1. A sports apparel store will be launching a mega sale exclusive only to its current members. Non – members who are interested in attending this event can register to be a member online. The online registration will eventually be captured as records and exported into a text file named `MEMBERS.txt`.

Every record in `MEMBERS.txt` has the following format:

```
<memberName>, <memberContactNo>, <memberEmail>
```

Example of a sample record:

Clement Ng, 99511232, clement@pjc.com.sg

It can be assumed every record is **unique** and all members' data have been validated and verified.

Task 1.1

Write program code for the `FUNCTION storeMembers` and procedure `printMemArray` using the following specifications:

```
FUNCTION storeMembers (filename: STRING): ARRAY
```

- `storeMembers` **reads** every single record taken from the text file that is specified by the single parameter `filename` as `INPUT`.
- The records read will be **stored** and **returned** as `OUTPUT` in an `ARRAY`.

```
PROCEDURE printMemArray (membersArray: ARRAY)
```

- `printMemArray` takes in a single parameter `membersArray` as `INPUT` and **displays** the required output given below:
    - o the **details of every single record** from `membersArray`,
    - o **total number of records** displayed.
- An example of the `OUTPUT` display with the headings:

```
    Name                 Contact No                 Email
    Ali                   99199191               ali@pjc.com
    ……                    ……………………               ……………
-------------------------------------------------------------
Total no. of records: ……
```

Evidence 1:
Your **program code** for `storeMembers` and `printMemArray` in **Task 1.1.** **[14]**

Task 1.2

Write a program code to first call `storeMembers` by using `MEMBERS.txt` as its input parameter and store the result into an `ARRAY`, and then followed by `printMemArray.`

Evidence 2:
Screenshot of running program code for **Task 1.2.** **[1]**

Task 1.3

Write program code for procedure `searchMemberByOption` that **searches** for a member record specified by the following `INPUT` parameters:

- `memberList`, an `ARRAY` that stores the all the members' records.
- `searchKey`, a `STRING` as the keyword to find the target member record.
- `searchChoice` that accepts either `INTEGER` value 1 (search member by contact number) or 2 (search member by email address) as the option on how to perform the search

The `OUTPUT` of `searchMemberByOption` should **display** the **outcome of the search** (i.e. found/not found), and **details of the member record when it is found**.

Evidence 3:
Your **program code** for `searchMemberByOption` in **Task 1.3.** **[7]**

Task 1.4

Call `searchMemberByOption` in your **main program** by using the following:

- **Assign** `ARRAY` `memberList` as the returned result of `storeMembers`.
- **Request** for user to enter:
  - o `STRING searchKey,`
  - o `INTEGER` `searchChoice`.
- **Implement** range check on `searchChoice` to ensure its validity and correctness.
- **Pass** `memberList`, `searchKey, searchChoice` into `searchMemberByOption`.

Evidence 4:
Screenshot of running program code for **Task 1.4,** to **search** for:

- Member's contact number: 99911812
- Member's email address: hannibal@live.net **[3]**

[25 marks]

[Note: Question 2 is a continuation from Question 1. Hence you may make use of any functions or procedures taken from Question 1 in answering this question]

2. The sports apparel store wishes to boost advertising and marketing. Therefore, they have decided give out a $30 online voucher each to 5 **different** lucky members selected at random.

Task 2.1

Write program code to **randomly** select **5 different** members taken from the members list and **display** their details as shown below:

```
5 Lucky Draw Winners ($30 online voucher):
==========================================
 Name                 Contact No            Email
  Ali                  99199191             ali@pjc.com
  ……                   …………………             ……………
```

Evidence 5
Program code for **Task 2.1.**
(you may make use of `randint(a,b)` in this task) **[9]**

Evidence 6
Screenshot of running program code for **Task 2.1.** **[1]**

[10 marks]

3. A data structure is required to store nodes. A linked list is maintained of all the nodes. A node contains a data value and a pointer, which is initially set to `NONE`. Subsequently, items in the list are linked using the pointer.

Each node is implemented as an instance of the class `ConnectionNode`. The class `ConnectionNode` has the following properties:

Class: `ConnectionNode`		
Attributes		
Identifier	Data Type	Description
`data`	`STRING`	The node data
`next`	`CLASS`	The node pointer

The structure for the linked list `LinkedList` is implemented as follows:

Identifier	Data Type	Description
`head`	`CLASS`	Initially set to `None`, this points to the first node in the list.

