**C++第四次实验报告——动态存储与链表**

**22920212204392 黄勖**

**一、名词解释**

1、动态存储与静态存储

[解释] **动态存储：*在程序运行期间***根据需要进行动态的分配存储空间的方式。动态存储变量是在程序执行过程中，使用它时才分配存储单元，使用完毕立即释放。典型的例子是函数的形式参数，在函数定义时并不给形参分配存储单元，只是在函数被调用时，才予以分配， 调用函数完毕立即释放。如果一个函数被多次调用，则反复地分配、 释放形参变量的存储单元。

**静态存储**：***在程序编译时***分配固定的存储空间的方式。该存储方式通常是在变量定义时就分定存储单元并一直保持不变， 直至整个程序结束。全局变量，静态变量等就属于此类存储方式。

**内存中用户存储空间的分配情况（三种）：**

1. 程序区：存放程序语句
2. 静态存储区：存放全局变量，在程序编译时给全局变量分配存储区，程序执行完毕就释放。
3. 动态存储区：存放以下数据：函数形式参数。在调用函数时给形参分配存储空间；自动变量(未加static声明的局部变量) ；函数调用时的现场保护和返回地址等；

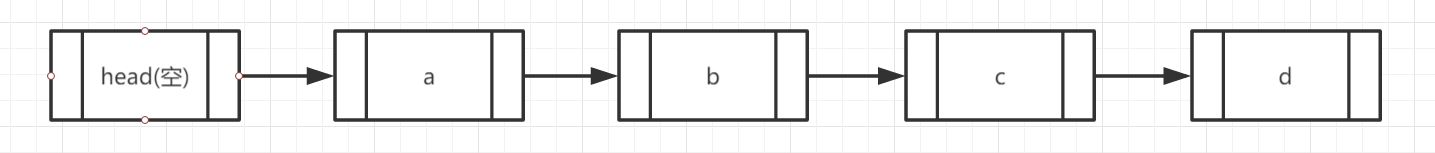
**动态内存分配：**发生在程序调入和执行的时候，动态地分配或者回收存储空间的分配内存的方法。（堆一定是动态分配的）

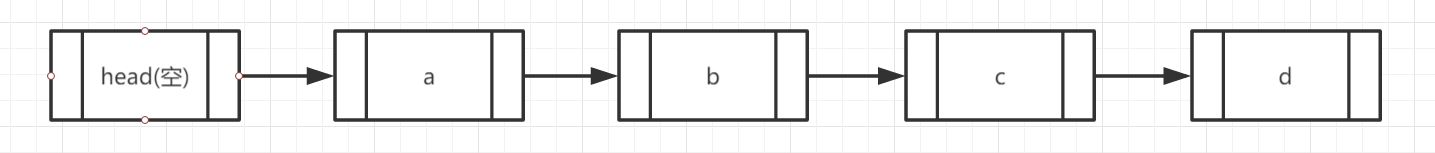
**静态内存分配：**发生在程序编译和连接的时候，预先分配存储空间（栈可以是静态分配如全局变量，也可以是动态分配**，**栈动态分配是由编译器进行释放）

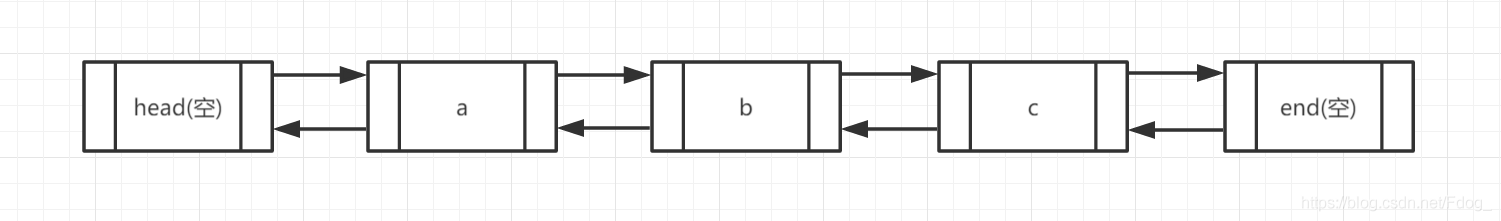
1. 动态数组

[解释] 动态数组，是相对于静态数组而言。静态数组的长度是预先定义好的，在整个程序中，一旦给定大小后就无法改变。而动态数组则不然，它可以随程序需要而重新指定大小。动态数组的内存空间是从堆（heap）上分配（即动态分配）的。是通过执行代码而为其分配存储空间。当程序执行到这些语句时，才为其分配。程序员自己负责释放内存。

1. 单链表、循环链表、双向链表

[辨析] 单链表是一种单向遍历的数据结构，它包含两个域，一个信息域和一个指针域。这个链接指向表中的下一个节点，而最后一个节点则指向一个空值NULL。找一个节点的时候需要从第一个节点开始每次访问下一个节点，一直访问到需要的位置。也可以提前把一个节点的位置另外保存起来，然后直接访问。

循环链表首节点和末节点被连接在一起。这种方式在单向和双向链表中皆可实现。要转换一个循环链表，你开始于任意一个节点然后沿着列表的任一方向直到返回开始的节点。循环链表可以被视为"无头无尾"，循环链表中没有NULL指针，涉及遍历时，终止条件不再是单链表的P！=NULL；而是判断他们是否等于某一个特定的指针，单链表只能从已知结出发，访问其后续结点，而循环链表从已知结点出发，可以访问链表中所有结点。

