1. (100 pts) Implement the following functions for linked lists. You can assume that all
the nodes in the linked list are distinct and each node appears in the list at most once.

\[
\begin{align*}
\text{node } & \ast \text{delete(node } \ast \text{head, int } k) \\
\text{node } & \ast \text{recursive\_delete(node } \ast \text{head, int } k) \\
\end{align*}
\]

- \textit{delete} deletes the node with info } k \text{ from the linked list and returns the new linked
  list. It returns the linked list without modification if } k \text{ does not appear in the
  list.
- \textit{recursive\_delete} is a recursive function that deletes the node with info } k \text{ from the
  linked list and returns the new linked list. It returns the linked list without
  modification if } k \text{ does not appear in the list.

\text{\textit{This is an in-class exercise and no submission is required.}}

\[
\begin{align*}
\text{node} & \ast \text{ recursive\_delete(node } \ast \text{head, int } k) \\
\text{2. } & \text{if (head } = \text{NULL)} \\
& \quad \text{return (NULL);} \\
& \text{if (head } \rightarrow \text{info } = \text{=} k) \\
& \quad \text{3. node } \ast \text{ ptr } = \text{head } \rightarrow \text{next;} \\
& \quad \text{free(head);} \\
& \quad 3. \text{return (ptr);} \\
& \text{head } \rightarrow \text{next } = \text{delete(ptr } \rightarrow \text{next, } k); \\
& \quad \text{return (head);} \\
\end{align*}
\]
1. (100 pts) Write a function to find the middle node in a linked list and return a pointer to the middle node. Function prototype is given below.

```
node *middleNode(node *list1)
```

If the list has \(2k+1\) elements where \(k\) is an integer \(\geq 0\), return the \((k+1)^{th}\) node from the start. If the list has \(2k\) elements where \(k\) is an integer \(\geq 0\), return the \((k+1)^{th}\) node from the start. For the following linked list the function returns a pointer to node containing 12.

```
list1 → 5 → 6 → 12 → 18 → 21 → Null
```

This is an in-class exercise and no submission is required.

```
node *middleNode(node *list)
{
    node *slow, *fast;
    slow = list;
    fast = list;
    while (fast != NULL && fast->next != NULL)
    {
        slow = slow->next;
        fast = fast->next->next;
    }
    return (slow);
}
```