`LinkedList` also has the following methods:

Identifier		Description
`isListEmpty`	`FUNCTION RETURNS BOOLEAN`	Returns whether the list is empty.
`listLength`	`FUNCTION RETURNS INTEGER`	Returns the number of nodes in the list.
`insertEnd`	`PROCEDURE`	Inserts a node at the end of the list.
`insertHead`	`PROCEDURE`	Inserts a node before the first node of the list.
`insertAt`	`PROCEDURE`	Inserts a node at the specified position. The node at the head of the list has position `0`.
`deleteAt`	`PROCEDURE`	Deletes a node at the specified position.
`deleteEnd`	`PROCEDURE`	Deletes the last node.
`printList`	`FUNCTION PRINTS STRING`	Prints the data value of each node in the list.

Task 3.1

Write program code that implements `ConnectionNode` and `LinkedList`. Copy and append all the code in `LINKEDLIST.TXT` to your code, and run your program.

Evidence 7

Your program code.
Screenshot of the output. **[30]**

4.

Connect 4 is a game played by two players. In the figure shown, one player uses red tokens and the other uses yellow. Each player has 21 tokens. The game board is a vertical grid of 6 rows and 7 columns.

Columns get filled with tokens from the bottom. The players take turns to choose a column that is not full and drop a token into this column. The token will occupy the lowest empty position in the chosen column. The winner is the player who is the first to connect 4 of their own tokens in a horizontal, vertical or diagonal line. If all tokens have been used and neither player has connected 4 tokens, the game ends in a draw.

Your task is to write a program to play this game on a computer by following these specifications:

- Represent the game board using a **2D array**;
- Designate players using ‘O’ and ‘X’;
- Player ‘O’ **always** start first;
- Players take turn in placing their tokens;
- Display game board after every turn;
- Check for a winner after a token is placed;
- Winner is the player who is the first to connect 4 of their tokens horizontally or vertically.
- The game can also be won by connecting 4 tokens ***diagonally***, but you are **<u>NOT REQUIRED</u>** to write code for winning with diagonally connected tokens.

Use this top-level pseudocode with the given modules:

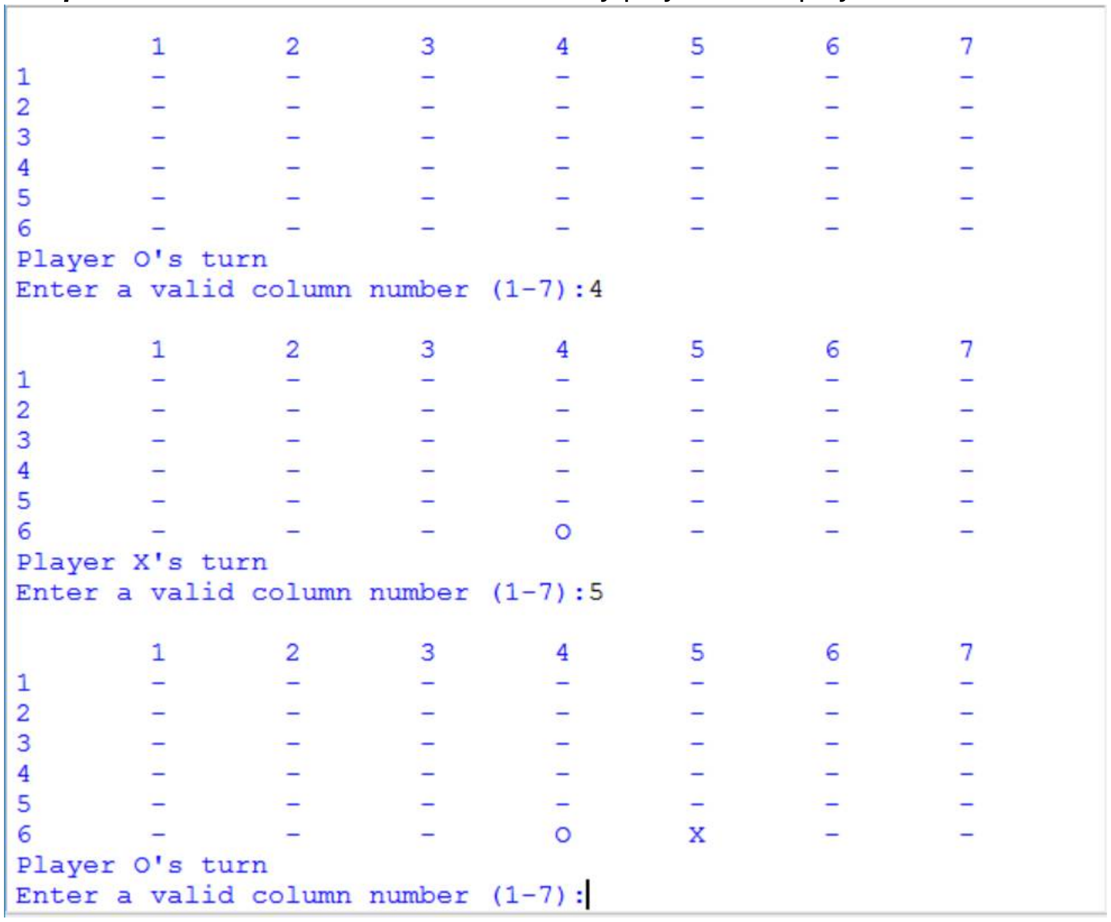
```
CALL InitialiseBoard
CALL SetUpGame
CALL OutputBoard
WHILE GameFinished = FALSE
    CALL ThisPlayerMakesMove
    CALL OutputBoard
    CALL CheckIfThisPlayerHasWon
    IF GameFinished = FALSE THEN
        CALL SwapThisPlayer
    ENDIF
ENDWHILE
```

