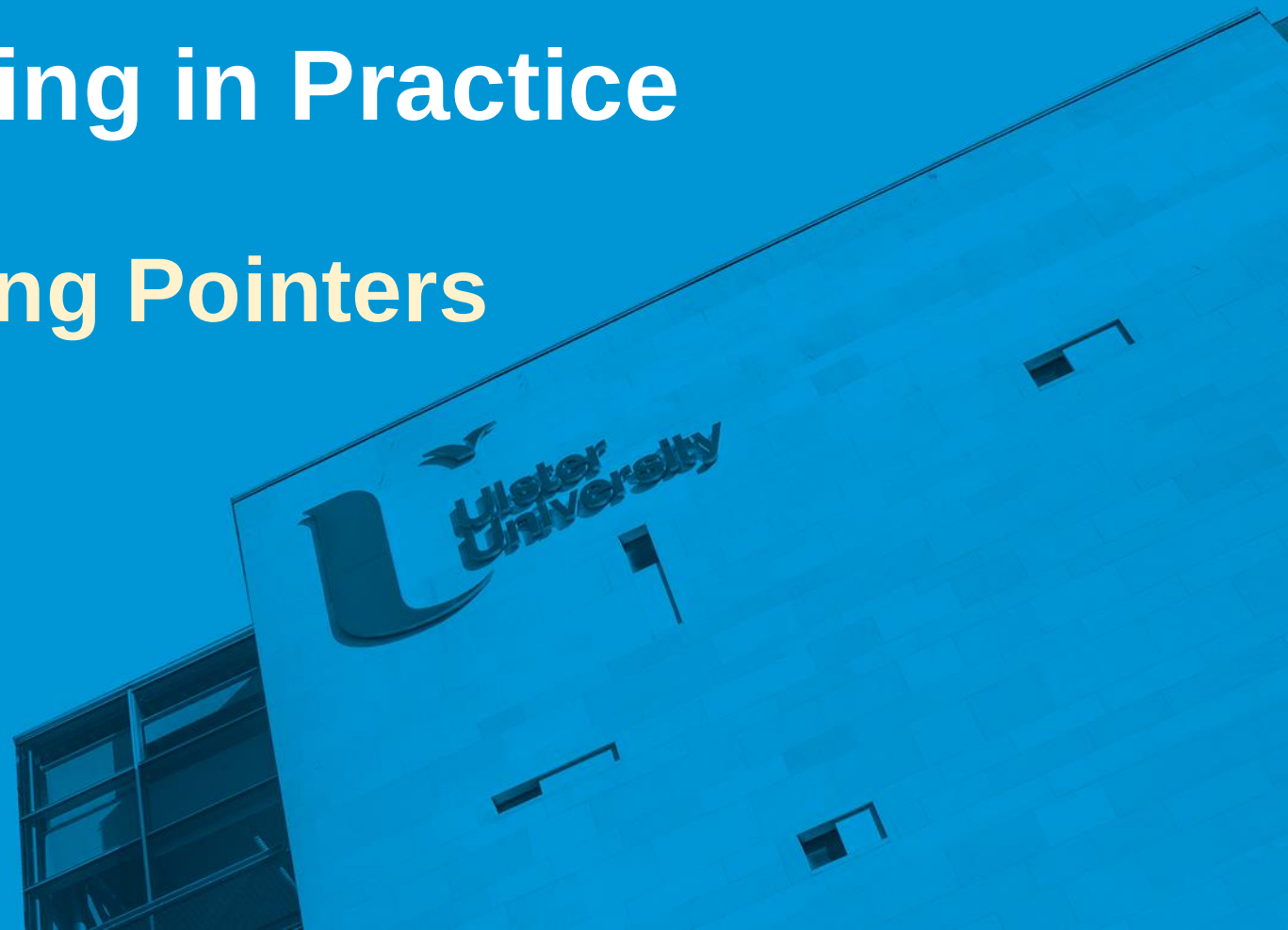
Ulster University

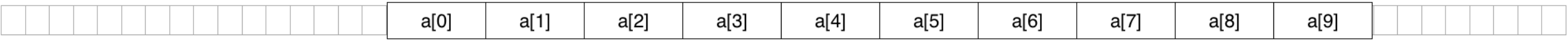
COM410 Programming in Practice

A4.1 Defining and Using Pointers



Problems with Array Implementation

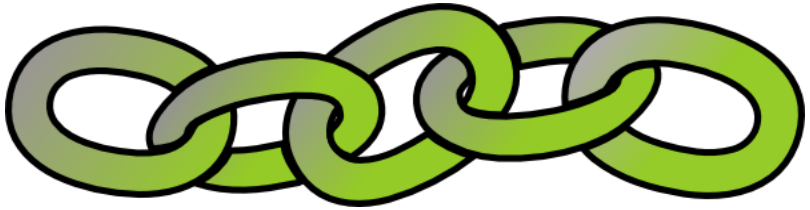
- Previously we used a fixed-size array to implement the `Bag` ADT



- Some (potential) issues with such an implementation:
  - Array has a fixed size
  - The array may become full
  - May have wasted space
  - Resizing is possible but requires overhead of time

