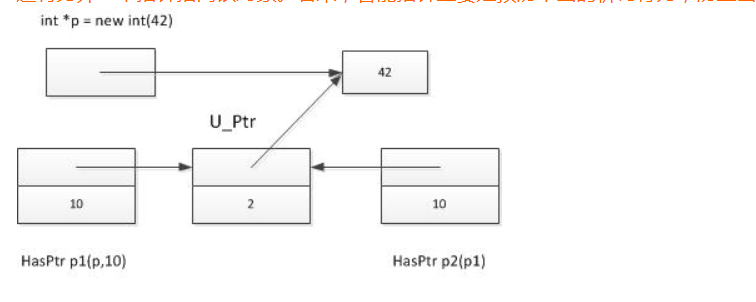
**四种智能指针**，即boost库中的auto\_ptr、scoped\_ptr、shared\_ptr、weak\_ptr。

智能指针(smart pointer)是存储指向动态分配（堆）对象指针的类，用于生存期控制，能够确保自动正确的销毁动态分配的对象，防止内存泄露。它的一种通用实现技术是使用引用计数(reference count)。智能指针类将一个计数器与类指向的对象相关联，引用计数跟踪该类有多少个对象共享同一指针。每次创建类的新对象时，初始化指针并将引用计数置为1；当对象作为另一对象的副本而创建时，拷贝构造函数拷贝指针并增加与之相应的引用计数；对一个对象进行赋值时，赋值操作符减少左操作数所指对象的引用计数（如果引用计数为减至0，则删除对象），并增加右操作数所指对象的引用计数；调用析构函数时，构造函数减少引用计数（如果引用计数减至0，则删除基础对象）。  
    智能指针就是模拟指针动作的类。所有的智能指针都会重载 -> 和 \* 操作符。智能指针还有许多其他功能，比较有用的是自动销毁。这主要是利用栈对象的有限作用域以及临时对象（有限作用域实现）析构函数释放内存。当然，智能指针还不止这些，还包括复制时可以修改源对象等。智能指针根据需求不同，设计也不同（写时复制，赋值即释放对象拥有权限、引用计数等，控制权转移等）。auto\_ptr 即是一种常见的智能指针。

智能指针模板创建：

1. **template** <**class** T>
2. **class** smartpointer
3. {
4. **private**:
5. T \*\_ptr;
6. **public**:
7. smartpointer(T \*p) : \_ptr(p)  //构造函数
8. {
9. }
10. T& operator \*()        //重载\*操作符
11. {
12. **return** \*\_ptr;
13. }
14. T\* operator ->()       //重载->操作符
15. {
16. **return** \_ptr;
17. }
18. ~smartpointer()        //析构函数
19. {
20. **delete** \_ptr;
21. }
22. };
23. // 定义仅由HasPtr类使用的U\_Ptr类，用于封装使用计数和相关指针
24. // 这个类的所有成员都是private，我们不希望普通用户使用U\_Ptr类，所以它没有任何public成员
25. // 将HasPtr类设置为友元，使其成员可以访问U\_Ptr的成员
26. **class** U\_Ptr
27. {
28. **friend** **class** HasPtr;
29. **int** \*ip;
30. **size\_t** use;
31. U\_Ptr(**int** \*p) : ip(p) , use(1)
32. {
33. cout << "U\_ptr constructor called !" << endl;
34. }
35. ~U\_Ptr()
36. {
37. **delete** ip;
38. cout << "U\_ptr distructor called !" << endl;
39. }
40. };

 HasPtr类需要一个析构函数来删除指针。但是，析构函数不能无条件的删除指针。”  
      条件就是引用计数。如果该对象被两个指针所指，那么删除其中一个指针，并不会调用该指针的析构函数，因为此时还有另外一个指针指向该对象。看来，智能指针主要是预防不当的析构行为，防止出现悬垂指针。

  
 如上图所示，HasPtr就是智能指针，U\_Ptr为计数器；里面有个变量use和指针ip，use记录了\*ip对象被多少个HasPtr对象所指。假设现在又两个HasPtr对象p1、p2指向了U\_Ptr，那么现在我delete  p1，use变量将自减1，  U\_Ptr不会析构，那么U\_Ptr指向的对象也不会析构，那么p2仍然指向了原来的对象，而不会变成一个悬空指针。当delete p2的时候，use变量将自减1，为0。此时，U\_Ptr对象进行析构，那么U\_Ptr指向的对象也进行析构，保证不会出现内存泄露。   
    包含指针的类需要特别注意复制控制，原因是复制指针时只复制指针中的地址，而不会复制指针指向的对象。  
    大多数C++类用三种方法之一管理指针成员  
    （1）不管指针成员。复制时只复制指针，不复制指针指向的对象。当其中一个指针把其指向的对象的空间释放后，其它指针都成了悬浮指针。这是一种极端  
    （2）当复制的时候，即复制指针，也复制指针指向的对象。这样可能造成空间的浪费。因为指针指向的对象的复制不一定是必要的。  
   （3） 第三种就是一种折中的方式。利用一个辅助类来管理指针的复制。原来的类中有一个指针指向辅助类，辅助类的数据成员是一个计数器和一个指针（指向原来的）（此为本次智能指针实现方式）。  
     其实，智能指针的引用计数类似于java的垃圾回收机制：java的垃圾的判定很简答，如果一个对象没有引用所指，那么该对象为垃圾。系统就可以回收了。