在二叉查找树类中增加三个成员函数：删除小于某个指定值的所有元素，删除大于某个指定值得所有元素，删除某一指定范围中的所有元素

【解】先看一下删除小于指定值 x 的操作 delLess。这个操作可以用递归的观点来看。如果根结点的值大于 x，那么小于 x 的这些值肯定在左子树上，于是对左子树递归调用 delLess。如果根结点值小于 x，那么根结点连同左子树都要被删除。于是将右子树替代整棵树，把根结点和左子树删掉。但原来的右子树上也可能有小于 x 的元素，于是对原来的右子树递归调用 delLess。这个过程可见代码清单 8-10

代码清单 8-10 删除小于某个指定值的所有元素

```
1.  template <class Type>
2.  void BinarySearchTree<Type>::delLess( const Type x )
3.   {  delLess(x, root); }
4.
5.  template <class Type>
6.  void BinarySearchTree<Type>::delLess( const Type x, BinaryNode *&t ){
7.    if ( t==NULL ) return;
8.    if ( t->data < x ) {    // 删除根结点及左子树
9.          BinaryNode *tmp = t;
10.         t = t->right;
11.         makeEmpty(tmp->left);
12.           delete tmp;
13.       delLess(x,t);
14.       }
15.    else delLess(x,t->left); // 在左子树上删除所有小于 x 的元素
16. }
```

删除大于 x 的所有元素的过程与 delLess 基本类似，具体见代码清单 8-11。

代码清单 8-11 删除大于 x 的所有元素

```
1.  template <class Type>
2.  void BinarySearchTree<Type>::delGreat( const Type x )
3.  {  delGreat(x, root);  }
4.
5.  template <class Type>
6.  void BinarySearchTree<Type>::delGreat( const Type &x, BinaryNode *&t )
7.  {  if ( t == NULL ) return;
8.     if ( t->data > x ) {              // 删除根结点及右子树
9.          BinaryNode *tmp = t;
10.         t = t->left;
11.           makeEmpty(tmp->right);
12.             delete tmp;
13.         delGreat(x,t);
14.       }
```

```
15.       else delGreat(x,t->right);  // 在右子树上删除所有大于 x 的元素
16.   }
```

删除某一范围的所有元素也是采用递归实现的，代码清单 8-12 给出了这一过程。私有的成员函数 delRange 删除以 t 为根的树上值为 x 和 y 之间的所有元素。具体的工作过程是：如果根结点小于 x，那么左子树和根结点上都不包含这一范围内的元素，但右子树上可能有。于是对右子树递归调用这个过程。如果根结点大于 y，表示根结点和右子树与删除无关，于是对左子树递归调用这个过程。如果这两个条件都不满足，也就是说，根结点正好是在 x 和 y 之间，必须删除。于是把根结点从这棵树上删去，再次对这棵树调用 delRange。

代码清单 8-12 删除某一范围的所有元素

```
1.    template <class Type>
2.    void BinarySearchTree<Type>::delRange( Type x, Type y )
3.     { if (x <= y) delRange(x, y, root ); }
4.
5.    template <class Type>
6.    void BinarySearchTree<Type>::delRange( Type x, Type y, BinaryNode *&t )
7.    {   if ( t == NULL ) return;
8.        if ( t->data < x ) delRange( x, y, t->right ); // 被删元素在右子树上
9.        else if ( t->data > y ) delRange( x, y, t->left ); // 被删元素在左子树
10.           else {                          // 根结点需要删除
11.               remove(t->data, t);
12.               delRange(x, y, t);
13.             }
14.   }
```