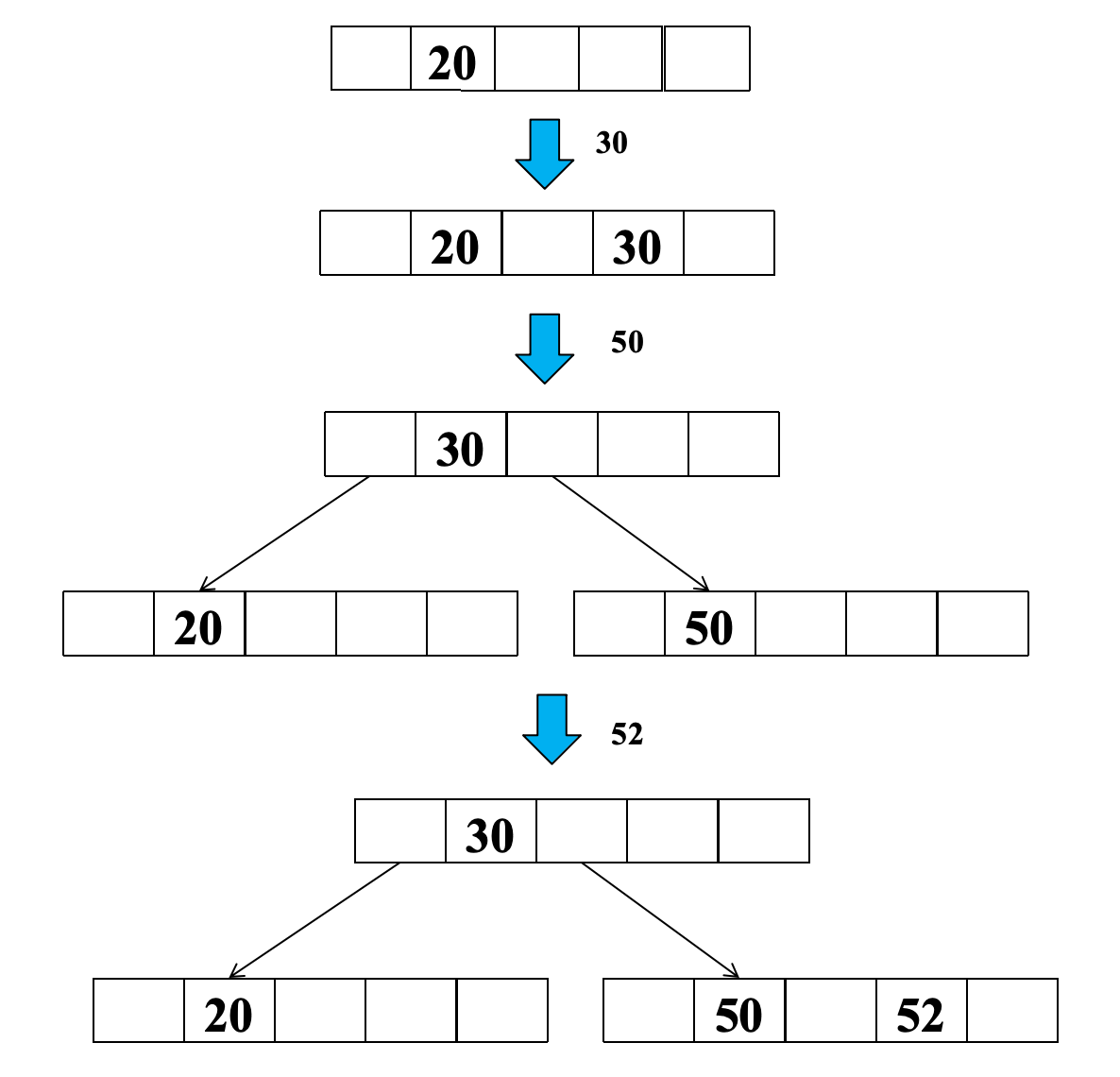
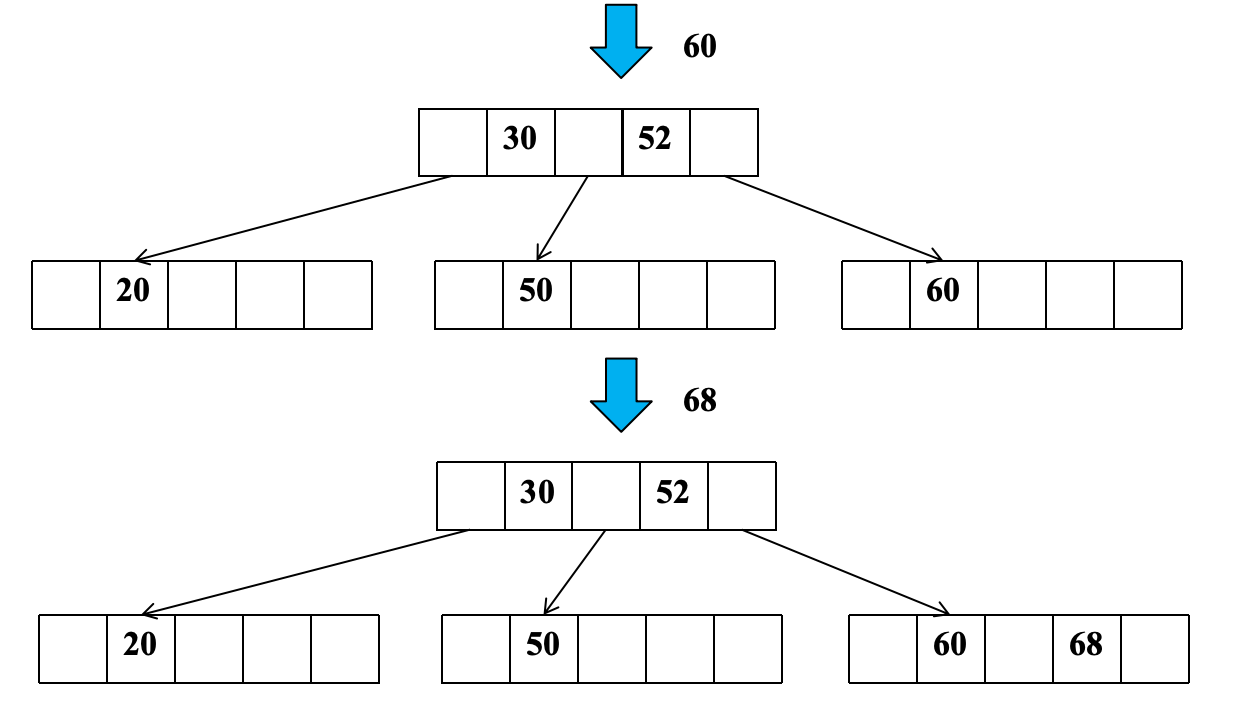
