LinkNode::LinkNode(const LinkNode& p) : data(p.data) {} // 拷贝构造

LinkNode::LinkNode(LinkNode&& p) : data(p.data) // 移动构造

{

p.data = NULL;

next = p.next;

p.next = nullptr;

}

重载《《

ostream& operator<<(ostream& cout, LinkNode\* pCurrent)

{

cout << pCurrent->data << " ";

return cout;

}

bool operator>(const LinkNode& node1, const LinkNode& node2)

{

return node1.data > node2.data;

//if (node1.data > node2.data) return true;

//else return false;

}

// 重置前置++

LinkNode& LinkNode::operator++() // 前置++返回引用

{

this->data++;

return \*this;

}

// 重置后置++

LinkNode LinkNode::operator++(int) // 后置++返回值

{

LinkNode temp = \*this;

this->data++;

return temp;

}

LinkNode& LinkNode::operator+=(const LinkNode& node) {

this->data += node.data;

return \*this;

反转

LinkNode\* pPrev = nullptr;

LinkNode\* pCurrent = header->next;

LinkNode\* pNext = nullptr;

while (pCurrent != nullptr)

{

pNext = pCurrent->next;

pCurrent->next = pPrev;

pPrev = pCurrent;

pCurrent = pNext;

}

header->next = pPrev;

重载=

MyClass& operator=(const MyClass& t1) {

v = t1.v;

return \*this;

}

重载+

Number operator+(Number& n1, Number& n2) // 建议使用全局函数进行运算符的重载，一个运算符可以进行多次重载

{

Number temp;

temp.m\_A = n1.m\_A + n2.m\_A;

temp.m\_B = n2.m\_A + n2.m\_B;

return temp;

1. *// Function to Merge Arrays L and R into A.*
2. *// lefCount = number of elements in L*
3. *// rightCount = number of elements in R.*
4. void Merge(int \*A,int \*L,int leftCount,int \*R,int rightCount) {
5. int i,j,k;
7. *// i - to mark the index of left aubarray (L)*
8. *// j - to mark the index of right sub-raay (R)*
9. *// k - to mark the index of merged subarray (A)*
10. i = 0; j = 0; k =0;
12. while(i<leftCount && j< rightCount) {
13. if(L[i] < R[j]) A[k++] = L[i++];
14. else A[k++] = R[j++];
15. }
16. while(i < leftCount) A[k++] = L[i++];
17. while(j < rightCount) A[k++] = R[j++];
18. }