2) (10 pts) ALG (Linked Lists)

Consider the following code:

```
void doTheThing(node *head, node *current)
{
  if (current == NULL)
    return;

  else if (current == head->next)
  {
    if (current->data == head->next->next->data)
      doTheThing(head, head->next->next->next);
    else if (current->data == head->next->next->data + 1)
      doTheThing(head, head->next->next->next->next);
    else if (current->data == head->next->next->data + 5)
      doTheThing(head, current->next->next->next);
    else if (current->data == head->next->next->data + 10)
      doTheThing(head, head->next);
    else
      doTheThing(head, current->next);
  }

  else
    doTheThing(head, current->next);
}
```

Draw a linked list that simultaneously satisfies **both** of the following properties:

1. The linked list has **exactly four nodes**. Be sure to indicate the integer value contained in each node.

2. If the linked list were passed to the function above, the program would either crash with a segmentation fault, get stuck in an infinite loop, or crash as a result of a stack overflow (infinite recursion).

Note: When this function is first called, the head of your linked list will be passed as *both* arguments to the function, like so:

```
doTheThing(head, head);
```

Hint: Notice that all the recursive calls always pass *head* as the first parameter. So, within this function, *head* will always refer to the actual head of the linked list. The second parameter is the only one that ever changes.

Solution:

The only way to get wrecked with this code is to trigger the `doTheThing(head, head->next)` call. Since we only trigger that call when `current == head->next`, then making that recursive call results in infinite recursion. (Continued on the following page.)

That specific recursive call is only executed when `current == head->next` (i.e., when `current` is the second node in the linked list) and when that second node has a value that is 10 greater than the value in the third node. For example:

```
[1]->[18]->[8]->[3]->

* The first and last values can be anything, but the second value needs to be
exactly 10 greater than the third value.
```

Note that none of the excessive `->next->next->next` accesses would ever cause segfaults here, since we always have four nodes in the linked list, and we never get to those accesses unless *current* is the second node in the linked list. Even the `head->next->next->next->next` access wouldn't cause a segfault; it would just pass NULL to the function recursively, which would hit a base case and return gracefully.

Grading:

10 points for a correct answer.

Note: If the 2nd node has a value 10 less than the 3rd node, still award 10/10.

5 points if the 2nd node has a value 1 greater than the 3rd node, thereby triggering the `head->next->next->next->next` access. That doesn't cause a segfault, but it's an understandable mistake and is the next best thing.

Note: If the 2nd node has a value 1 less than the 3rd node, still award 5/10.

2 points otherwise, as long as they draw a linked list with exactly four nodes, and each node contains an integer. (Also, any circular list with 4 nodes gets 2 points maximum.)

0 points otherwise.

FURTHER GRADING NOTES: A circular list should get at most 2 points. It's clear from the question that the intent is for the list not to be circular but a regular linked list. The reason this is clear is that the way the code is written, we look for the base case with a NULL pointer, but a circular linked list doesn't have one of those. So, ANY circular linked list of size 4 will cause an infinite loop, so one can put any four values down and the second item would be satisfied, which means the second item would be irrelevant. This should help a student realize that the intent was for the answer to be a regular linked list that isn't circular and what's being graded are the specific values they pick for the four nodes. When people refer to a regular linked list, they just say "linked list", they don't say "a linked list that isn't circular and doesn't have links." Instead, the assumption is that unless specified otherwise, a linked list has a head and a single pointer to the next node.