The identifiers used in the pseudocode and explanations are given as follow:

Identifier	**Explanation**
`Board[1..6, 1..7]`	• 2D array to represent the board
`InitialiseBoard`	• Procedure to initialise the board to all blanks. • Use a suitable character to represent blank.
`SetUpGame`	• Procedure to set initial values for `GameFinished` and `ThisPlayer`
`GameFinished`	• FALSE if the game is not finished • TRUE if a player has won or board is full
`ThisPlayer`	• 'O' when it is Player O's turn • 'X' when it is Player X's turn
`OutputBoard`	• Procedure to output the current contents of the board
`ThisPlayerMakesMove`	• Procedure to get current player to input column number and place the token into the chosen board location. • Validation must be done on user input of column number.
`CheckIfThisPlayerHasWon`	• Procedure to check if the token just placed makes the current player a winner. • Checks should be made on whether the token just placed connected 4 tokens to form a horizontal or vertical line, and whether the game ends in a draw. • You **DO NOT** need to do diagonal check.
`SwapThisPlayer`	• Procedure to change player's turn

You **must use the above identifiers** and **other additional identifiers** of your own.

Row numbers and **column numbers** are displayed with the board's contents. Here is a ***sample screenshot*** of the first turns taken by player O and player X:



```
        1       2       3       4       5       6       7
1       -       -       -       -       -       -       -
2       -       -       -       -       -       -       -
3       -       -       -       -       -       -       -
4       -       -       -       -       -       -       -
5       -       -       -       -       -       -       -
6       -       -       -       -       -       -       -
Player O's turn
Enter a valid column number (1-7):4

        1       2       3       4       5       6       7
1       -       -       -       -       -       -       -
2       -       -       -       -       -       -       -
3       -       -       -       -       -       -       -
4       -       -       -       -       -       -       -
5       -       -       -       -       -       -       -
6       -       -       -       O       -       -       -
Player X's turn
Enter a valid column number (1-7):5

        1       2       3       4       5       6       7
1       -       -       -       -       -       -       -
2       -       -       -       -       -       -       -
3       -       -       -       -       -       -       -
4       -       -       -       -       -       -       -
5       -       -       -       -       -       -       -
6       -       -       -       O       X       -       -
Player O's turn
Enter a valid column number (1-7):
```

Task 4.1

Write program code for `InitialiseBoard`, `SetUpGame`, `OutputBoard`, and call these procedures. You may introduce **other additional identifiers** of your own, including parameter(s) and return value(s).

Evidence 8: Program code for `InitialiseBoard`, `SetUpGame`, `OutputBoard` and calling these procedures. Include a screenshot of running these procedures. **[8]**

Task 4.2

Write program code for `ThisPlayerMakesMove`. You may introduce **other additional identifiers** of your own, including parameter(s) and return value(s).

Evidence 9: Program code for `ThisPlayerMakesMove`. **[7]**

Task 4.3

Write program code for `CheckIfThisPlayerHasWon`. You may introduce **other additional identifiers** of your own, including parameter(s) and return value(s).

Evidence 10: Program code for `CheckIfThisPlayerHasWon`. **[12]**

Task 4.4

Write program code for `SwapThisPlayer`. You may introduce **other additional identifiers** of your own, including parameter(s) and return value(s).

Evidence 11: Program code for `SwapThisPlayer`. **[3]**

Task 4.5

Write program code for the top-level pseudocode that makes use of all the procedures from **Task 4.1 to 4.4**.

Evidence 12: Program code for the top-level pseudocode. **[4]**

Evidence 13: Run your program and produce screenshots for a game which ends in a draw and another game which player `X` wins. **[1]**

[35 marks]

- END OF PAPER -