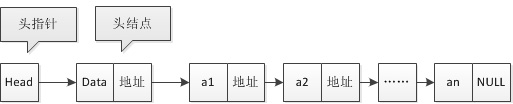
双向链表中不仅有指向后一个节点的指针，还有指向前一个节点的指针。第一个节点的"前连接"指向NULL，最后一个节点的"后连接"指向NULL。这样可以从任何一个节点访问前一个节点，也可以访问后一个节点，以至整个链表。一般是在需要大批量的另外储存数据在链表中的位置的时候用。由于另外储存了指向链表内容的指针，并且可能会修改相邻的节点，有的时候第一个节点可能会被删除或者在之前添加一个新的节点。这时候就要修改指向首个节点的指针。

1. 头结点、首结点、头指针

[辨析] 头结点是为了操作的统一与方便而设立的，放在第一个元素结点之前，其数据域一般无意义（当然有些情况下也可存放链表的长度、用做监视哨等等）。

首结点也就是第一个元素的结点，它是头结点后边的第一个结点。

链表中第一个结点的存储位置叫做头指针，如果链表有头结点，那么头指针就是指向头结点数据域的指针。



1. 内存泄漏、悬浮指针

[解释]

内存泄漏：一般我们常说的内存泄漏是指堆内存的泄漏。堆内存是指程序从堆中分配的，大小任意的（内存块的大小可以在程序运行期决定），使用完后必须显示释放的内存。应用程序一般使用malloc，realloc，new等函数从堆中分配到一块内存，使用完后，程序必须负责相应的调用free或delete释放该内存块，否则，这块内存就不能被再次使用，我们就说这块内存泄漏了。

悬浮指针：声明了但没有被赋值的指针，它指向内存中的任意一个空间。避免悬浮指针的一个方法是开始就赋值为NULL

**二、填空题**

在存储数据时，有顺序存储（如数组等）和链式存储（如链表），请回答：

1、当线性表的元素总数基本稳定，且很少进行插入和删除操作，但要求以最快的速度存取线性表中的元素时，应采用（ **顺序** ）存储结构。

2、在一个长度为n的顺序表中第i个元素（1<=i<=n）之前插入一个元素时，需向后移动（ **n-i+1** ）个元素。

3、顺序存储结构是通过（ **元素在存储器中的相对位置** ）表示元素之间的关系的；链式存储结构是通过（ **元素在存储器中的位置的指针** ）表示元素之间的关系的。

**三、选择题**

1、new是C++语言提供的用于动态数据生成的（ B ），返回值为（ **C** ）。

A 函数 B 运算符 C void \*指针 D 与new后面的类型一致的类型

2、链表不具有的特点是（ **B** ）。

A 插入、删除不需要移动元素 B 可随机访问任一元素

C 不必事先估计存储空间 D 所需空间与线性长度成正比

3、在单链表指针为p的结点之后插入指针为s的结点，正确的操作是（ **B** ）。

A s->next=s;s->next=p->next; B s->next=p->next;p->next=s;

C p->next=s;p->next=s->next; D p->next=s->next; p->next=s;

4、对于一个头指针为head的带头结点的单链表，判定该表为空表的条件是（ **B** ）。

A head==NULL B head->next==NULL C head->next==head D head!=NULL

**四、简答题**

1、比较静态数组和动态数组之间的异同点。

同：都是一种可以随机访问数据的数据结构。

异：**静态数组**在编译时必须知道其长度即需用常量表达式指定数组长度，数组只在定义它的块语句内存在，其内存在使用结束后由计算机自动释放，它的效率较高。

**动态数组**在运行时动态地分配数组。虽然数组长度是固定的，但动态分配的数组不必在编译时知道其长度，可以在运行时才确定数组长度。与数组变量不同，动态分配的数组将一直存在，直到程序释放它为止。

2、在比较顺序存储（如数组等）和链式存储（如链表）特性的基础上，重点分析他们的应用情景（适合解决什么问题）。

**在存储特性上：**

顺序存储的线性表称为顺序表，顺序表中的存储元素是连续的

优点是访问方便、内存利用率高；缺点是内存分配不够灵活、插入删除不方便。

链式存储的线性表称为链表，链表中的存储元素不一定是连续的，元素节点中存放数据元素以及相邻元素的地址信息（链表可以充分利用碎片内存）

优点是内存分配灵活、插入删除方便；缺点是访问不方便、内存利用率低。

**应用场景：**

**总结：**

顺序：频繁的查找，较少删除插入

链式：频繁插入删除、对空间使用情况不确定

**分析：**

　　（1）顺序表的存储空间是静态分配的，在程序执行之前必须明确规定它的存储规模，也就是说事先对“MaxSize”要有合适的设定，设定过大会造成存储空间的浪费，过小造成溢出。因此，当对线性表的长度或存储规模难以估计时，不宜采用顺序表。然而，链表的动态分配则可以克服这个缺点。链表不需要预留存储空间，也不需要知道表长如何变化，只要内存空间尚有空闲，就可以再程序运行时随时地动态分配空间，不需要时还可以动态回收。因此，当线性表的长度变化较大或者难以估计其存储规模时，宜采用动态链表作为存储结构。

