DHS 2018 P1Q1 [13 marks]

1. The file `LOG.TXT` contains the access log entries of an organisation's website from 0000 to 1800 hours on 1 August 2018.

The entries have the following format:

1. **host** (domain name or IP address) making the request.
2. **timestamp** in the format "DAY MON DD HH:MM:SS YYYY", where **DAY** is the day of the week, **MON** is the name of the month, **DD** is the day of the month, **HH:MM:SS** is the time of day using a 24-hour clock, and **YYYY** is the year. The timezone is -0400.
3. **request** in quotes.
4. **HTTP reply status code**.
5. **bytes in the reply.**

Task 1.1

Determine the top 5 hosts which accessed the website during this period, in descending frequency order.

Sample output:

```
Top 5 hosts:
1 139.230.35.135                187
2 ns2.sharp.co.jp                95
3 194.157.109.130                62
3 ix-dfw12-08.ix.netcom.com      62
4 piweba1y.prodigy.com           55
5 205.163.36.61                  30
```

Evidence 1

Program code. [9]

Task 1.2

Determine the host which returned the largest reply size and the largest reply size.

Sample output:

```
slip4086.sirius.com 12345
```

Evidence

Program code. [4]

HCI 2018 P1Q4 [20 marks]

2. The Romans had their own system of number representation which used a sequence of upper case letter characters to represent a number. We shall consider the denary number 1 to 20 only.

The following letters represent each of the values shown:

Roman Numeral	Represents
I	One
V	Five
X	Ten
L	Fifty

A number is always written with the smallest number of characters, with the letters in sequence starting with the character with the largest value.

- For example, 6 is written `VI` (not `IIIIII`)

The exceptions to this sequence are as follows:

- one less than 5 -- which is written as `IV`
- one less than ten -- which is written as `IX`
- ten less than fifty -- which is written as `XL`

Task 4.1

Write program code with the following specification:

- Input a denary integer number in the range 1 to 20
- Validate the input
- Calculate the Roman numeral representation (write this code as a function)
- Output the Roman number.

Evidence 21

Your program code. [6]

Task 4.2

Draw up a list of **three** suitable tests and provide screenshot evidence for your testing.

Evidence 22

Annotated screenshots for each test data run. [3]

Task 4.3

Write additional program code with appropriate data validation for the following specification:

- Input two Roman numeral numbers between 1 and 20
- Ouput the sum of the numbers as a Roman numeral number.

Evidence 23
Your program code. [8]

Task 4.4
Draw up a list of **three** suitable tests and provide screenshot evidence for your testing.

Evidence 24
Annotated screenshots for each test data run. [3]

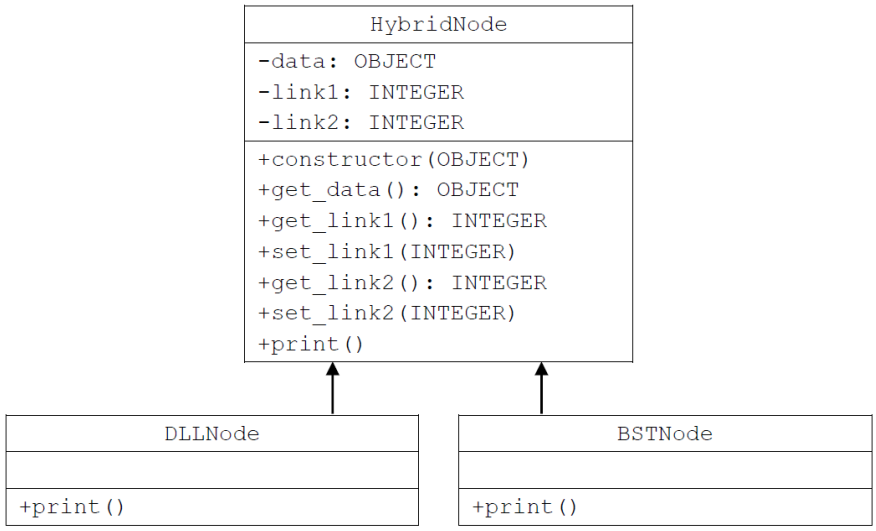
NJC 2018 P1Q3 [35 marks]

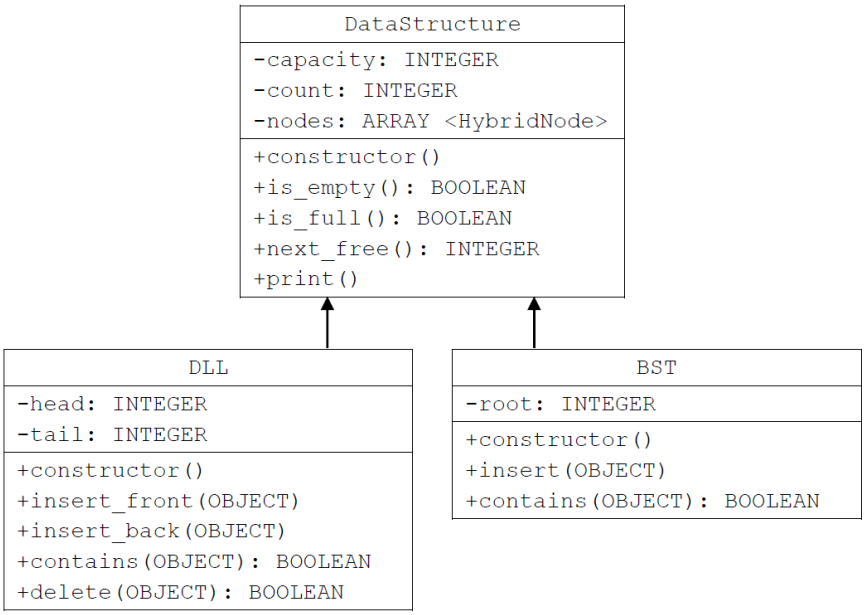
3 You have been tasked to implement a Doubly Linked List and a Binary Search Tree using object-oriented programming (OOP).

