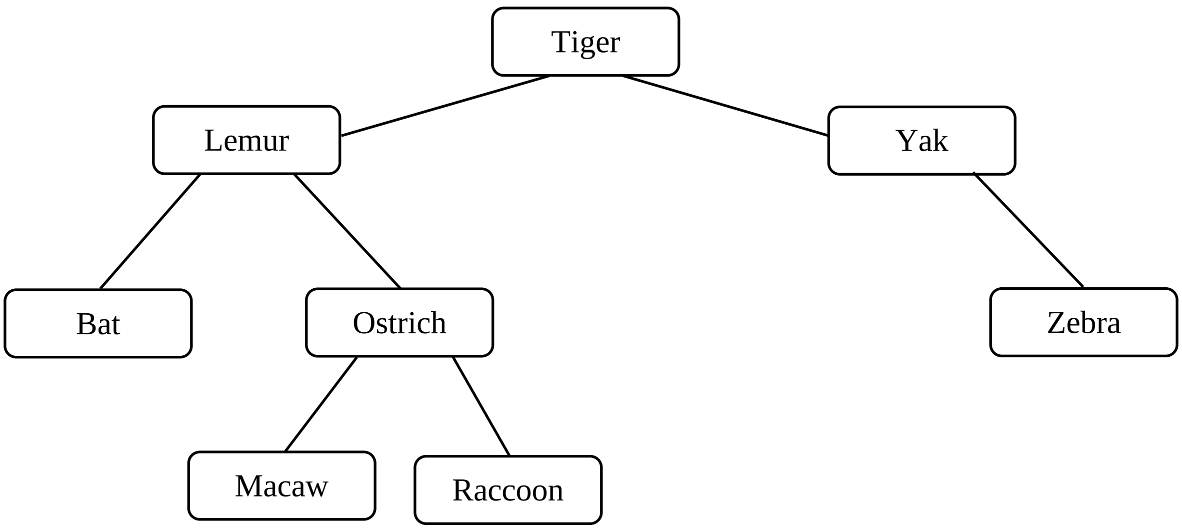
[Turn over

3 A binary tree structure is used to store the names of animals in the Singapore Zoo. Each animal's name is unique. The binary tree abstract data type (ADT) has commands to create a new tree, add data items and print the tree.

The sequence of commands:

```
CreateTree()
AddToTree('Tiger')
AddToTree('Lemur')
AddToTree('Bat')
AddToTree('Yak')
AddToTree('Ostrich')
AddToTree('Raccoon')
AddToTree('Macaw')
AddToTree('Zebra')
```

would create the following binary tree:



The program to implement the ADT will use the classes `Tree` and `Node` designed as follows:

Tree
thisTree : ARRAY of Node root : INTEGER

constructor()
add(newItem)
print()

Node
data : STRING leftPtr : INTEGER rightPtr : INTEGER

constructor()
setData(s : STRING)
setLeftPtr(x : INTEGER)
setRightPtr(y : INTEGER)
getData() : STRING
getLeftPtr() : INTEGER
getRightPtr() : INTEGER

The program code will do the following:

- Create a new tree, which has:
    - no nodes
    - the root set to –1
- Use the root as a pointer to the first node in the tree
- Add a new node to the tree in the appropriate position. The left and right pointers of this node should have the initial value of –1.
- Use the `print()` method to output, for each node, in array order:
    - the data item
    - the left pointer
    - the right pointer.

Task 3.1

Write program code to define the classes `Tree` and `Node`.

Evidence 5

Your program code. [26]

Task 3.2

The program is to be tested.
Write a sequence of program statements to:

- create a new tree
- add the data items as shown in the sequence of commands on the previous page
- print the array contents.

Execute your program to test it.

Evidence 6

Your program code.
Screenshot of test run. [3]

Task 3.3

A method `postOrderTraversal()` is to be added. This left-to-right post-order traversal outputs the data stored in the child nodes before outputting the data stored in the root node.
Write program code to:

- implement this method
- Test the program code with the data from Task 3.2.

Evidence 7

Your program code.
Screenshot of test run. [7]

[Turn over

4 Bowling is a sport in which a ‘bowler’ rolls 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.

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 roll of any frame is called a **strike**, denoted by ‘X’ on the score sheet. If a bowler takes two rolls 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 rolls.
A spare is worth 10 points plus the number of pins hit on the next roll.
The total score for a game ranges from 0 to 300 points.

The tenth frame
A bowler who strikes or spares the tenth frame will be given one extra roll. The number of pins hit on this roll will be added to the bowler’s score.

