**12.链表转数组线性表**

1)

/\*

\* brief：将链表转换为数组线性表

\* in:

\* chain<T>& raw 传入链表引用

\* out:

\* arrayList<T>\* 返回复制后数组线性表的指针

\*/

template<class T>

arrayList<T>\* chainToListGet ( chain<T>& raw ) {

/\*建立列表\*/

arrayList<T>\* temp = new arrayList<T>;

/\*进行循环复制\*/

int size = raw.listSize ();

for ( int i = 0; i < size; i++ ) {

temp->insert ( i, raw.get ( i ) );

}

/\*返回列表指针\*/

return temp;

}

复杂度：

O(n!)

除get，insert操作外，其他操作复杂度为O(1)，get(i)与insert(i,element)操作复杂度为O(i)，两操作均执行size次，不同次调用复杂度依次递增

故总体复杂度为O(n!)

测试：



2)

/\*

\* brief：使用迭代器将链表转换为数组线性表

\* in:

\* chain<T>& raw 传入的链表指针

\* out:

\* arrayList<T>\* 返回复制后数组线性表的指针

\*/

template<class T>

arrayList<T>\* chainToListIterator ( chain<T>& raw ) {

/\*建立列表\*/

arrayList<T>\* temp = new arrayList<T>;

/\*进行循环复制\*/

int size = raw.listSize ();

//要遍历 size 个值，迭代器要自增 size-1 ，因此首先对第一个元素进行复制

chain<int>::iterator it = raw.begin ();

temp->insert ( 0, \*it );

//再对剩余元素进行复制

for ( int i = 1; i < size; i++ ) {

it++;

temp->insert ( i, \*it );

}

/\*返回列表指针\*/

return temp;

}

复杂度：

O(n)

获取长度，迭代器自增、取值等操作的复杂度均为O(1)

函数中只有一个一层的n-1次for循环，因此复杂度为O(n)

测试：



**15.翻转-成员函数**

1)

void reverse () {

/\*链表为空或只有一个元素则直接返回\*/

if ( head->next == nullptr || head->next->next == nullptr ) {

return;

}

/\*翻转准备工作\*/

node\* p = head->next; //p 在前， q 在后，q->next指向p，达到元素指向翻转的目的

node\* q = head->next->next;

node\* temp = nullptr; //temp指向q的下一个元素

//在p、q指针交替翻转指向性后，temp指针赋值给q指针，q指针赋值给p指针

//进行新一轮循环

/\*翻转的主要工作\*/

while ( q != nullptr ) {

temp = q->next;

q->next = p; //翻转指向

p = q; //p、q后移一位

q = temp;

}

/\*翻转的扫尾工作\*/

head->next->next = head; //老二号元素现在在表位，应当将老二号元素后面接上老表头

head->next = nullptr; //在上一条语句中老表头成为新表尾，这里将next置空

head = p; //更新表头

return;

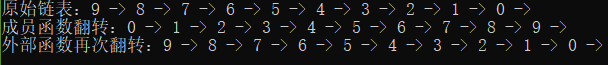
}

2)复杂度

准备工作与扫尾工作均为O(1)复杂度，翻转主要工作为n次for循环

因此，函数复杂度为O(n)。

3)测试



**16.翻转-外部函数**

1)

/\*

\* brief：建立新的翻转链表，使指针指向新链表

\* in: chain<T>\*\* in

\* out: void

\*/

template<class T>

void reverse ( chain<T>\*\* in ) { //需要传入指针的指针，通过指针的指针对指针进行操作；

chain<T>\* temp = new chain<T>;

/\*翻转复制的主体部分\*/

int size = ( \*in )->listSize ();

chain<int>::iterator it = ( \*in )->begin ();

temp->insert ( 0, \*it ); //使用迭代器，访问 size 个元素需要自增 size-1 次

//因此提前复制一次，接下来自增与复制循环 size-1 次即可

for ( int i = 1; i < size; i++ ) { //自增与复制操作循环 size-1 次

it++;

temp->insert ( 0, \*it );

}

/\*翻转复制的扫尾部分\*/

delete ( \*in ); //删除指针指向的旧链表，并让它指向新链表

\*in = temp;

return;

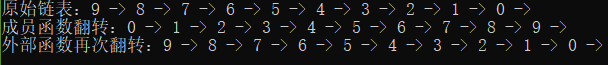
}

2)复杂度

准备与扫尾复杂度为O(1)，主体部分迭代器自增n-次，取值n次，自增与取值操作复杂度为O(1)

因此，函数整体复杂度为O(n)。

3)测试



**19.有序链表合并-非成员函数-不清空原链表**

/\*

\* brief：合并两个链表，返回合并后链表的指针

\* in:

\* chain<int>& a 传入的两个链表

\* chain<int>& b

\* out:

\* chain<int>\* 为减少复制的资源消耗，此处返回指针

\*/

chain<int>\* merge ( chain<int>& chain1, chain<int>& chain2) {

/\*复制准备部分\*/

//

chain<int>\* temp = new chain<int>;

chain<int>::iterator it1 = chain1.begin ();

chain<int>::iterator it2 = chain2.begin ();

/\*复制主体部分\*/

//两链表长度未知，因此有一条复制完毕即结束交替复制

//再之后，单独判断并复制未复制完的链表

while ( ( it1 != chain1.end () ) && ( it2 != chain1.end () ) ) {

if ( \*it1 <= \*it2 ) {

temp->insert ( temp->listSize (), \*it1 );

it1++;

} else {

temp->insert ( temp->listSize (), \*it2 );

it2++;

}

}

/\*复制扫尾部分\*/

//如果index等于listsize，说明该数组复制完毕；否则未复制完毕，继续复制剩余元素。

//chain1，chain2线性表必只剩一个没有复制完毕，因此此处两个判断只有一个能执行

if ( it1 != chain1.end () ) {

while ( it1 != chain1.end () ) {

temp->insert ( temp->listSize (), \*it1 );

it1++;

}

}

if ( it2 != chain1.end () ) {

while ( it2 != chain1.end () ) {

temp->insert ( temp->listSize (), \*it2 );

it2++;

}

}

return temp;

}

2)复杂度

对于两个链表分别来说，有size个元素，迭代器就自增size-1次，取值size次

对合并后的新链表来说，两个子链表总共有size个元素，新链表就插入size次

插入下标为0时，复杂度为O(1)

3)测试



**20.有序链表合并-成员函数-清空原链表**

chain<int>& merge ( chain<int>& a, chain<int>& b ) {

/\*建立新链表\*/

chain<int>\* temp = this;

chain<int>::iterator ai = a.begin ();

chain<int>::iterator bi = b.begin ();

while ( ( ai != a.end () ) && ( bi != a.end () ) ) {

if ( \*ai <= \*bi ) {

temp->insert ( temp->listSize (), \*ai );

ai++;

} else {

temp->insert ( temp->listSize (), \*bi );

bi++;

}

}

//如果index等于listsize，说明该数组移动完毕；否则未复制完毕，继续复制剩余元素。

//a，b线性表必只剩一个没有复制完毕，因此此处两个判断只有一个能执行

if ( ai != a.end () ) {

while ( ai != a.end () ) {

temp->insert ( temp->listSize (), \*ai );

ai++;

}

}

if ( bi != a.end () ) {

while ( bi != a.end () ) {

temp->insert ( temp->listSize (), \*bi );

bi++;

}

}

return \*this;

}

2)复杂度

19与20题代码几乎相同，但是20题增加了清空操作，清空操作复杂度为O(n)

总体复杂度仍然为O(n)

3)测试

