1 A data structure is required to store 20 nodes. A linked list is used maintained of all the nodes. A node contains a data value and two pointers: a left pointer and a right pointer. The nodes in the linked list are initially linked using their `LeftChild` pointers.

Each node is implemented as an instance of the class `Node`.

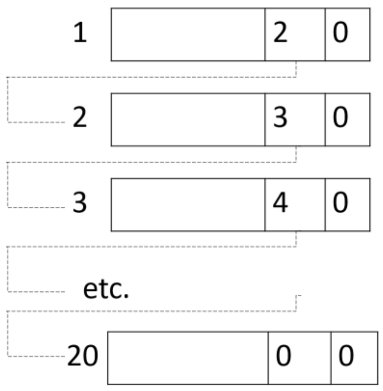
The class `Node` has the following properties:

Class: `Node`		
Attributes		
Identifier	Data Type	Description
`LeftPtr`	`INTEGER`	The left pointer for the node.
`Data`	`STRING`	The data value stored in the node.
`RightPtr`	`INTEGER`	The right pointer for the node.

The structure for the linked list is implemented as follows:

Class: `DataStructure`		
Attributes		
Identifier	Data Type	Description
`TreeData`	`ARRAY[1:20] OF Node`	An array used to store the 20 nodes.
`Root`	`INTEGER`	Index for the root position of the `TreeData` array. `Root` is initialized to 0.
`NextFree`	`INTEGER`	Index for the next available empty node in the array. `NextFree` is initialized to 1.
Methods		
`constructor`	`PROCEDURE`	Initialise `TreeData` array by setting pointers to indicate that all nodes are unused and linked. Initialise values for `Root` and `NextFree`.
`add`	`PROCEDURE`	Add a new data item to the linked list.
`display`	`PROCEDURE`	Display the current state of pointers and the array contents.
`Traversal`	`PROCEDURE`	Display the data item in order.

The diagram shows the empty data structure with the linked list to record the unused nodes.



Task 1.1

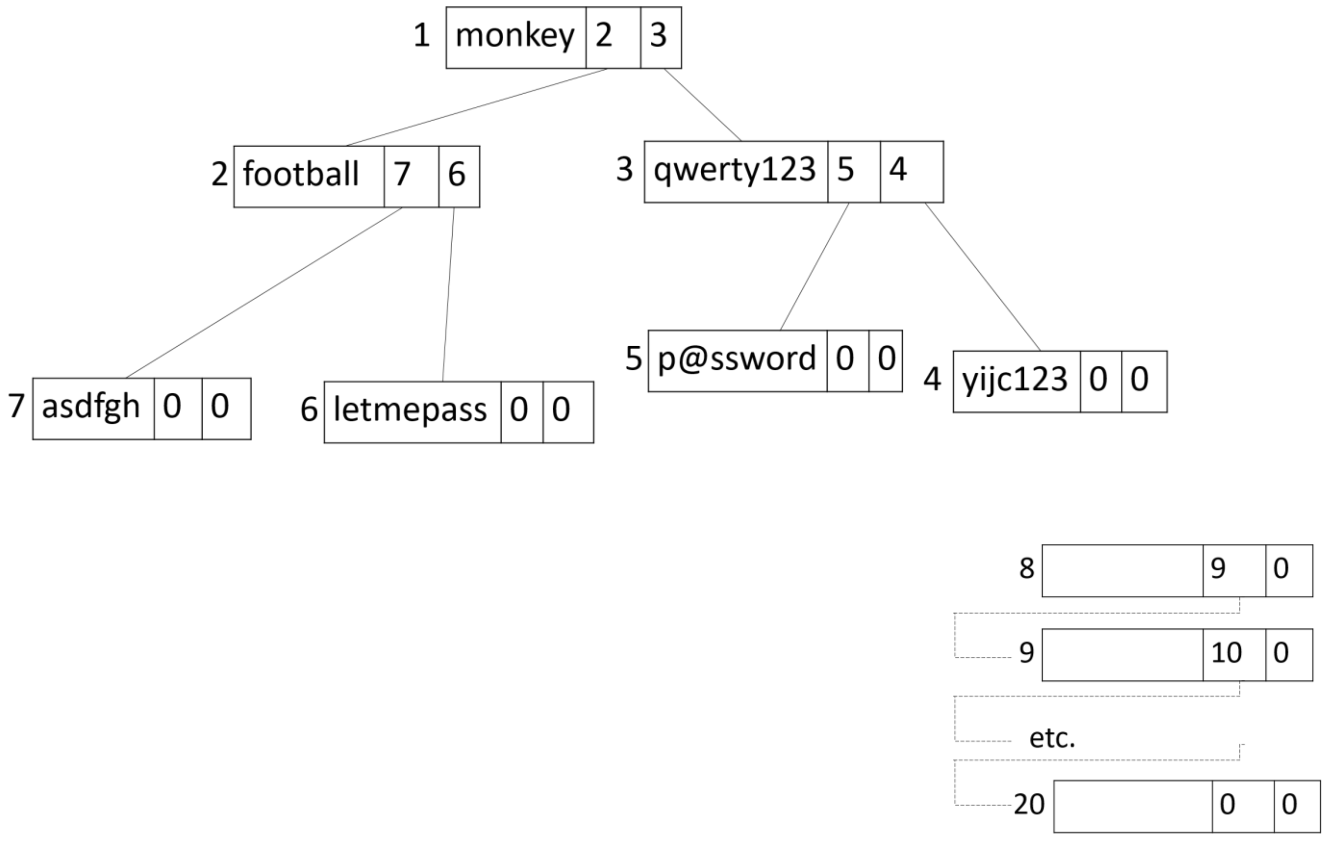
Write program code for the `Node` class and `DataStructure` class.

