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.

3) (5 pts) ALG (Stacks and Queues)

Consider the following function:

```
void doTheThing(void)
{
  int i, n = 9;  // Note: There are 9 elements in the following array.
  int array[] = {3, 18, 58, 23, 12, 31, 19, 26, 3};

  Stack *s1 = createStack();
  Stack *s2 = createStack();
  Queue *q = createQueue();

  for (i = 0; i < n; i++)
    push(s1, array[i]);

  while (!isEmptyStack(s1))
  {
     while (!isEmptyStack(s1))
       enqueue(q, pop(s1));   // pop element from s1 and enqueue it in q
     while (!isEmptyQueue(q))
       push(s2, dequeue(q));  // dequeue from q and push onto s2

     printf("%d ", pop(s2));  // pop from s2 and print element

     while (!isEmptyStack(s2))
       push(s1, pop(s2));     // pop from s2 and push onto s1
  }
  printf("Tada!\n");

  freeStack(s1);
  freeStack(s2);
  freeQueue(q);
}
```

What will be the <u>exact</u> output of the function above? (You may assume the existence of all functions written in the code, such as *createStack()*, *createQueue()*, *push()*, *pop()*, and so on.)

Solution: `3 18 58 23 12 31 19 26 3 Tada!`

(This function just ends up printing the contents of the array in order.)

Grading:

5 points for the correct output
4 points if their output was simply missing the "Tada!" or if their output was off by one value
2 points if they printed the array in reverse order.
0 points otherwise.

Feel free to award partial credit if you encounter something else that seems reasonable.