基于自制数据结构的高中学校管理系统

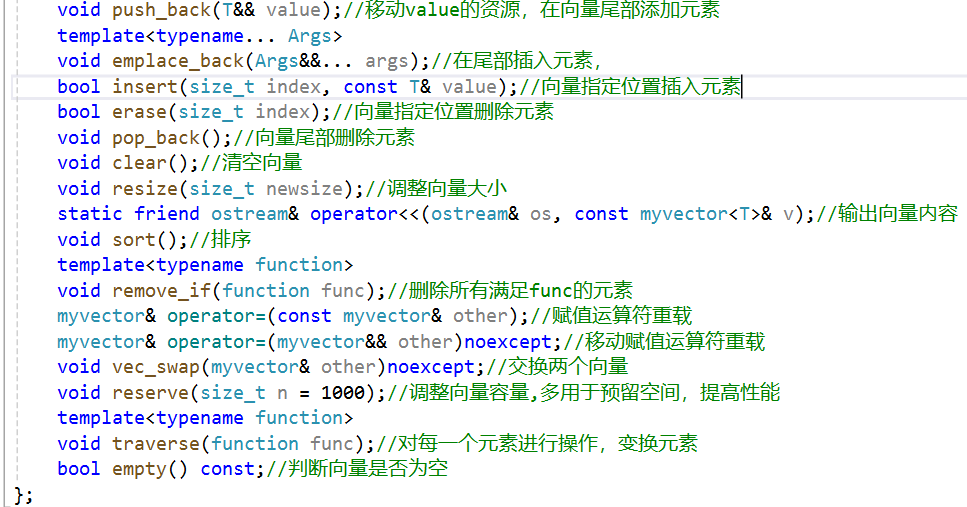
第一部分：自制数据结构

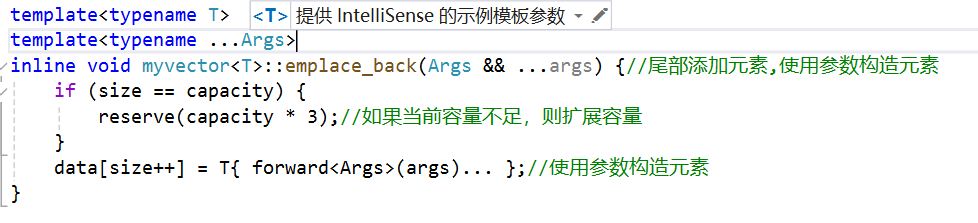
1.//仿照C++的vector实现一个简单的向量类

//由1个类构成，包含一个T\* data数组，一个size\_t size，一个size\_t capacity

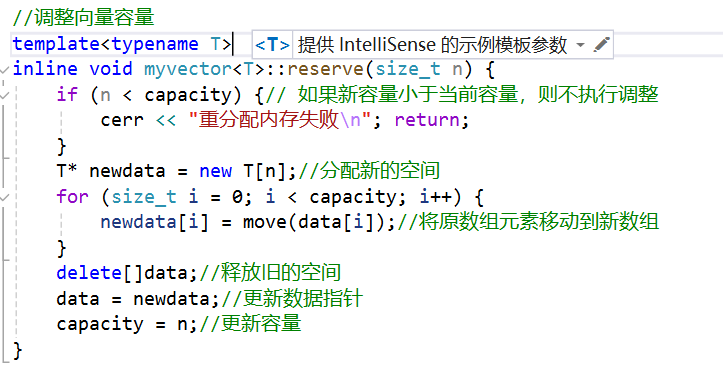
//大部分可以参考线性表，myvector可以视为可扩容的线性表；











2.//定义链表类

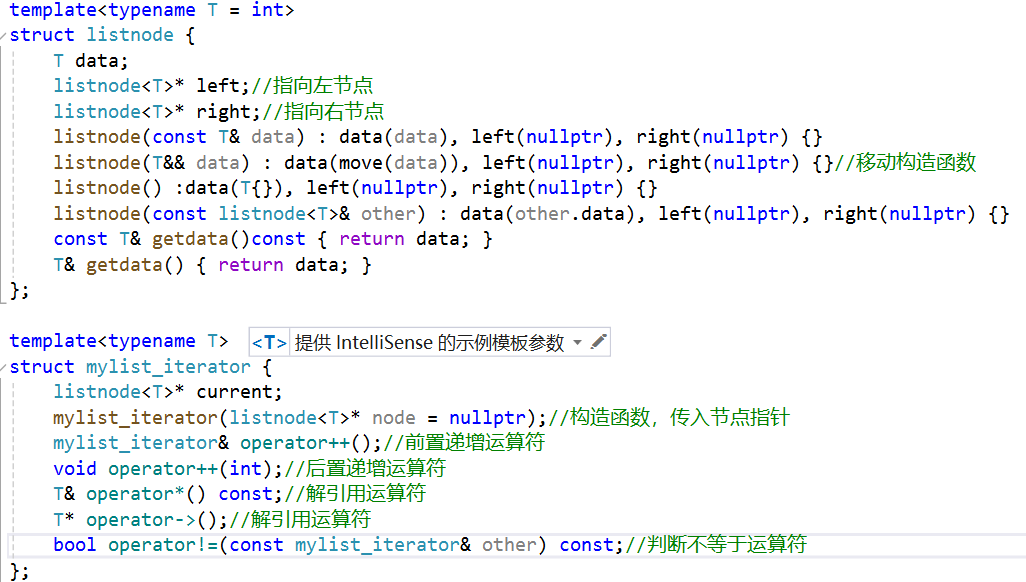
// 由3个类(结构体）构成：listnode，mylist\_iterator，mylist

