Report

To improve the `remove()` function, which we hope it can remove both the nodes after it and itself, I think it's necessary to add some class member into `SingleLinkedList` class. Otherwise, we may need to do a search to find the previous node of `currentPos` which cost O(n) time.

1. declaration

So similar to the double linked list, I added a `correntPrev` member which tracks the previous node. Besides `correntPrev`, I added a `sentinel` member which acts as the root of the list and thus we can replace `head` with `sentinel->next`. Otherwise, there're many annoying corner cases to deal with. So the edition to the declaration of the class is like below:

```
template <typename T>
class SingleLinkedList
{
private:
                        ......
    //NEW MEMEBERS
    + Node* currentPrev = nullptr;

    - Node* head = nullptr;
    + Node* sentinel = nullptr;
                        ......
}
```

2. (copy) constructor / deconstructor

Because of the new sentinel for each list, we need to change almost every constructor. I first defined a helper function called `_insert()` , which takes a pointer of new node as the input and insert it after `correntPos`.

```
template<typename T>
void SingleLinkedList<T>::_insert(Node* newNode) {
    this->currentPos->next = newNode;
    this->currentPrev = currentPos;
    this->currentPos = currentPos->next;
    size++;
}
```

Accordingly, we need to initialize `sentinel` in the constructor and free in the deconstructor.

```
//constructors
template<typename T>
SingleLinkedList<T>::SingleLinkedList():sentinel(new Node()) {
    this->currentPos = sentinel;
}

template <typename T>
```

```
SingleLinkedList<T>::SingleLinkedList(std::initializer_list<T> _l)
{
    this->sentinel = new Node();
    this->currentPos = sentinel;
    for (auto i = _l.begin(); i != _l.end(); ++i)
    {
        Node* newNode = new Node(*i);
        _insert(newNode);
    }
}

//deconstructor
template <typename T>
SingleLinkedList<T>::~SingleLinkedList()
{
    _emptyList();
    if (sentinel != nullptr)
        delete sentinel;
};

//copy constructor
template<typename T>
SingleLinkedList<T>::SingleLinkedList(const SingleLinkedList<T> &_l)
{
    sentinel = new Node(NULL);
    _copy(_l);
}
```

3. Find

We need to update `currentPrev` whenever the `currentPos` is changed. So we also need to modify `find()`, which is not a const member function.

```
template<typename T>
bool SingleLinkedList<T>::find(const T &_val)
{
    Node* prev = sentinel;
    Node *p = sentinel->next;
    while (p != nullptr)
    {
        if (p->data == _val)
        {
            currentPos = p;
            currentPrev = prev;
            return true;
        }
        prev = prev->next;
        p = p->next;
    }
    return false;
}
```

4. homework functions

And at last the implement of the homework is like below:

```
//TO DO
template<typename T>
T SingleLinkedList<T>::getCurrentVal() const {
    //T getCurrentVal() const:返回当前位置的值
    //若当前位置为空，则报错"Empty current position! Can't get value!"
    //并停止退出
    if (!this->currentPos) {
        std::cout << "Empty current position! Can't get value!" << std::endl;
        return NULL;
    }
    return this->currentPos->data;
}

template<typename T>
void SingleLinkedList<T>::setCurrentVal(const T& _val) {
    //设置当前位置的值,若当前位置为空
    //则报错"Empty current position! Can't setvalue!"并停止退出;
    if (!this->currentPos) {
        std::cout << "Empty current position! Can't setvalue!" << std::endl;
        return;
    }
    this->currentPos->data = _val;
}

template<typename T>
void SingleLinkedList<T>::insert(T _val) {
    //若为空链表则插入一个元素；若非空，则在当前位置后插入一个元素，值为_val
    Node* n = new Node(_val);
    n->next = currentPos->next;
    _insert(n);
}

template<typename T>
bool SingleLinkedList<T>::isEmpty() const {
    return !this->size;
}


template<typename T>
void SingleLinkedList<T>::remove() {
//若当前位置为空则不进行操作；若链表仅有一个元素则删除此元素成为空链表；
//若链表有超过一个元素且当前位置不为最后一个，则删除后一个元素.
    if (this->currentPos != sentinel) {
        this->currentPrev->next = this->currentPos->next;
        delete  currentPos;
        this->currentPos = this->currentPrev;
        size--;
    }
}
```

5. test

Run the test program provided, the output is shown below. Notice that the last line of the output is somehow different with the given solution in the standard of scores. But I think it's right if `remove()` removes `currentPos`.

```
PS E:\c++\DSHW\data_structures\LinkedList> g++ -o test ./main.cpp; ./test
In file included from ./main.cpp:1:
./LinkedList.h: In instantiation of 'T SingleLinkedList<T>::getCurrentVal() const [with T = int]':
./main.cpp:5:67:   required from here
./LinkedList.h:187:16: warning: converting to non-pointer type 'int' from NULL [-Wconversion-null]
  187 |         return NULL;
      |                ^~~~
5
a       c       g
0 1
3
1.5     2       9       4.7     8.8
1.5     2       8.8     9       4.7


1.5     2       4.7     8.8
```

And run valgrind on linux:

```
cierra@cierra-runis:/mnt/hgfs/DSHW-2/data_structures/LinkedList$ make clean
rm -f main.o test
cierra@cierra-runis:/mnt/hgfs/DSHW-2/data_structures/LinkedList$ make linux_run
g++ -std=c++11 -Wall -O2 -I. -c main.cpp -o main.o
main.cpp: In function ‘int main()’:
main.cpp:21:10: warning: unused variable ‘f_c3’ [-Wunused-variable]
   21 |     bool f_c3 = c3.find(2.0);
      |          ^~~~
main.cpp:28:10: warning: unused variable ‘f’ [-Wunused-variable]
   28 |     bool f = c2.find(9.0);
      |          ^
In file included from main.cpp:2:
LinkedList.h: In instantiation of ‘T SingleLinkedList<T>::getCurrentVal() const [with T = int]’:
main.cpp:6:32:   required from here
LinkedList.h:187:16: warning: converting to non-pointer type ‘int’ from NULL [-Wconversion-null]
  187 |         return NULL;
      |                ^~~~
g++ -std=c++11 -Wall -O2 main.o -o test
./test
5
a       c       g
0 1
3
1.5     2       9       4.7     8.8
1.5     2       8.8     9       4.7


1.5     2       4.7     8.8
cierra@cierra-runis:/mnt/hgfs/DSHW-2/data_structures/LinkedList$ valgrind ./test
==4633== Memcheck, a memory error detector
==4633== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4633== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==4633== Command: ./test
==4633==
5
a       c       g
0 1
3
1.5     2       9       4.7     8.8
1.5     2       8.8     9       4.7


1.5     2       4.7     8.8
==4633==
==4633== HEAP SUMMARY:
==4633==     in use at exit: 0 bytes in 0 blocks
==4633==   total heap usage: 27 allocs, 27 frees, 74,128 bytes allocated
==4633==
==4633== All heap blocks were freed -- no leaks are possible
==4633==
==4633== For lists of detected and suppressed errors, rerun with: -s
==4633== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cierra@cierra-runis:/mnt/hgfs/DSHW-2/data_structures/LinkedList$
```