In this lab, you’ll explore simple data structures in assembly. In Java, you’re used to constructing data structures out of classes. The language takes care of the different types and sizes of the data members for you. In assembly, there is no type information so you have to manage all this yourself. Therefore we’ll be exploring simple data structures.

1 Linked Lists

You’ll be implementing a simple Linked List system. The Java code for a linked list node would look like:

```java
public class ListNode
{
    private int _data;
    private ListNode _next;

    public ListNode(int data) { _data = data; _next = null; }

    . . .
}
```

Now, in assembly, we don’t have any notions of type or protection, so we have to imagine how a list node would look in memory. The _data field is just an integer, so it is 4 bytes large. The _next field is a reference to another object, so it is a pointer to somewhere in memory. The ARM is a 32-bit addressable system, so we may need up to 32-bits to address anything in memory, so the _next field can also fit in 4 bytes. Therefore, in memory, our ARM equivalent of the Java class would look like:

```
<table>
<thead>
<tr>
<th>_data</th>
<th>_next</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bytes</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>
```

What does this mean? This means that in assembly, data structures are described completely by the size of their elements. To create a new ListNode, you just allocate 8 bytes and write to it. To access the _data field, you just LDR Rx, [Ry], where Ry holds the address of the ListNode. Similarly, to access _next you just LDR Rx, [Ry, #4].

Once you have ListNodes, all you need to have a real linked list is a pointer to the first node (null if the list is empty). This can be any 4-byte value in the static data area.

For the purposes of this lab, you may assume that the null pointer’s value is 0x0.
1. Write a procedure (named allocnode) that given a data value in R4 and a next value in R5 will allocate a new ListNode on the heap with those values in its fields.

2. Write a procedure (named pushfront) that, given a data value in R4, will allocate a new ListNode on the heap and place it at the beginning of the LinkedList.

3. Write a procedure (named pushback) that, given a data value in R4, will allocate a new ListNode on the heap and place it at the END of the LinkedList.