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]

PJC 2016 P1Q3 [30 marks] Modified to include socket programming

3. Tic-tac-toe is a game in which two players take turns marking `X` and `O` in the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

For example, player who marks `X` wins this game.

X	O	
X	X	X
O		O

The task is to write program code for a tic-tac-toe game for two human players.

Task 3.1

Decide on the design to be used for:

- the data structure to represent the 3x3 grid, using the identifier **`board`**
- the contents of the marks (`X`, `O` and blank) in the spaces
- how user input (`X` or `O`) in the spaces

Evidence 10: Your program design. Include program code for **`board`**. **[4]**

Task 3.2

Write a function **`displayBoard`** that will display the game board clearly to the players. You should use the **`board`** as a parameter in **`displayBoard`**.

Write another function **`displayInstructions`** to inform players on how to input X and O in the spaces on the game board.

Evidence 11: Your functions **`displayBoard`** and **`displayInstructions`**. **[4]**

Task 3.3

Write a function **`getPlayerMove`** to get players to make their move (by marking X or O) on the board. You should include validation on user input and check that the space is not already occupied. Use **`board`** as a parameter. You may include any other suitable parameters.

Evidence 12: Your function **`getPlayerMove`**. **[5]**

Task 3.4

Write a function **`checkWin`** that checks all the conditions for winning a game and returns True if a player has won the game, otherwise return False. Use **`board`** as a parameter. You may include any other suitable parameters.

Evidence 13: Your function **`checkWin`**. **[5]**

Modification to Task 3.5: Instead of a main function, make use of socket server and client scripting to allow 2 players to play the game

Task 3.5

Write a **`main`** function that makes use of the identifiers and functions from task 3.1 to task 3.4 and allows two human players to play a game of tic-tac-toe.

The **`main`** function should include the following:

- give instructions to players on how to input X or O
- ask whether player 1 or player 2 starts first move
- ensure players 1 and 2 take turns to make their move
- display the game board after every move made by a player
- check for winner
- display message on which player has won the game or whether the game ends in a draw

Evidence 14: Your **`main`** function. **[8]**

Evidence 15: Run your **`main`** function and produce screenshots of **three** games where player 1 wins one game, player 2 wins another game, and a drawn game. **[3]**

NJC 2020 Web App practice [10 marks]

1 A school directory stores information about schools and staff teaching in the schools. It allows users to query for staff information working in a particular school and department.

Information about a school includes:

- SchoolCode: a unique 4-digit number to identify the school.
- Name: name of the school.
- Address: address of the school

For staff working in schools, the following information is recorded:

- SchoolCode : schoolcode of the school the staff is working at
- Name: name of the staff
- Department: the department in the school the staff belongs to
- Contact: telephone number of the staff

It is assume that no staff working in the same schook will have the same name.

The information described above is currently stored in two files:

- `SCHOOL.TXT`
- `STAFF.TXT`

Task 1.1

Create an SQL file called `TASK1_1_<your name>_<NRIC number>.sql`.
Write the SQL code to create the the **two tables** in the database named `schools.db`. You are to create the necessary primary, foreign keys and constraints in your SQL code.

Execute the sql code in DB Browser to create the database and tables. [2]

Task 1.2

The files `SCHOOL.TXT` and `STAFF.TXT` contain information currently stored in the school directory system. Write Python code to migrate them to the database tables created in Task 1.1. The database tables should only contain the data in the two files.

Save your Python code as
`TASK1_2_<your name>_<NRIC number>.py`

[2]

Task 1.3

Write Python code in the file `main.py` to generate a web form that allows a user to search for staff/s using two fields:

- Name of School
- Department

a) The web form should be the default page when you access the web application

b) The search field for Name of School should allow for partial match. Example, the search for a school named "High" should return
   - NTU High School
   - Queens High School

   The search field for Department should be an exact match. [3]

Task 1.4

Write Python code in the file `main.py` to:

a) return a HTML page thate contains a table showing the results of the query **Task 1.3**

b) The information to be returned are:
   - Name of school
   - Name of staff
   - Department
   - Contact
   - Address of school

c) The format of the results should look like:

School	Name	Department	Contact	Address
Queens High Schhol	Melisa	Math	92468341	2 Queenstown
Queens High Schhol	Yolande	Math	97135073	2 Queenstown
NTU High Schhol	Uriel	Math	98946212	12 Nanyang Ave
NTU High Schhol	Caprice	Math	92780890	12 Nanyang Ave

[3]