但在链表中，除数据域外海需要在每个节点上附加指针。如果节点的数据占据的空间小，则链表的结构性开销就占去了整个存储空间的大部分。当顺序表被填满时，则没有结构开销。在这种情况下，顺序表的空间效率更高。由于设置指针域额外地开销了一定的存储空间，从存储密度的角度来讲，链表的存储密度小于1.因此，当线性表的长度变化不大而且事先容易确定其大小时，为节省存储空间，则采用顺序表作为存储结构比较适宜。

　　（2）基于运算的考虑（时间）

　　顺序存储是一种随机存取的结构，而链表则是一种顺序存取结构，因此它们对各种操作有完全不同的算法和时间复杂度。例如，要查找线性表中的第i个元素，对于顺序表可以直接计算出a(i)的的地址，不用去查找，其时间复杂度为0(1).而链表必须从链表头开始，依次向后查找，平均需要0(n)的时间。所以，如果经常做的运算是按序号访问数据元素，显然顺表优于链表。

　　反之，在顺序表中做插入，删除时平均移动表中一半的元素，当数据元素的信息量较大而且表比较长时，这一点是不应忽视的；在链表中作插入、删除，虽然要找插入位置，但操作是比较操作，从这个角度考虑显然后者优于前者。

　　（3）基于环境的考虑（语言）

　　顺序表容易实现，任何高级语言中都有数组类型；链表的操作是基于指针的。相对来讲前者简单些，也用户考虑的一个因素。

　　总之，两种存储结构各有长短，选择哪一种由实际问题中的主要因素决定。通常“较稳定”的线性表，即主要操作是查找操作的线性表，适于选择顺序存储；而频繁做插入删除运算的（即动态性比较强）的线性表适宜选择链式存储。

3、问题域中有一数据序列（a1，a2，…，an），用顺序存储表示时，ai和ai+1的物理位置相邻吗？用链表表示时呢？

**顺序存储：相邻。链表：不相邻。**

**五、程序阅读题**

1、阅读程序，并输出结果

#include <iostream>

int main() {

using namespace std;

double \*p3=new double[3];

p3[0]=0.2; p3[1]=0.5; p3[2]=0.8;

cout<<”p3[1] is”<<p3[1]<<”.\n”;

p3=p3+1;

cout<<”Now p3[0] is”<<p3[0]<<” and”;

cout<<” p3[1] is”<<p3[1]<<” .\n”;

p3=p3-1;

delete [] p3;

return 0;

