NATIONAL JUNIOR COLLEGE
Science Department

General Certificate of Education Advanced Level
Higher 2

COMPUTING 9597/01

Paper 1

14 August 2018

3 hour 15 minutes

Additional Materials:	Pre-printed A4 Paper Removable storage device Electronic version of `DATA.TXT` data file Electronic version of `STRINGS.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 pages.

1 The text file `DATA.TXT` contains the records of 3 floating point values collected once a day from 9 August 1965 to 9 August 2018 (inclusive). Each set of 3 values (i.e., the set of 3 values for one day) is stored on one line of the file in the following format:

D<Day *i*>V1<Value 1>V2<Value 2>V3<Value 3>

Note that the value of *i* from <Day *i*> ranges from 0 to 19358 (i.e., *i* is the index for each day between 9 August 1965 and 9 August 2018).

Task 1.1

Write a program to read and store the contents of `DATA.TXT`. The 3 values for each day should be stored in a tuple (DATE, VALUE 1, VALUE 2, VALUE 3), where:

- DATE corresponds to a string in the format: DDMMYYYY.
- VALUE 1, VALUE 2 and VALUE 3 correspond to the 3 values for the given DATE.

Evidence 1

- The program code to perform **Task 1.1**. [4]

In order to analyse the data, you have been tasked to first sort the values in descending order based on the mean of VALUE 1, VALUE 2 and VALUE 3. You must use the Bubble Sort Algorithm to perform the required sorting.

Task 1.2

Write the function `bubble_sort_means`, whose input corresponds to the tuples generated in **Task 1.1**, and which outputs those tuples sorted in descending order based on the mean of VALUE 1, VALUE 2 and VALUE 3.

Evidence 2

- The program code for the function `bubble_sort_means`. [4]

You are next required to apply the `bubble_sort_means` function in order to compute the median value over the means of VALUE1, VALUE 2 and VALUE 3. You may not utilise any other form of sorting to compute this median value.

You should then output the median value using the following sample format.

```
The median value is: 12345.67
```

Task 1.3

Write the code that utilises the function `bubble_sort_means` to compute the median value and then output it using the specified formatting.

Evidence 3

- The program code to compute and output the median value. [3]

Evidence 4

- A screenshot of the output for **Task 1.3**. [1]

To check the values that arose on specific days, you are next required to sort the data in ascending order based on DATE. You must use the Insertion Sort Algorithm to perform the required sorting.

Task 1.4

Write the function `insertion_sort_dates`, whose input corresponds to the tuples generated in **Task 1.1**, and which outputs those tuples sorted in ascending order based on DATE.

Evidence 5

- The program code for the function `insertion_sort_dates`. [4]

Finally, you are required to:

- Repeatedly request a valid date from the user
- Perform a binary search using the output from the `insertion_sort_dates` function to find the specific tuple corresponding to the given date, and then output the 3 values for that date (i.e., VALUE 1, VALUE 2, VALUE 3)

The following is a sample of the input and output for the above functionality.

```
Please enter a valid YEAR: 2000
Please enter a valid MONTH: 6
Please enter a valid DAY: 22

The values on 2000-06-22 are: 00111.22, 22222.00, 03333.40

Do you wish to continue? (Y/N): N
```

Note that your code must include all relevant exception handling.

Task 1.5

Write the program code for the functionality specified above.

Evidence 6

- The program code for the functionality specified. [9]

2 A teacher is interested in exercise questions that perform division over integers specified using different bases. Assist the teacher by implementing a function that will allow him to check the answers of such exercise questions.

Task 2.1
Write the function `universal_base_division`, which has the following parameters (listed in order):

- `dividend: STRING`
- `dividend_base: INTEGER`
- `divisor: STRING`
- `divisor_base: INTEGER`
- `result_base: INTEGER`

The function will return two values (i.e., strings):

- the quotient of the division operation represented in base: `result_base`
- the remainder of the division operation represented in base: `result_base`

Note that `dividend` and `divisor` both represent integer values.

Note the following when implementing the above.