Include appropriate getters and setters for the `Node` class.

Do not attempt to write the code for the `add`, `display` and `traversal` methods within the `DataStructure` class yet. [8]

The data structure is used to store passwords of a user to allow for easy retrieval. The diagram below shows how the passwords are stored in the data structure when the following passwords are added:

`monkey, football, qwerty123, yijc123, p@ssword, letmepass, asdfgh`



Task 1.2

Write code to implement `add` and `display` methods for the `DataStructure` class. [14]

Task 1.3

Write a sequence of program statements to:

- instantiate the empty data structure
- add the given passwords into the data structure accordingly
- use the `display()` method to print the array contents

Execute your program to test it. [3]

Task 1.4

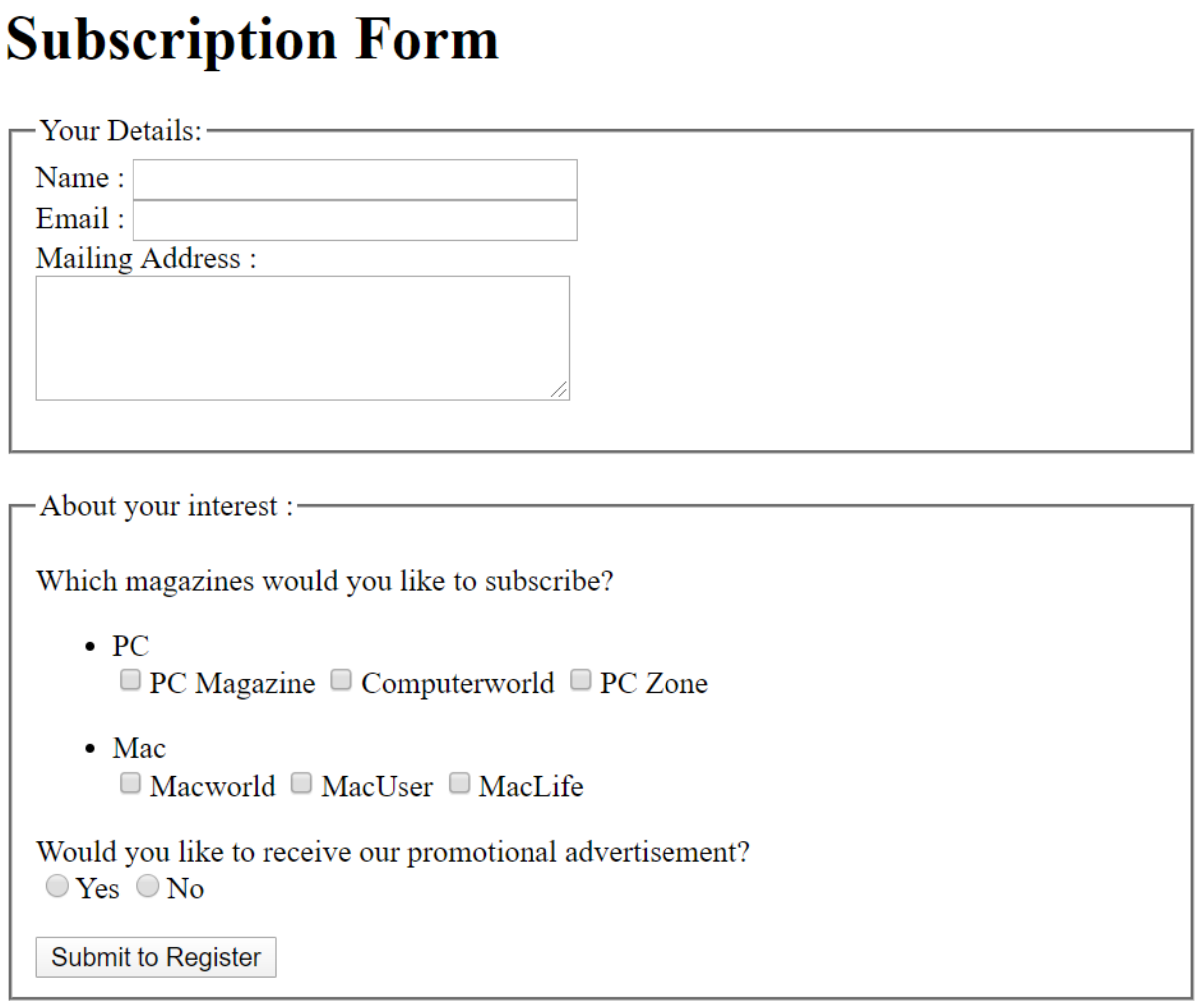
Write program code to implement the `Traversal` method. [5]