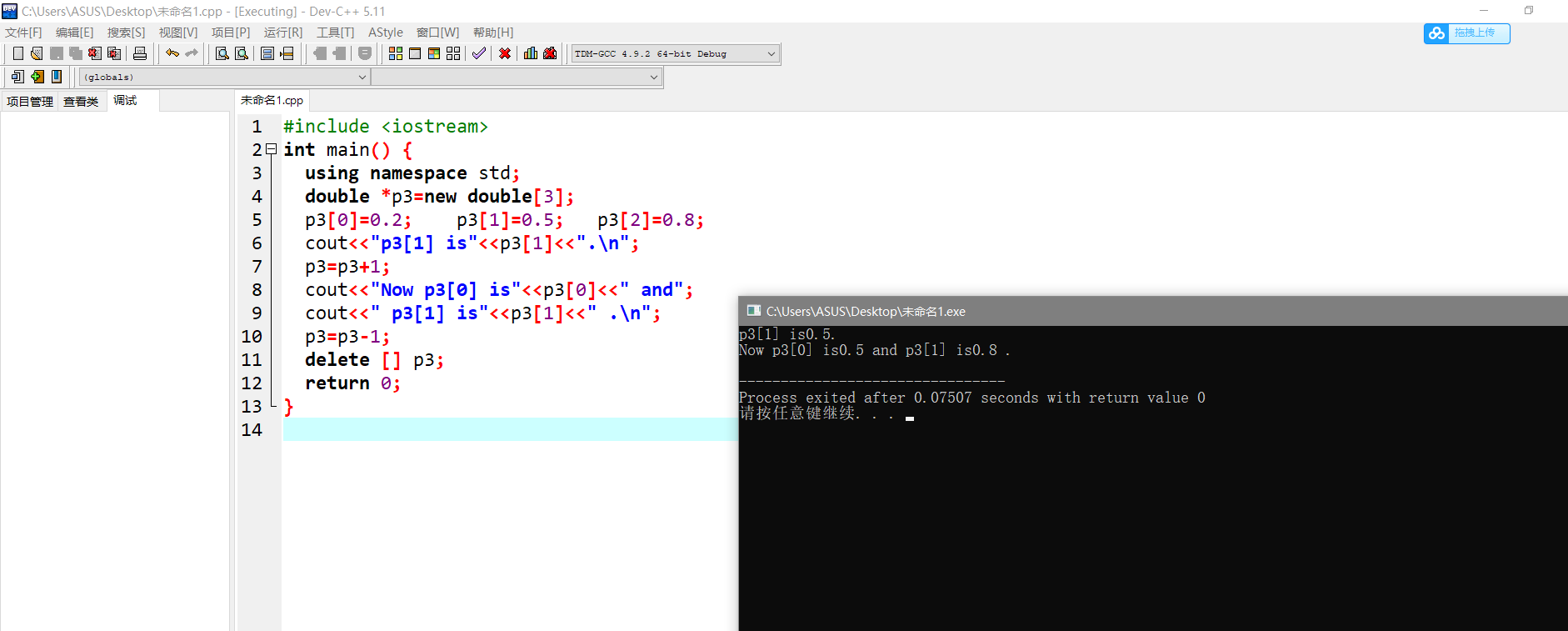
}

**预测输出结果：**

**p3[1] is0.5.**

**Now p3[0] is0.5 and p3[1] is0.8 .**

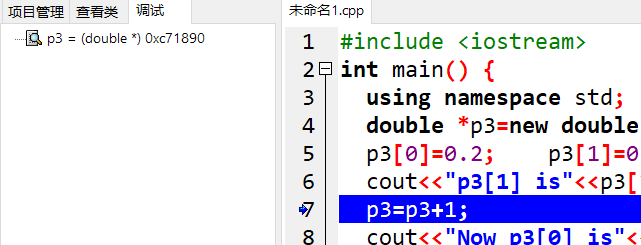
**结果验证：**



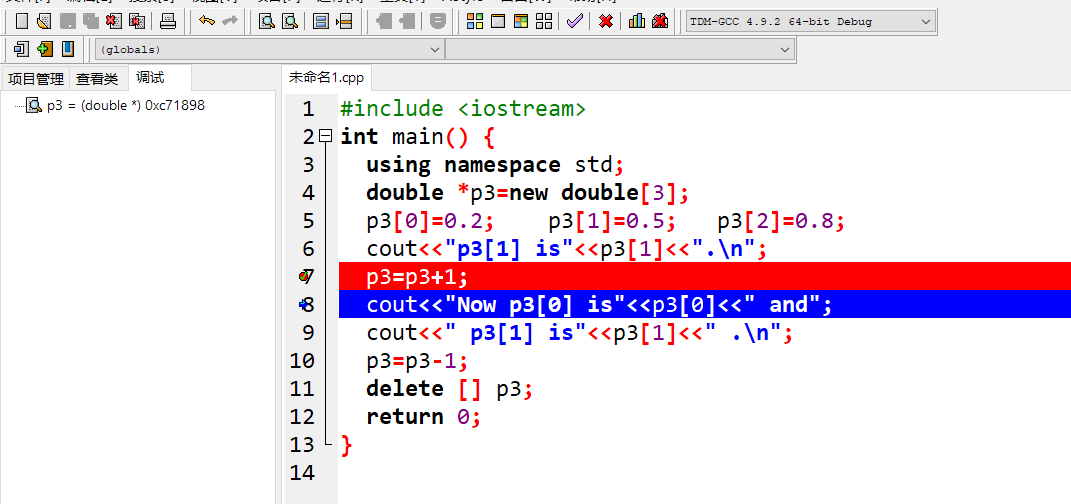
**结果正确。**

**程序调试：**

**断点：**



**在对指针+1之后：指针后移了一个double变量大小的长度。**

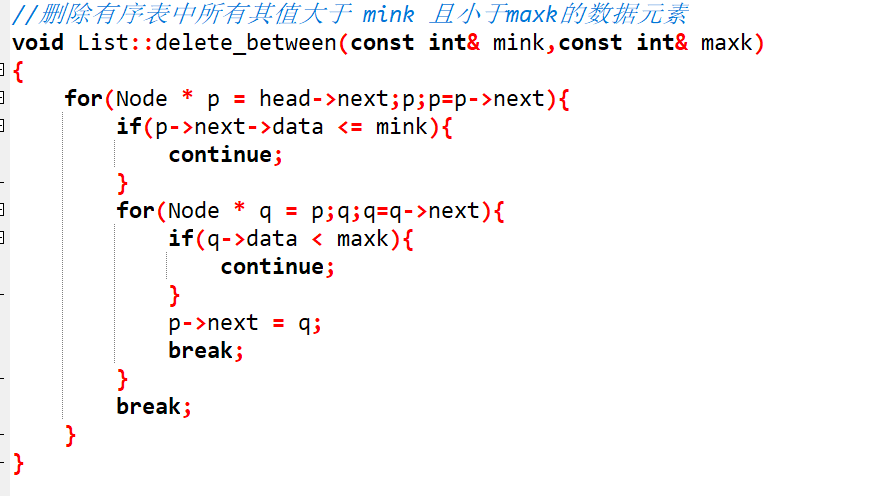