- Each base must be an integer between 2 and 30.
- If any base specified does not satisfy the above, output:
  `"You must specify a base between 2 and 30."`

Evidence 7

- The program code for the function `universal_base_division`. [11]

Task 2.2

Evaluate your code by testing the following:
`universal_base_division("mma01",23,"30501",6,2)`

Evidence 8

- A screenshot of the output from **Task 2.2**. [1]

Task 2.3
Using the table below, define a list of at least five test cases appropriate to thoroughly test the `universal_base_division` function.

Input	Purpose of the test	Expected output
⋮	⋮	⋮

Evidence 9

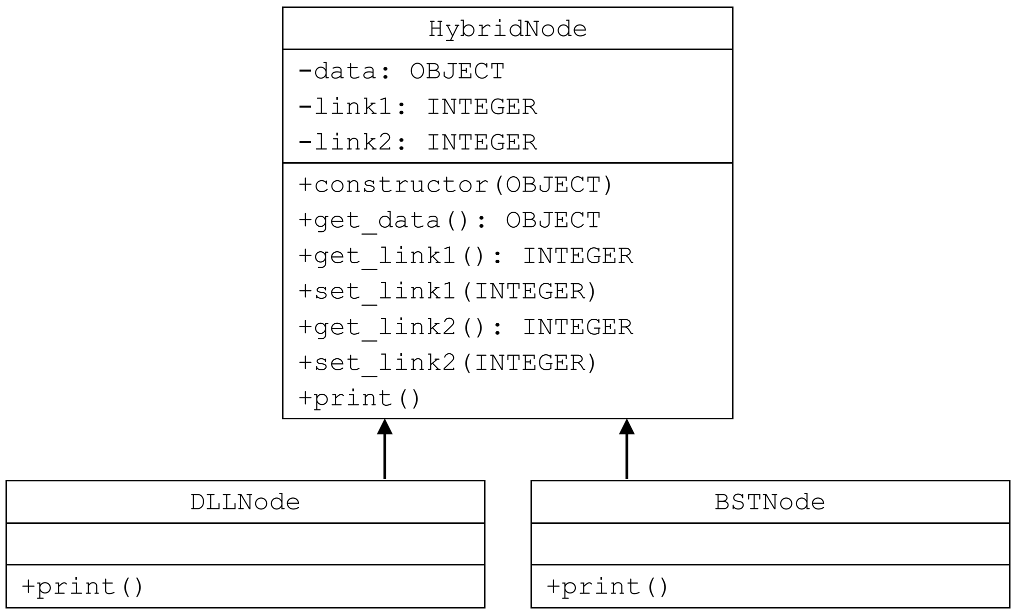
- The completed table of test cases. [3]

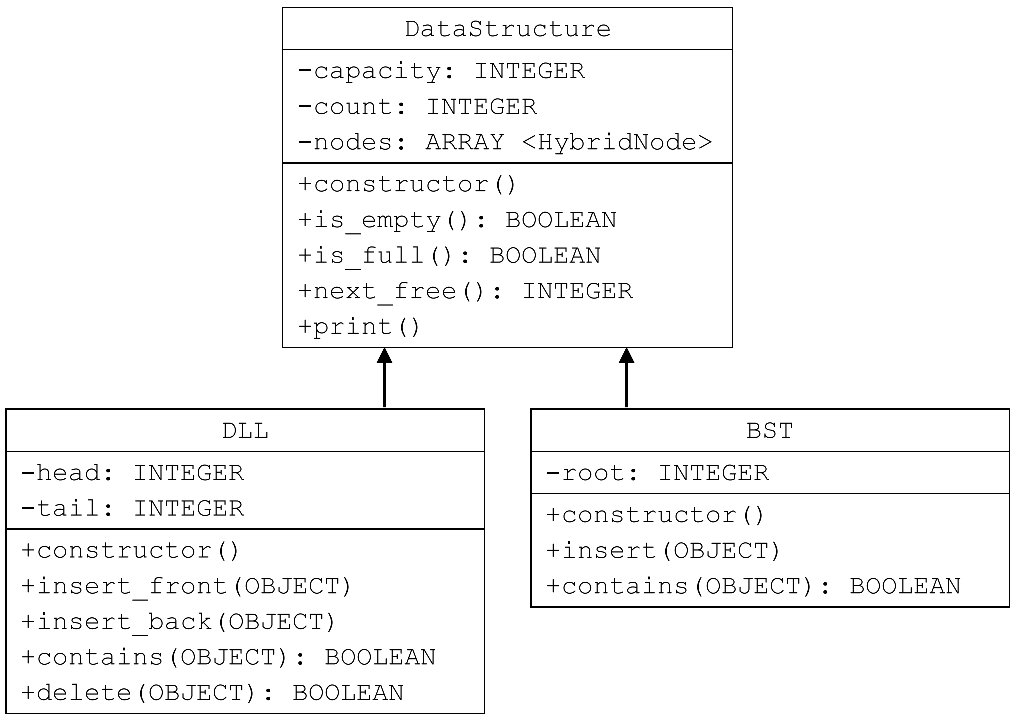
3 You have been tasked to implement a Doubly Linked List and a Binary Search Tree using object-oriented programming (OOP).

