链表remove()函数重新设计

1.当前 `remove()` 函数逻辑

```
template<typename T>
void SingleLinkedList<T>::remove()
{
    if (currentPos == nullptr)
        // Do nothing
    else if (size == 1)
    {
        Node* t = head;
        head = nullptr;
        currentPos = nullptr;
        delete t;
        size = 0;
    }
    else if (currentPos->next != nullptr)
    {
        Node* t = currentPos->next;
        currentPos->next = t->next;
        delete t;
        --size;
    }
}
```

2.修改思路

将 `remove()` 函数改为删除当前指向的节点（即 `currentPos` 所指的节点）. 为了达到此效果，我们需要修改函数逻辑。删除当前节点时，需要处理以下几种情况：

1. **链表为空或 `currentPos` 为 `nullptr`**：直接返回。
2. **当前节点是头节点**：需要更新 `head` 指向下一个节点。
3. **当前节点是其他节点**：需要找到当前节点的前驱节点，并更新其指针以跳过当前节点。

修改后函数实现如下：

```
template<typename T>
void SingleLinkedList<T>::remove()
{
    // 如果 currentPos 是 nullptr，什么都不做
    if (currentPos == nullptr)
        return;
```

```
    // 删除头节点的特殊情况
    if (currentPos == head)
    {
        Node* t = head;
        head = head->next;
        currentPos = head;  // 更新 currentPos 为新的头节点
        delete t;
        --size;
    }
    else
    {
        // 查找 currentPos 的前驱节点
        Node* prev = head;
        while (prev->next != currentPos)
        {
            prev = prev->next;
        }

        // 删除当前节点
        prev->next = currentPos->next;
        delete currentPos;
        currentPos = prev->next;  // 更新 currentPos 为下一个节点
        --size;
    }

    // 如果链表为空，则将 currentPos 设置为 nullptr
    if (size == 0)
    {
        currentPos = nullptr;
    }
}
```

这样就实现了`remove()`函数删除当前指向的节点（即`currentPos`所指的节点）

3.其他修改思路

我认为**一种更好的思路**是将`find()`与`remove()`函数的功能相结合，直接实现查找后删除特定节点的功能（`findAndRemove()`函数）。下面给出具体实现：

```
template<typename T>
bool SingleLinkedList<T>::findAndRemove(const T &_val)
{
    // 特殊情况：链表为空
    if (head == nullptr)
        return false;
```

```
    // 如果要删除的是头节点
    if (head->data == _val)
    {
        Node* t = head;
        head = head->next;
        delete t;
        --size;
        currentPos = head;  // 更新 currentPos
        return true;
    }

    // 遍历链表查找节点
    Node* prev = head;
    Node* p = head->next;
    while (p != nullptr)
    {
        if (p->data == _val)
        {
            prev->next = p->next;
            delete p;
            --size;
            currentPos = prev->next;  // 更新 currentPos
            return true;
        }
        prev = p;
        p = p->next;
    }

    return false;  // 如果没有找到目标值
}
```