Problem 4

The idea of this algorithm is to have two pointers, both initialized at the starting node of the list. Then move them at different speeds. If the two pointers eventually collide, then there is a cycle. If a nullpointer is reached, then they do not. More precisely the algorithm is as follows:

1. Create two pointers, fastpointer and slowpointer at the starting node of the list.
2. Check if the next node of the node pointer to by fastpointer is not null. If it is return false. Then check if the next node is the the slowpointer, if yes, return true. otherwise set the fastpointer to the next node.
3. Repeat step 2. If nothing is returned, then in addition to moving the fastpointer to its next node, move the slowpointer to its next node.
4. Repeat until a null pointer is reached, in which case false is returned, or until fastpointer == slowpointer, in which case we return true.

This algorithm runs in linear time because:

1. In the case when the linked list is a not a cycle, we just run through n elements doing a constant amount of operations each time.
2. In the case when the list is a cycle, it only takes n steps for the fastpointer to catch up to the slowpointer, where n is the length of the list.

Here is a c++ implementation of this check:

~~~c++ bool checkIfCycle(Node *node) { Node *slowpointer = node; Node *fastpointer = node; bool isCycle = false; while (true) { if (!fastpointer->next) break; else fastpointer = fastpointer->next;

```
    if (!fastpointer->next)
        break;
    else if (fastpointer == slowpointer)
    {
        isCycle = true;
        break;
    }
    else
    {
        fastpointer = fastpointer->next;
        slowpointer = slowpointer->next;
    }
}
return isCycle;
```

}

~~~