The implementations of these data structures are to adopt the use of an array to store their node objects. Consequently, all link information will correspond to integers representing the indices of the array that stores these nodes.

For example, root = 3 implies that actual root node is stored at index 3 of the array of nodes. Note that in order to indicate that no link has been defined, the value -1 should be utilised.

More specifically, you are to follow the following OOP design:





Attribute/Method	Description

`HybridNode.`
`constructor(OBJECT)`
	Initialisation of a `HybridNode` requires the input of the object to be stored. The given object is to be stored in the attribute `data`.

`HybridNode.print()`	This method should output the `data`, `link1` and `link2` values using the following format: `DATA: <value>; LINK1: <value>; LINK2: <value>`
`DLLNode.link1` `DLLNode.link2`	For each node in the doubly linked list (i.e., `DLLNode`), the attributes `link1` and `link2` are to be treated as the previous and next link references respectively.
`DLLNode.print()`	This polymorphed method should output the `data`, `link1` and `link2` values using the following format: `DATA: <value>; PREV: <value>; NEXT: <value>`
`BSTNode.link1` `BSTNode.link2`	For each node in the binary search tree (i.e., `BSTNode`), the attributes `link1` and `link2` are to be treated as the left child and right chid link references respectively.
`BSTNode.print()`	This polymorphed method should output the `data`, `link1` and `link2` values using the following format: `DATA: <value>; LEFT: <value>; RIGHT: <value>`

`DataStructure.capcity`
`DataStructure.count`
`DataStructure.nodes`
	The attributes of the `DataStructure` class are meant to facilitate the use of the array of nodes. The `capacity` attribute denotes the total number of nodes that can be stored (this is to be defined on construction). The `count` attribute denotes the number of nodes currently in the structure. And finally, the nodes attribute corresponds to the array of nodes. Do note that the array of nodes should be initialised to be empty, using null values, upon creation – i.e., any empty node should have a null value.

`DataStructure.`
`next_free(): INTEGER`
	This method search the attribute nodes for the first instance of an empty cell (i.e., an index housing a null value), and then returns that index. This is to facilitate the insertion of new nodes.

`DataStructure.`
`print()`
	This methods prints the contents of the data structure – i.e., all the nodes, in the order in which they appear in the array of nodes. Do note that empty cells of the array should also be appropriately reflected.

`DLL.head` `DLL.tail`	The DLL class maintains both a reference (i.e., index reference) to the start and end of the doubly linked list.

`DLL.contains(OBJECT):`
`BOOLEAN`
	This method will return True if the specified object can be found in the doubly linked list. Or else, False is returned.

`DLL.delete(OBJECT):`
`BOOLEAN`
	This method will search for the specified object. If the object is found, it is deleted and True is returned. If it is not found, False is returned.

`BST.contains(OBJECT):`
`BOOLEAN`
	This method will return True if the specified object can be found in the binary search tree. Or else, False is returned.

Task 3.1

Write the program code to implement the `HybridNode`, `DLLNode` and `BSTNode` classes.