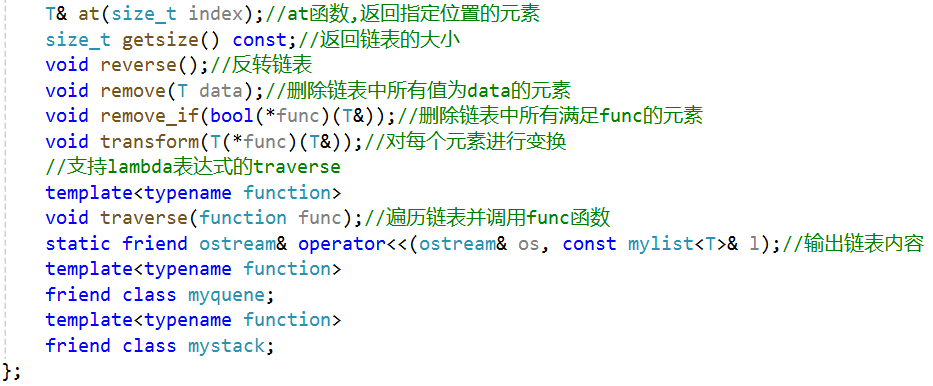
// listnode：链表节点，包含数据，左右指针

// mylist\_iterator：迭代器类，用于遍历链表

// mylist：链表类，包含头尾指针，大小

//包含头尾指针，大小，以及迭代器





3.//以下为hashmap实现

//由4个类（结构体）构成：

//1.mypair<K, V>：键值对类，包含两个成员变量first和second，分别表示键和值

//2.node<T>：节点类，包含两个成员变量data和next，分别表示键值对和下一个节点的指针

//3.myhash\_map<K, V>：哈希表类，包含两个成员变量buckets和size，分别表示散列表的桶数组和元素个数

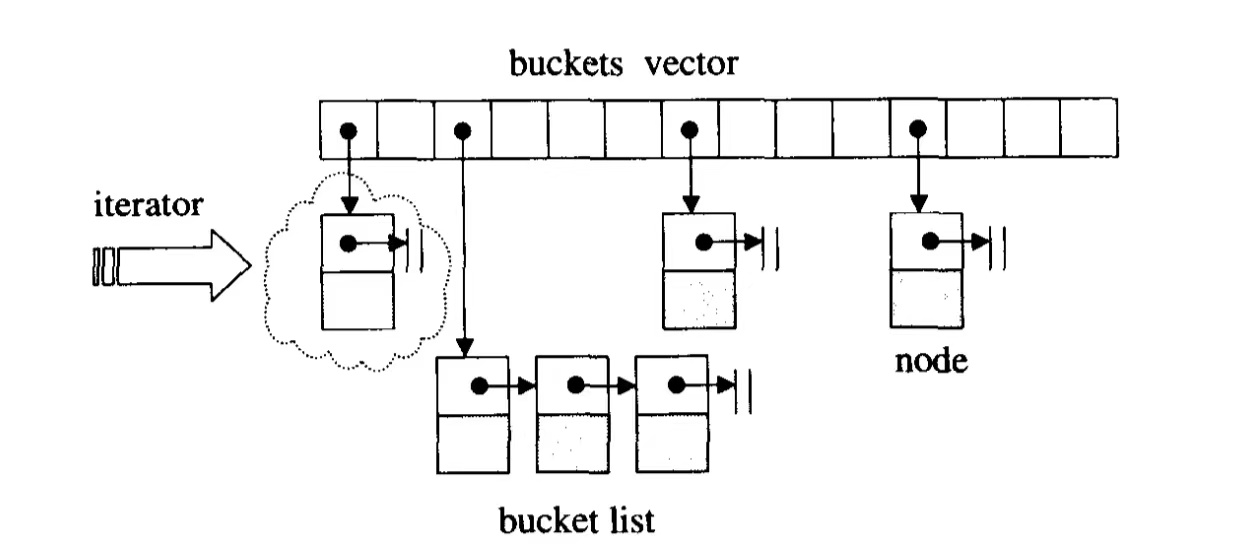
//4.myhashmap\_iterator<K, V>：迭代器类，包含四个成员变量table、bucket\_index、current、m\_capacity，

// 分别表示散列表的桶数组、