**六、程序题**

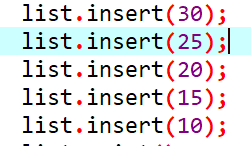
（一）链表应用题

1 已知线性表中的元素以值递增有序，并以单链表作存储结构。试写一程序，删除有序表中所有其值大于 mink 且小于maxk的数据元素。

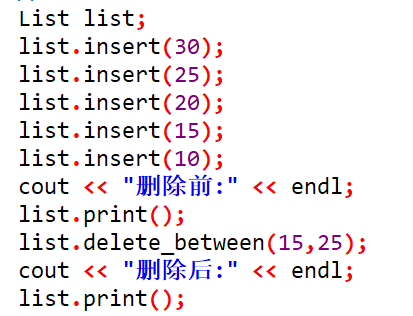
**代码实现：（链表的创建见cpp文件）**



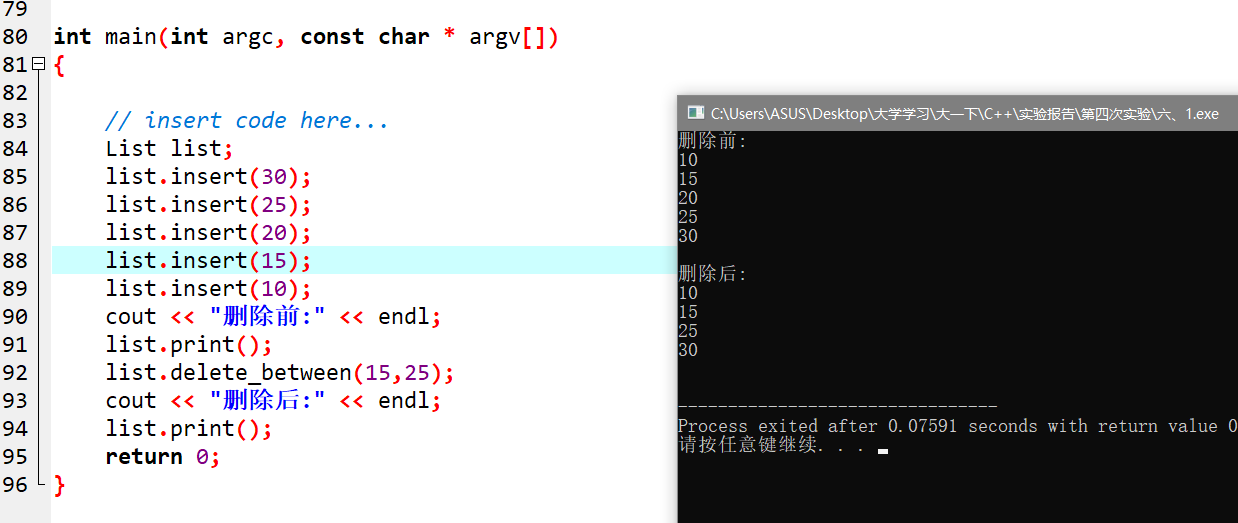