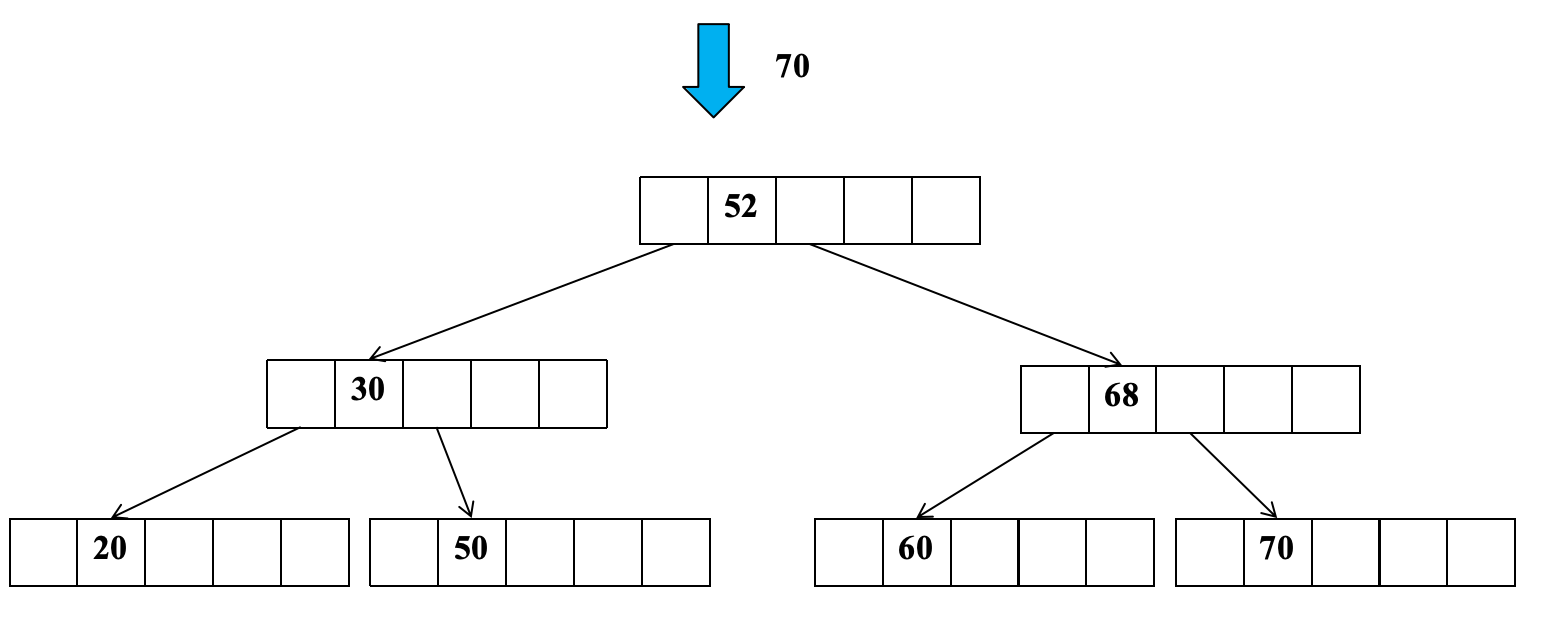
**27. 试从空树开始，按以下顺序向2-3树中插入关键字建树：20，30，50，52，60，68，70。如果此后再删去50和68，画出每一步执行后的状态。**

