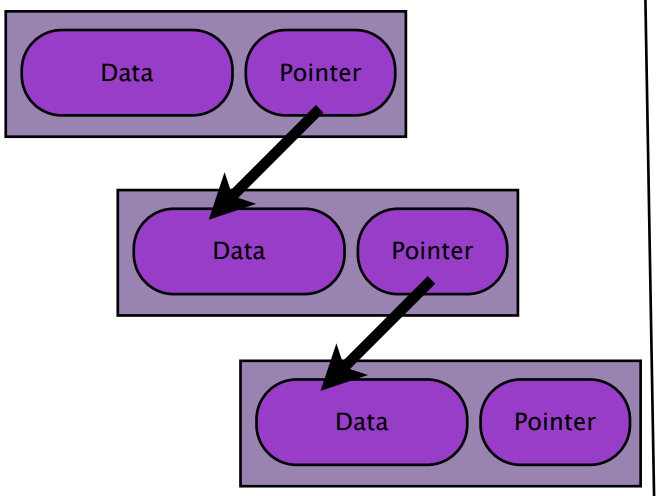
Linked Lists

A linked list is comprised of **nodes** that can be stored wherever there is space. These nodes are **linked** together via pointers, which point to where the next node is stored in memory.



Advantages

- **Inserting and deleting** elements is less time consuming
  - as only the pointers of nodes are changed
- **Dynamic allocation** of memory at run time from the heap

Disadvantages

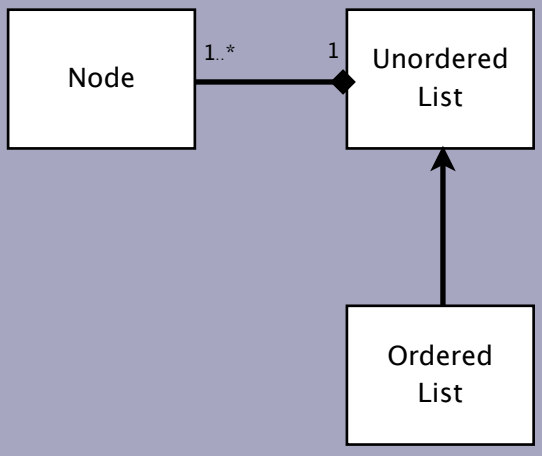
- **Memory leakage** if memory is not deallocated correctly when it is no longer required eventually there will be no memory left in the heap
- **Node or pointer corruption** if a node becomes corrupted it may no longer point to the next node in the list

Implementation

We will implement a linked list as an object. There are two types of linked list that we will consider:

- Unordered list
- Ordered list

These could shown as a class diagram:



The Heap is the name given to the memory locations available to the application programs for dynamic allocation.

Class definitions

From the classes identified on the class diagram we can produce the following definitions:

Node
Data NextPointer
GetData SetData GetNextPointer SetNextPointer

Unordered List
Head
isEmpty add getList length search remove

Ordered List
search add

The methods in Unordered list can be described as follows:

- **`isEmpty`** returns true or false depended on whether the list is empty or not
- **`add`** a new node is added to the head of the list
- **`getList`** returns the list as a string in the order the items were added (head last)
- **`length`** returns the number of nodes in the list
- **`search`** returns true or false depended on whether the item is present in the list
- **`remove`** removes the item from the list altering the required pointers

The Task

Implement the node type and the unordered and ordered list types in Python as objects with the given attributes and methods.

How will you implement the search and add methods differently in the Unordered List so that they have the appropriate functionality?

There is some help on the back of this document. Try not to use it...

```
FUNCTION search(item: Integer)
    current: Node
    found: Boolean
    found ← FALSE
    WHILE current != None AND found = FALSE DO
        IF (CALL getData from current) = item THEN
            found ← TRUE
        ELSE
            current ← (CALL getNext from current)
        END IF
    END WHILE
    RETURN found
END FUNCTION

FUNCTION add(item: Integer)
    temp: Node
    temp ← Node(item)
    (CALL setNext in temp) ← head of List
    head of List ← temp
END FUNCTION

FUNCTION remove(item: Integer)
    current,previous: Node
    found: Boolean
    current ← head of List
    found ← FALSE
    WHILE found = FALSE DO
        IF (CALL getData from current) = item THEN
            found ← TRUE
        ELSE
            previous ← current
            current ← (CALL getNext from current)
        END IF
    END WHILE
    IF previous = None THEN
        head of List ← (CALL getNext from current)
    ELSE
        (CALL setNext of previous) ← (CALL getNext of current)
    END IF
END FUNCTION

FUNCTION search(item: Integer)
    current: Node
    found, stop: Boolean
    current ← head of List
    found ← FALSE
    stop ← FALSE
    WHILE current != None AND stop = FALSE DO
        IF (CALL getData from current) = item THEN
            found ← TRUE
        ELSE
            IF (CALL getData from current) > item THEN
                  stop ← TRUE
            ELSE
                  current ← (CALL getNext from current)
            END IF
        END IF
    END WHILE
    RETURN found
END FUNCTION

FUNCTION add(item: Integer)
    current,previous,temp: Node
    stop: Boolean
    current ← head of List
    previous ← None
    stop ← FALSE
    WHILE current != None and stop = FALSE DO
        IF (CALL getData from current) < item THEN
            stop ← TRUE
        ELSE
            previous ← current
            current ← (CALL getNext from current)
        END IF
    END WHILE
    temp ← Node(item)
    IF previous = None THEN
        (CALL setNext in temp) ← head of List
        head of List ← temp
    ELSE
        (CALL setNext in temp) ← current
        (CALL setNext in previous) ← temp
    END IF
END FUNCTION
```