**链表输入：**



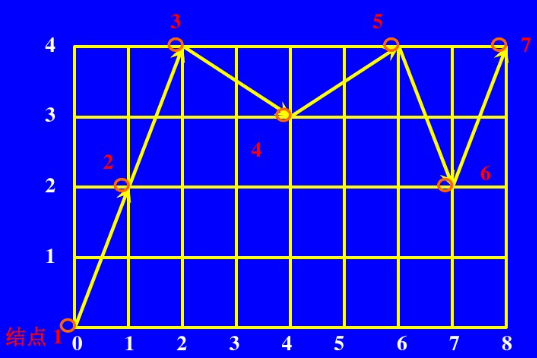
**链表输出：**



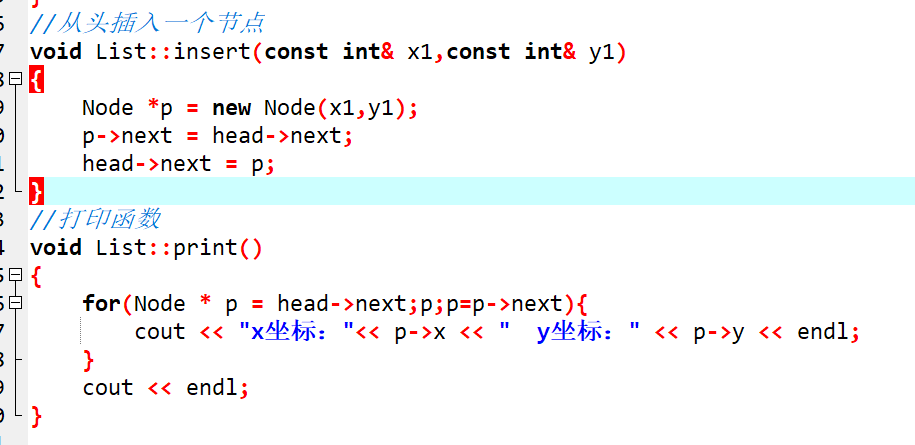
**最终结果：（只有20被删除了）**



2跳马。依下图将每一步跳马之后的位置（x,y）存放到一个“结点”中，再用“链子穿起来”，形成一条链。

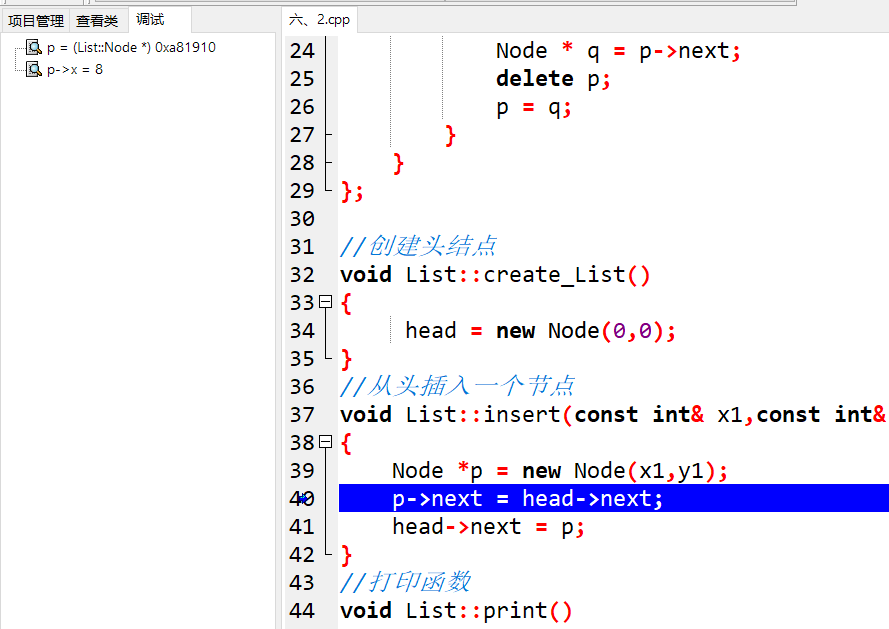


**存储实现：**



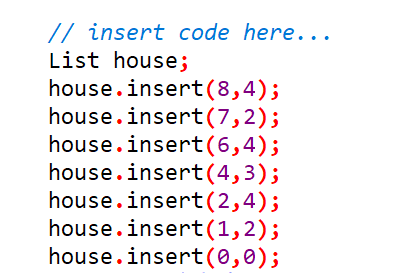