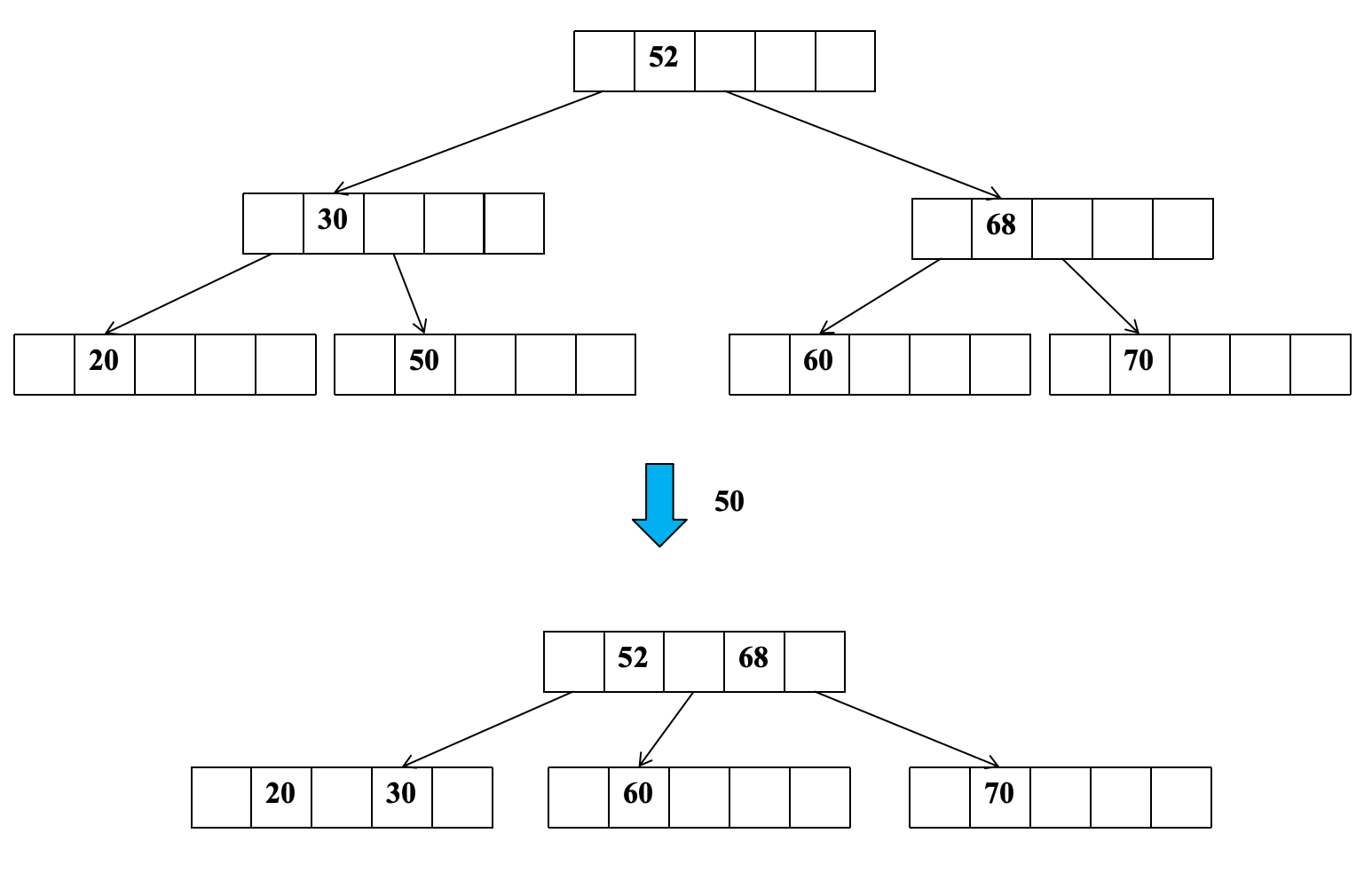
插入：

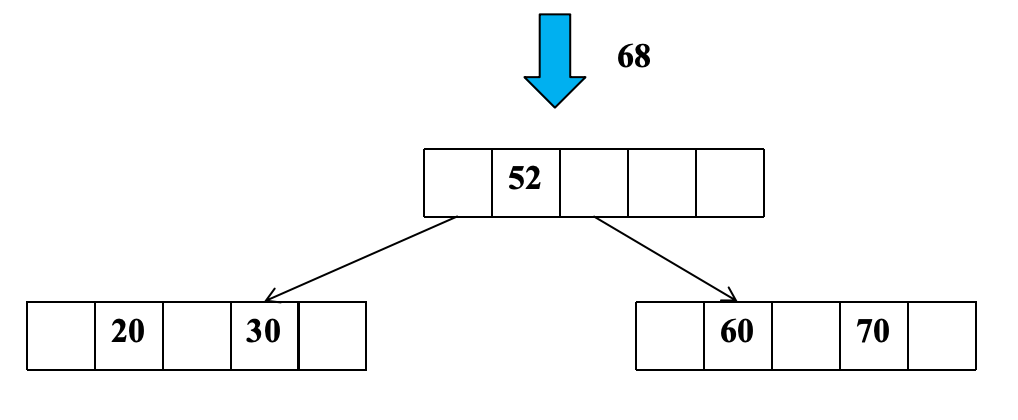
****

****

****

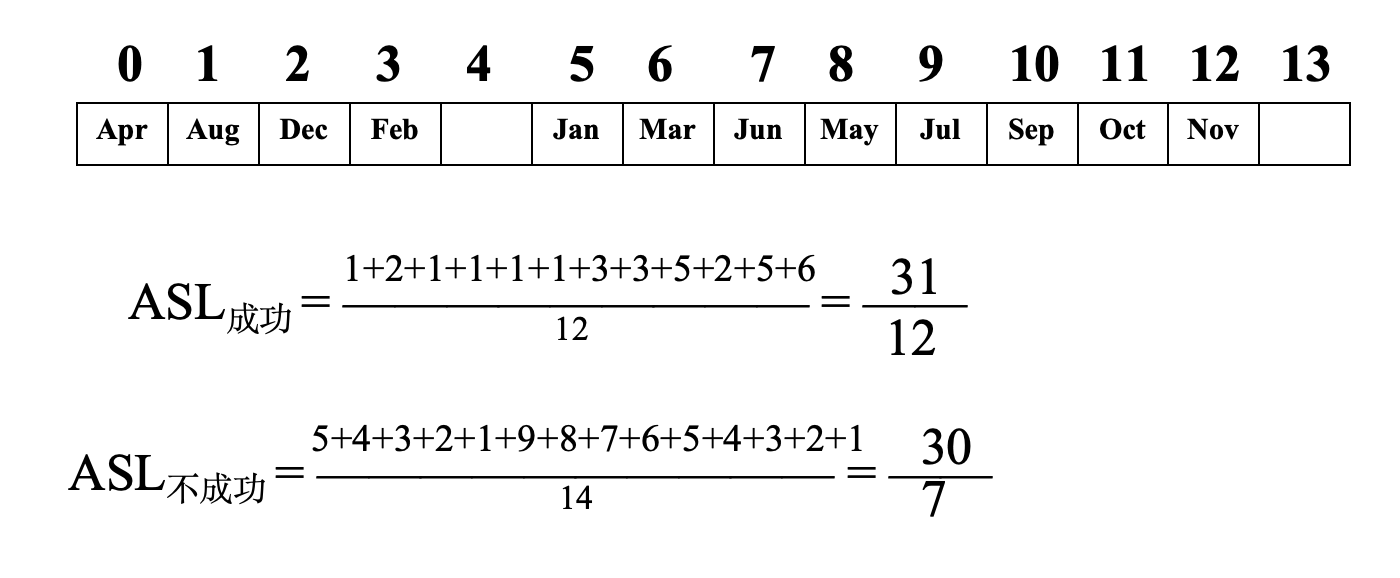
删除：

****

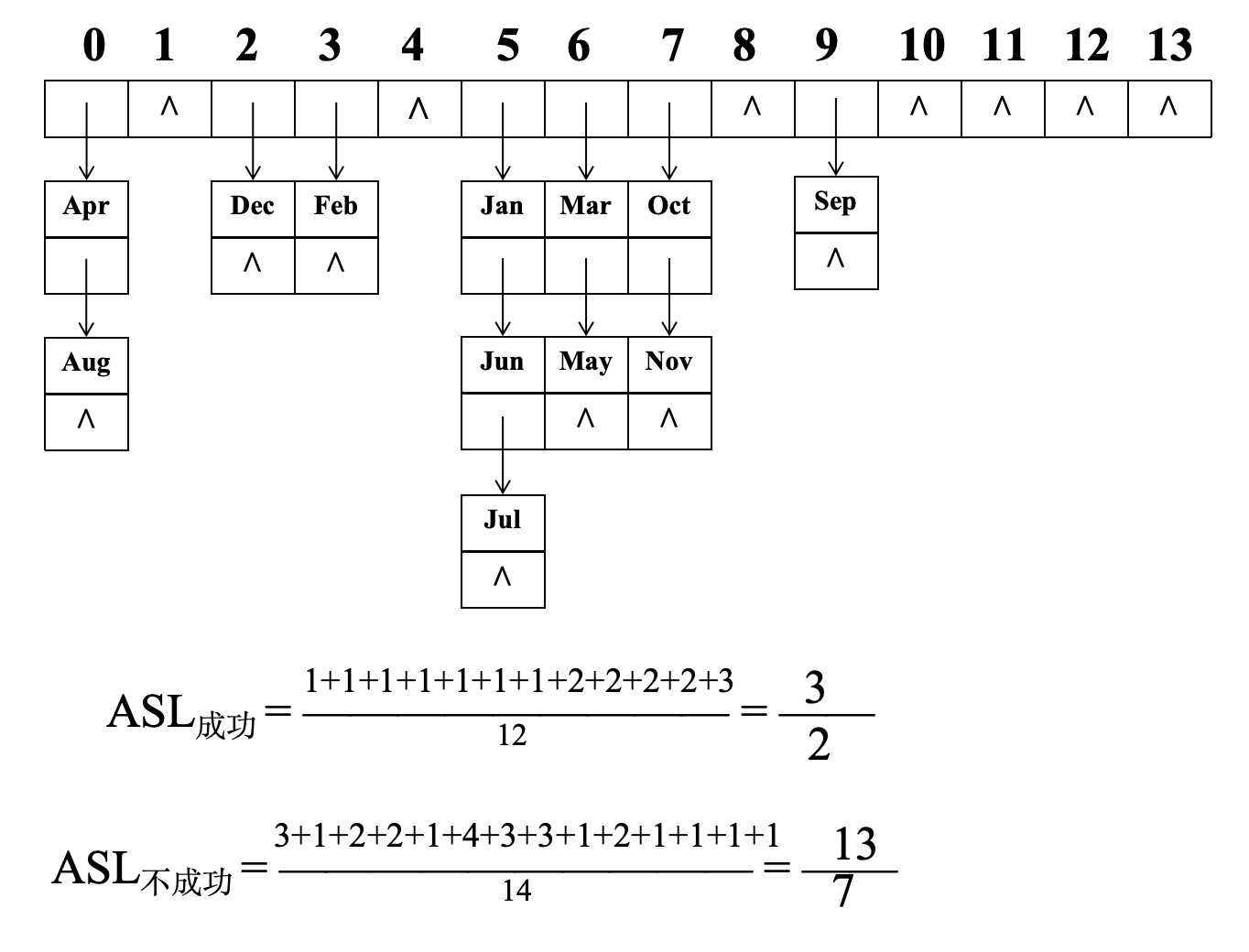
****

1. **在地址空间为0~16的散列区中，对关键字序列（Jan、Feb、…、Dec）造两个哈稀表：H(x) = i div 2 ，其中 i 为关键字中第一个字母在字母表中的序号，分别用线性探测法和链地址法处理冲突，并分别求这两个哈稀表在等概率情况下查找成功、不成功时的平均查找长度。**

线性探测法：

****

链地址法：

****

**29. 试证明：当待排序列已呈现出有序状态时，快速排序的时间复杂度为 O ( n2 )。**

当待排序列有序时，每次选取的pivot都是最后一个元素，每次只需递归处理左边的n-1个元素的子序列。

n+(n-1)+(n-2)+...+1=(n2+n)/2

因此，时间复杂度为O(n2)。