Linked Data Organisation

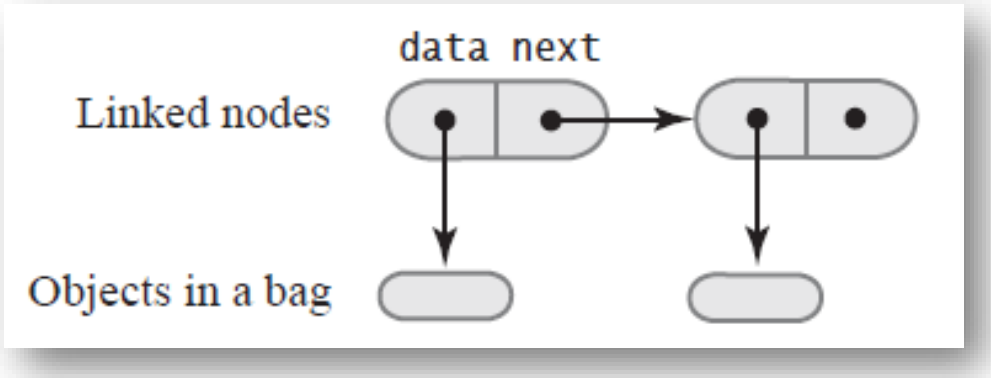
- The section introduces an implementation approach that uses memory only as needed (for a new entry) and returns unneeded memory to the system (after an entry is removed)
- By using a linked data organisation to implement the Bag ADT we avoid moving data when adding or removing bag entries



- A linked list (linked chain) is a linear data structure often used to implement other data structures

Linked Data Organisation

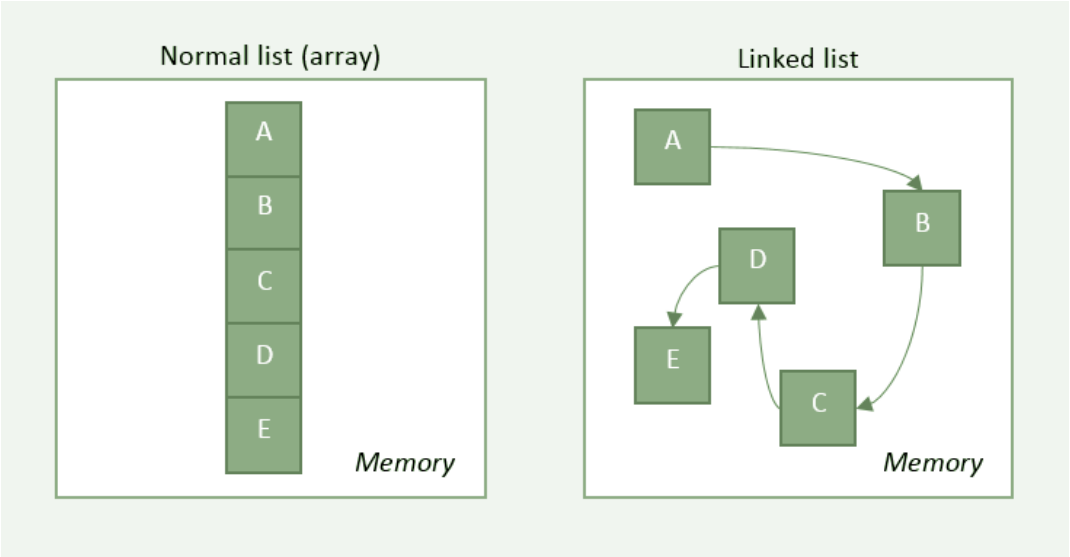
- The linked list (linked chain) is formed from a sequence of nodes
- Each node typically stores a reference (pointer) to a piece of data (an entry in a bag) and a reference to another node (address of the next node in the chain)



- In a linked list the last node points to `null` which signifies the end of the chain

Linked List vs. Array

- A linked list is similar to an array in its approach to sequence and order
- Unlike an array, a linked list:
  - Is not restricted to a fixed-size number of elements
  - Is not stored contiguously in memory
  - Nodes can be inserted and removed without reallocation of memory



- Arrays are quicker at accessing elements
- Arrays are slower at inserting or removing elements
- A linked list can grow and shrink dynamically at run-time

The `Node` Class

- To maintain a linked collection, we first need to define a `Node`
- Class with **two data fields**, **constructor** and both **accessor** and **mutator** methods
- Data field (e.g. `data`) contains a reference to one of the objects in the bag
- Pointer field (e.g. `next`) contains a reference to the next node in the sequence
- Constructor creates a new node setting the `data` field supplied and initialising the `next` field to `null`
- Accessor methods returns the values of the `data` and `next` fields
- Mutator methods set/update the values of the `data` and `next` fields

The Node Class

```
1  public class Node {
2
3      private String data;
4      private Node next;
5
6      public Node(String dataValue) {
7          this.data = dataValue;
8          this.next = null;
9      }
10
11     public String getData() { return this.data; }
12
13     public void setData(String dataValue) { this.data = dataValue; }
14
15     public Node getNext() { return this.next; }
16
17     public void setNext(Node nextNode) { this.next = nextNode; }
18
19 }
```

Instance variables – a data payload (assuming String here) and a pointer to the next node

Constructor to create a new Node

Public methods to

- return the data payload
- set the data payload
- return the next node pointer
- set the next node pointer

Scenario

- In your `Anytown` project, create a new file ***Node.java*** and implement the `Node` class definition where the type of the payload is a `Building` object
  - Test the definition of `Node` by creating a new file ***NodeTest.java*** containing a class `NodeTest` in which the `main()` method defines three nodes called `node1`, `node2` and `node3` with new `Building` objects as the payload of each such that each building has a unique address field.
  - Set the `next` fields of the nodes so that `node1` points to `node2` and `node2` points to `node3`.
  - Now, without referring directly to `node2` or `node3`,write code to print the values of all 3 nodes.