Sample Scores

Frame	1	2	3	4	5	6	7	8	9	10
Result	**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

Frame	1	2	3	4	5	6	7	8	9	10
Result	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

Frame	1	2	3	4	5	6	7	8	9	10
Result	**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

The following algorithms calculate the total score for a bowling game.

```
//Used interchangeably in the code for readability
Ten ← 'X'
Strike ← Ten

//Converting the 'X' or 'number' to INTEGER
FUNCTION Pins(Throw as STRING)
    IF Throw = Ten THEN
       RETURN 10
    ELSE
        RETURN INTEGER(Throw)
    ENDIF
END FUNCTION

//Recursive Procedure
FUNCTION Bowling_Score(Throws as STRING)

    //Helper function to keep track of current frame number
    FUNCTION Bowling_Score_Helper(Throws as STRING, Frame_Num as INTEGER)

        //Frame 10 with no bonus
        IF Frame_Num = 10 AND LENGTH(Throws) = 2 THEN
           RETURN SUM(Pins(Throws[0]), Pins(Throws[1]))
        ENDIF

        //Frame 10 with bonus
        IF Frame_Num = 10 AND LENGTH(Throws) = 3 THEN
           RETURN SUM(Pins(Throws[0]),Pins(Throws[1]),Pins(Throws[2]))
        ENDIF

        //A Strike
        IF Throws[0] = Strike THEN
           Frame_Score ← 10 + SUM(Pins(Throws[1]), Pins(Throws[2]))
           RETURN Frame_Score
                   + Bowling_Score_Helper(Throws[1:], Frame_Num + 1)
        ENDIF

        Frame_Score ← SUM(Pins(Throws[0]), Pins(Throws[1]))

        //A Spare
        IF Frame_Score = 10 THEN
            RETURN 10 + Pins(Throws[2])
                       + Bowling_Score_Helper(Throws[2:], Frame_Num + 1)
        ENDIF

        //Frame with no bonus
        RETURN Frame_Score + Bowling_Score_Helper(Throws[2:], Frame_Num + 1)
    END FUNCTION

    RETURN Bowling_Score_Helper(Throws, 1)
END FUNCTION
```

Note: The above pseudocode is available in the text file `PSEUDOCODE_TASK_4_1.TXT` but with ‘=’ used in place of ‘←’ shown above.

[Turn over

Task 4.1

Write a program to calculate a bowling score using the algorithms provided on the previous page.

Evidence 8

Your program code for Task 4.1. [5]

Evidence 9

Screenshot of the results of the following function calls:

1. `Bowling_Score('X2815X91X365452X0X')` [1]
2. `Bowling_Score('91739182X90X90X82X')` [1]

Task 4.2

Draw up a list of **three** suitable test cases. Complete a table with the following headings:

Bowling Score	Purpose of the test	Expected Output

Amend your program code to include the handling of the test cases listed in your table.

Evidence 10

The completed table. [3]

Evidence 11

Your amended program code that includes **internal commentary**. [3]

Evidence 12

Screenshots for each test data run. [3]

At the 2018 Singapore Championship, bowlers play a total of six games each in the qualifying round. The eight bowlers with the highest total score qualify for the Masters Competition. You may assume that there are no bowlers with the same total score.

`SCORES.TXT` is a text file containing the register number, country and the scores for six games of twenty bowlers in the qualifying round, with one bowler per line in the format:

```
<Register Number> <Country> <Score 1> <Score 2> … <Score 6>
```

For example:

```
 157 MAS XXXXX9091XXX82 90XXXXXXXXXX9 … 8172XXXX919191XX8
```

Task 4.3

Design and write program code to:

- Read the entire contents of `SCORES.TXT`.
- Calculate the score of each game for all twenty bowlers, using your code in **Evidence 8**. You may assume that all the scores in `SCORES.TXT` are valid scores.
- Calculate the total score for each bowler and store this in an appropriate data structure together with the respective `Register Number` and `Country`.
- Use **bubble sort** to sort the bowlers according to their total score.
- Output the Official Results of the competition.

An example of the output should look like this:

```
Official Results

Position       Register Number     Country   Total Score
    1               175              SIN       1734
    2               299              SIN       1724
    .                .                .          .
    .                .                .          .
    .                .                .          .
    20              245              MAS       1270
```

Evidence 13

The bubble sort code procedure. [4]

Evidence 14

Your program code for Task 4.3.
Screenshot showing the output. [11]

BLANK PAGE