**原理：倒序存储各个坐标，每次从头插入节点**

**程序调试：**

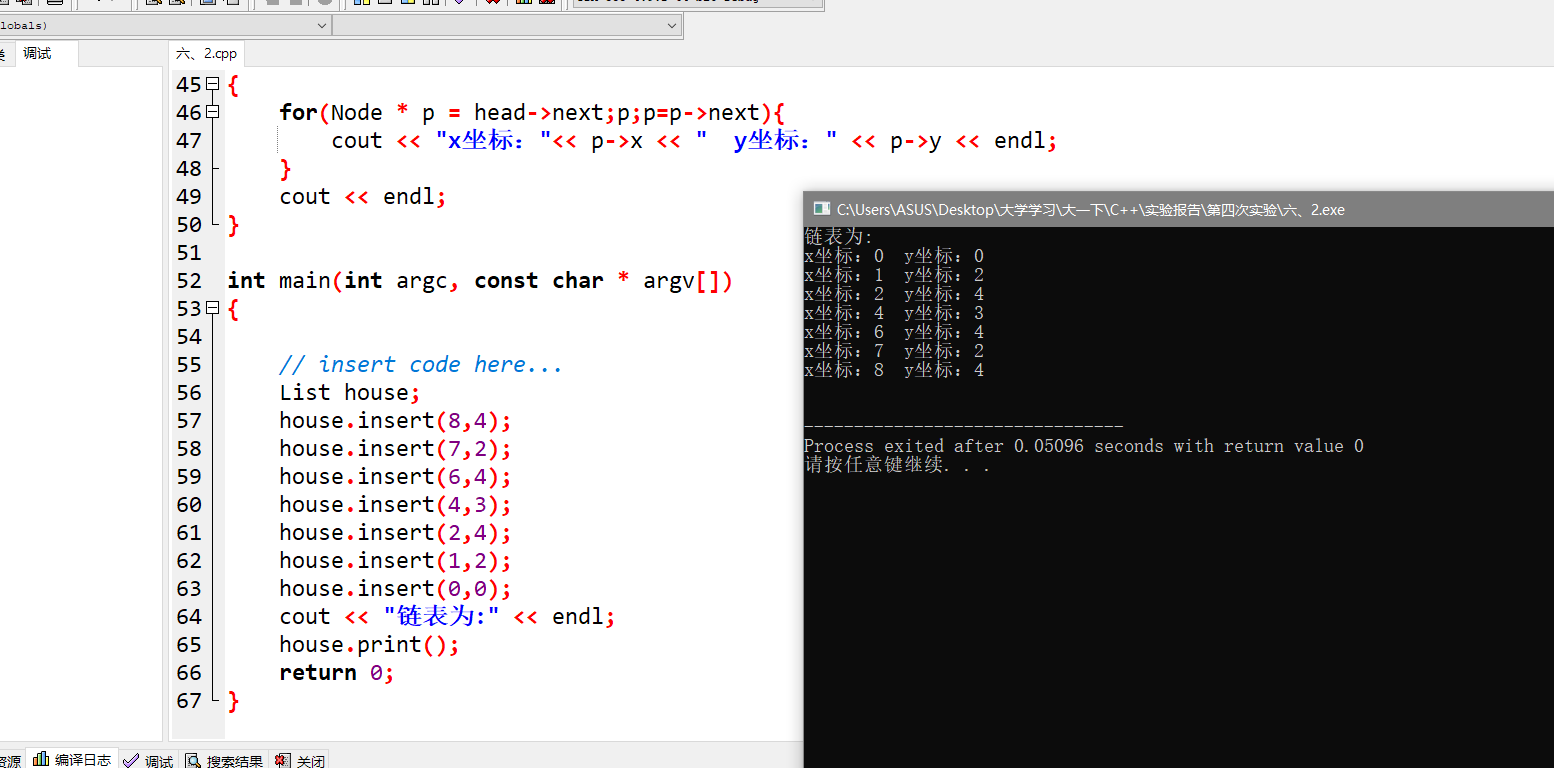


**（查看存储链表的过程）**

**链表输入：**



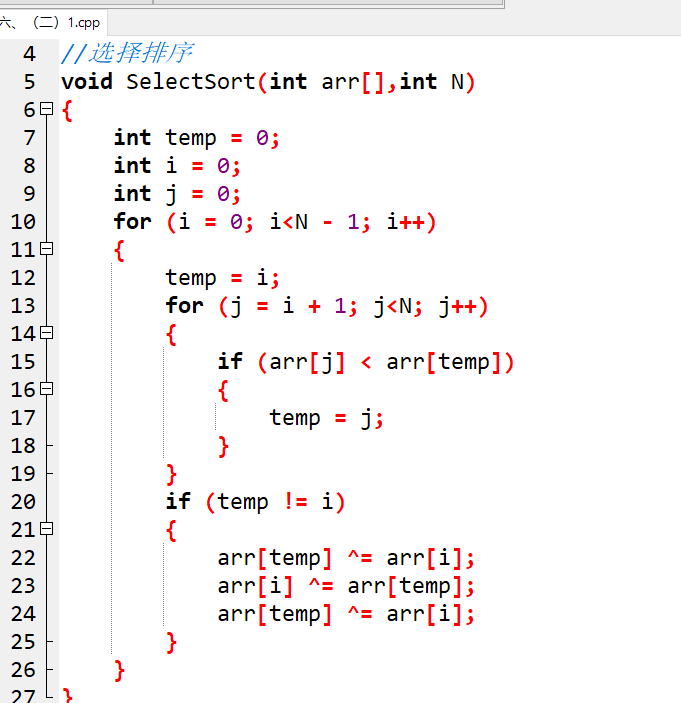
**程序结果：**



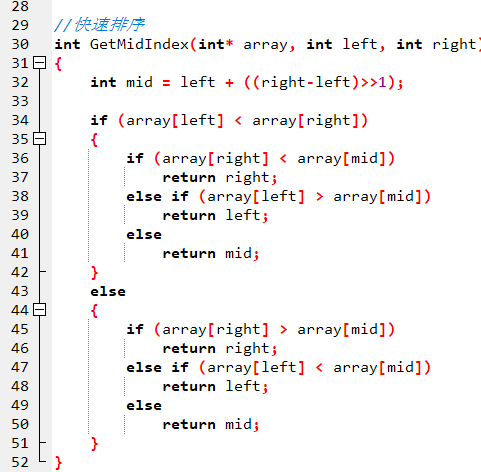
（二）动态数组的应用题

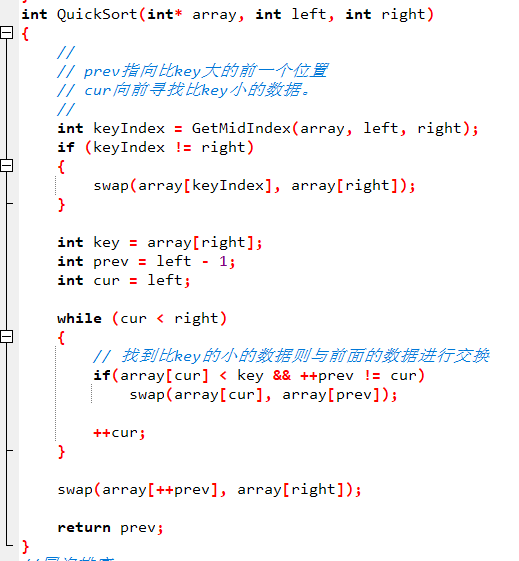
1、用new创建一个数组，从键盘上读取数组长度和数据，分别采用：选择排序、快速排序和冒泡排序，并输出数据。