**30. 试以单链表为存储结构实现简单插入排序。**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node \*next;

};

// 创建链表并返回头结点

Node \*initNodeList(void)

{

Node \*head = new Node{-1, NULL}; // 头结点

Node \*tail = head; // 尾指针

int num;

cout << "请输入链表数据域（以 -1 结束）：" << endl;

cin >> num;

while (num != -1)

{

Node \*newNode = new Node{num, NULL}; // 新结点

tail->next = newNode, tail = newNode;

cin >> num;

}

return head;

}

// 从小到大排序

void simpleInsertSort(Node \*head)

{

Node \*cur = head->next;

while (cur != NULL)

{

bool flag = false;

Node \*srh = head;

while (srh != cur)

{

if (cur->next && srh->next->data > cur->next->data) // 交换

{

Node \*temp = cur->next->next;

cur->next->next = srh->next, srh->next = cur->next, cur->next = temp;

flag = true;

break;

}

srh = srh->next;

}

if (flag == false)

cur = cur->next;

}

return;

}

// 遍历输出链表数据

void printNodeList(Node \*head)

{

Node \*cur = head->next;

if (cur)

{

cout << "排序后的链表为：" << endl;

while (cur->next != NULL)

{

cout << cur->data << ' ';

cur = cur->next;

}

cout << cur->data << endl;

}

return;

}

int main()

{

Node \*head = initNodeList();

simpleInsertSort(head);

printNodeList(head);

return 0;

}

测试数据：