Evidence 10

- The program code for the `HybridNode`, `DLLNode` and `BSTNode` classes. [5]

Task 3.2

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

Evidence 11

- The program code for the `DataStructure` class. [5]

Task 3.3

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

Then test it by executing the following blocks of instructions (in the sequence specified) on an instance of it:

Block 1	`insert_front(30)`
	`insert_front(20)`
	`insert_front(10)`
	`insert_back(40)`
	`insert_back(50)`
	`insert_back(60)`
	`print()`

Block 2	`delete(30)`
	`delete(10)`
	`delete(60)`
	`delete(40)`
	`print()`

Block 3	`insert_front(10)`
	`insert_back(60)`
	`print(contains(100))`
	`print(contains(10))`
	`print(contains(20))`
	`print(contains(50))`
	`print(contains(60))`
	`print()`

Block 4	`delete(100)`
	`delete(10)`
	`delete(20)`
	`delete(50)`
	`delete(60)`
	`delete(100)`
	`print()`

Note that each block of instructions is to be run in sequence. That is, run the instructions in block 1, then block 2, etc., without resetting the instance in between each block of instructions.

Evidence 12

- The program code for the `DLL` class. [12]

Evidence 13

- 4 screenshots; each capturing the output from each of the blocks of instructions. [4]

Task 3.4

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

Then test it by executing the following blocks of instructions (in the sequence specified) on an instance of it:

Block 1	`insert(50)`
	`insert(25)`
	`insert(35)`
	`insert(75)`
	`insert(85)`
	`insert(15)`
	`insert(65)`
	`print()`

Block 2	`print(contains(50))`
	`print(contains(25))`
	`print(contains(75))`
	`print(contains(19))`
	`print(contains(20))`
	`print(contains(21))`
	`print(contains(29))`
	`print(contains(30))`
	`print(contains(31))`
	`print(contains(69))`
	`print(contains(70))`
	`print(contains(71))`
	`print(contains(79))`
	`print(contains(80))`
	`print(contains(81))`

Note that each block of instructions is to be run in sequence. That is, run the instructions in block 1, then block 2, etc., without resetting the instance in between each block of instructions.

Evidence 14

- The program code for the `BST` class. [7]

Evidence 15

- 2 screenshots; each capturing the output from each of the blocks of instructions. [2]

4 You are given the text file `STRINGS.TXT`, which contains multiple entries of the strings "ABC", "ACB", "BAC", "BCA", "CAB", "CBA", with each such string stored on a separate line of the file.

You must write the code to analyse the frequencies of various substrings within the file.

The substrings in question correspond to all the possible strings containing at least 2 letters from “A”, “B” and “C”, such that there are no repeats of each of the letters.

You are to write the code for the function `generate_substrings`, which takes in a string of unique letters (e.g., “XYZ”), and then outputs an array containing all the possible strings comprised of at least 2 letters from the unique letters specified in the input string – i.e., if the input is “XYZ”, then the function should output [“XY”, “XZ”, “YX”, “YZ”, “ZX”, “ZY”, “XYZ”, “XZY”, “YXZ”, “YZX”, “ZXY”, “ZYX”].

Task 4.1

Write the program code for the function `generate_substrings`.

Evidence 16

- The program code for the function `generate_substrings`. [10]

Evidence 17

- A screenshot of the output for `generate_substrings("AGMSZ")`. [1]

You are next required to write 3 functions:

- `starts_with`
- `contains`
- `ends_with`

Each of these functions takes in 2 string inputs, `source` and `key`, and will search `source` for `key` at the position indicated by the function name. If `key` is found in the correct position within `source`, then the function will return True, or else, it will return False.

Your implementation of these 3 functions must be modular.

Task 4.2

Write the program code for the functions `starts_with`, `contains` and `ends_with`.

Evidence 18

- The program code for the functions `starts_with`, `contains` and `ends_with`. [10]

Use the functions implemented in **Task 2.1** and **Task 2.2** to determine the number of strings in the file `STRINGS.TXT` that start with each of the valid substrings of “ABC” (based on the definition of valid substrings stated at the beginning of this question). Then determine the number of strings in the file `STRINGS.TXT` that end with each of the valid substrings of “ABC”.

Task 4.3

Write the program code for the functionality specified above.

Evidence 19

- The program code for the functionality specified above. [2]

Evidence 20

- A screenshot of the output for the frequencies of strings in `STRINGS.TXT` starting with the various substrings identified. [1]

Evidence 21

- A screenshot of the output for the frequencies of strings in `STRINGS.TXT` ending with the various substrings identified. [1]