选择排序：

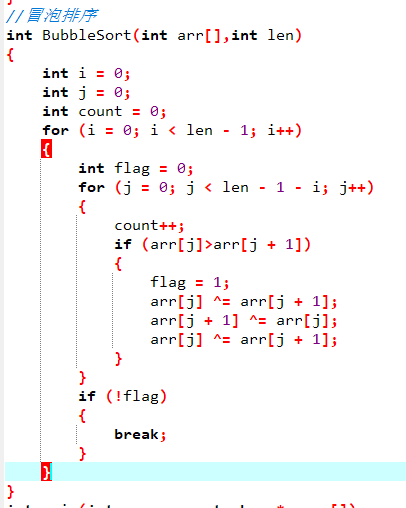


快速排序：

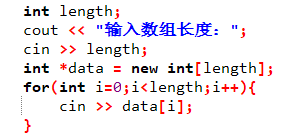




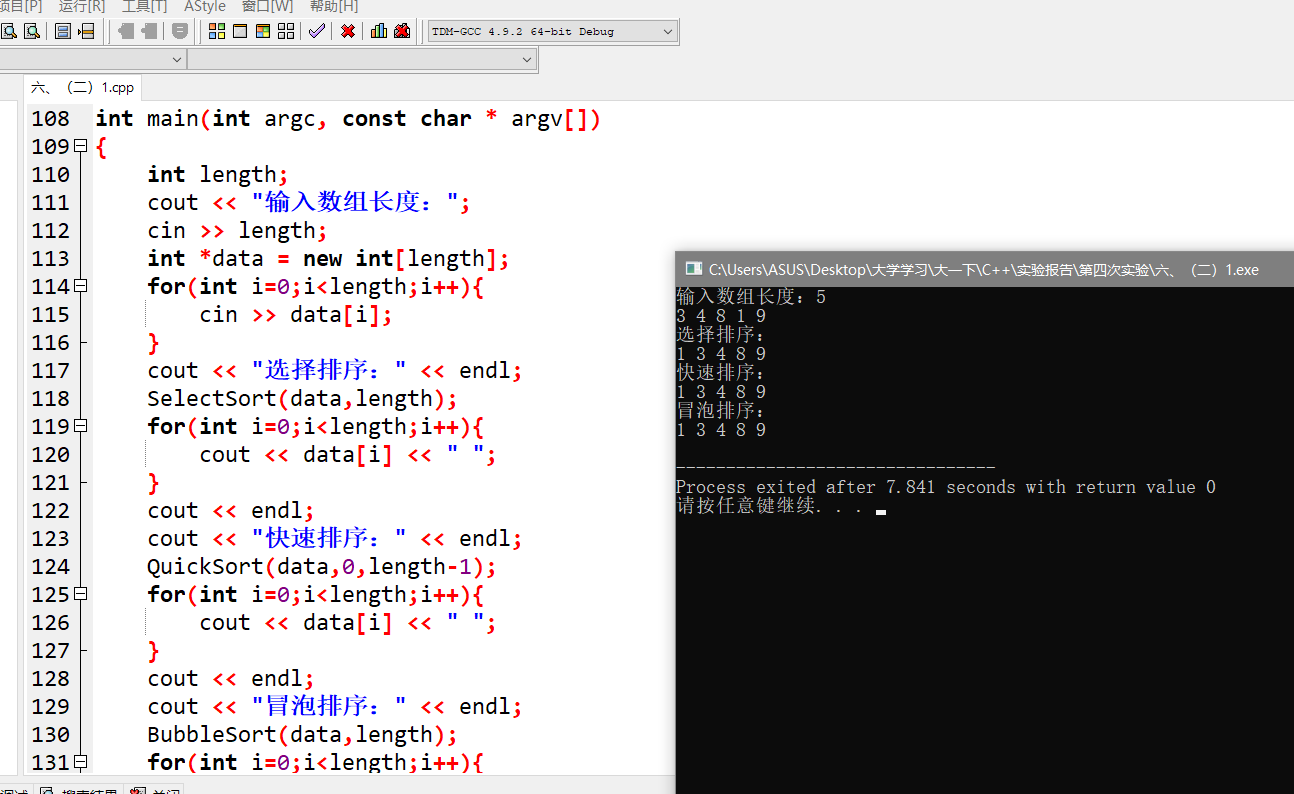
冒泡排序：



动态数组读入：

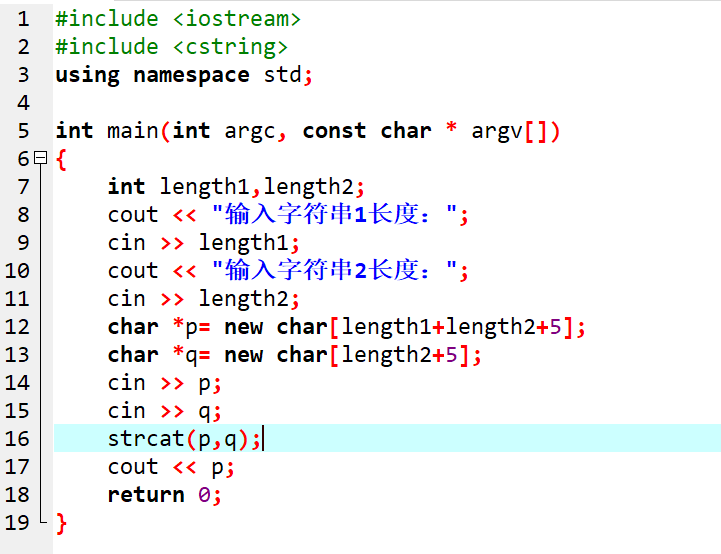


程序黑盒测试：

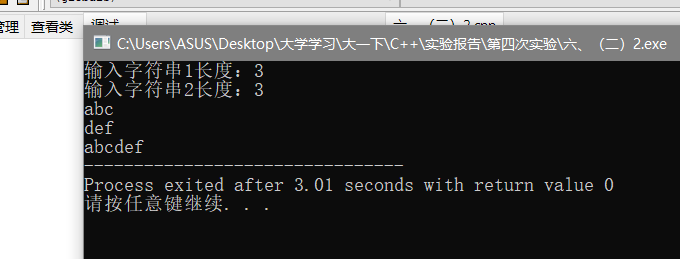


2、用动态数组分别存储两个字符串，并将两个字符串连接的结果存储至第一个动态数组中。【重点描述，若动态数组空间申请不够时，如何动态扩展新空间】

代码实现：



黑盒测试：



动态拓展：不够时可以用new新开辟一个两倍空间大小的数组，利用memcpy()函数拷贝数据。例如下图代码：