Save the Python codes for **Question 1** as `<your name>_Q1.ipynb`

2 A large company owns an online business that sells computer components and magazines. On its website, there is a subscription form for the user to subscribe their magazines.

Task 2.1

Create a HTML file called `index.html` to display the following Subscription Form for users to submit their request to the back-end server.



Subscription Form

Your Details:

Name :

Email :

Mailing Address :

About your interest :

Which magazines would you like to subscribe?

- PC
  ☐ PC Magazine ☐ Computerworld ☐ PC Zone
- Mac
  ☐ Macworld ☐ MacUser ☐ MacLife

Would you like to receive our promotional advertisement?
○ Yes ○ No

Save and submit the file `index.html` in the **Task 2.1** folder. [5]

The back-end server uses the following program code, `server.py`, to display the Subscription Form on the clients' browser when they visit the company's website.

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

app.run(debug=True, port=5000)
```

Task 2.2

For the back-end server to receive the inputs in the Subscription Form, an additional route '`/form`' should be included. Modify the program code to:

- prevent the user from accessing the '`/form`' route directly
- receive the inputs for `name` and `email` from the subscription form (ignore the other inputs)
- reject empty or null inputs
- reply by sending a `success.html` page back to the client's browser, displaying clearly the `name` and `email`, to acknowledge the submission

Save and submit the file `success.html` and the modified code for `server.py` file in the **Task 2.2** folder. [5]

The list of computer components and their prices can be found in the file `pricelist.csv`.

Task 2.3

Write program code to:

- create the database `mypricelist.db`
- create the table `components` with the following fields:
    - o `id`: an auto generated INTEGER attribute to be used as the primary key
    - o `description`: a TEXT attribute for the name of the component
    - o `price`: an INTEGER attribute for the price of the component
- import all the data from the given `pricelist.csv` file
- use a strategy to prevent *SQL Injection* when importing the data from the CSV file. [5]

Task 2.4

Write program code to:

- read from the **given** database `pricelist.db` and display information in the table `components` in a neatly tabulated format
- compute and display the total cost of the **first ten items** in the table `components` [5]

Save the Python codes as `<your name>_Q2.ipynb` for **Task 2.3** and **2.4**.

The back-end server uses the following program code, `server.py`, to display the list of computer components and their prices on the clients' browser.

```
from flask import Flask, render_template, request

app = Flask(__name__)

import sqlite3
db = sqlite3.connect('pricelist.db')

@app.route('/')
def index():
    db = sqlite3.connect('pricelist.db')
    c = db.cursor()
    c.execute('''SELECT id, description, price FROM components''')
    data = c.fetchall()
    return render_template("index.html", data=data)

app.run(debug=True, port=5000)
```

Task 2.5

Create a HTML file called `index.html` to display the Order Form which

- has the list of components and their prices as a *drop-down menu*
- allows the user to select **one** component to order and submit the required quantity to the back-end server

Save and submit the file `index.html` in the **Task 2.5** folder. [5]

Task 2.6

For the back-end server to receive the inputs in the Order Form, an additional route '`/order`' should be included. Modify the program code to:

- prevent the user from accessing the '`/order`' route directly
- accept the user's order, the name of the component and the required quantity, and insert a new record into the table `myorder` in the given database `pricelist.db`
- acknowledge the submission by sending a `success.html` page back to the client's browser, displaying clearly the item ordered and its quantity

Save and submit the file `success.html` and the modified code for `server.py` file in the **Task 2.6** folder. [5]

~ End of Paper 2 ~