// 当前迭代的桶索引、

// 当前迭代器指向的节点、

// 散列表的容量

//myhash\_map内部结构：数组套链表(如下)

//myhash\_map的实现：

//1.hash函数：将键映射到数组的索引上，通过hash函数计算出索引值

//2.冲突解决：当两个键映射到同一个索引时，发生冲突，解决冲突的方法为链表法

//3.扩容：当元素个数超过散列表的最大容量时，需要扩容，扩容的方法为重新申请一块新的内存，将原有元素拷贝到新内存中，

// 然后将原有散列表的内存释放掉

//4.查找：通过hash函数计算出索引值，然后在桶数组中查找，如果找到则返回值，否则返回空值

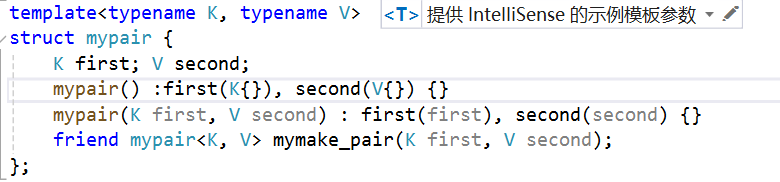
//5.插入：通过hash函数计算出索引值，然后在桶数组中查找，如果找到则更新值，否则插入新节点

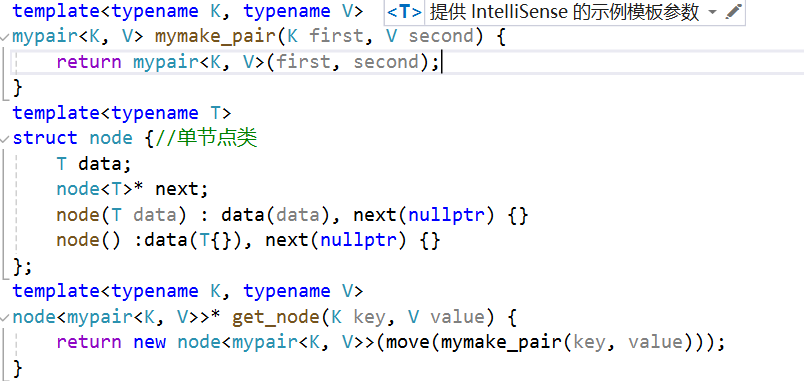
//6.删除：通过hash函数计算出索引值，然后在桶数组中查找，如果找到则删除节点，否则什么都不做

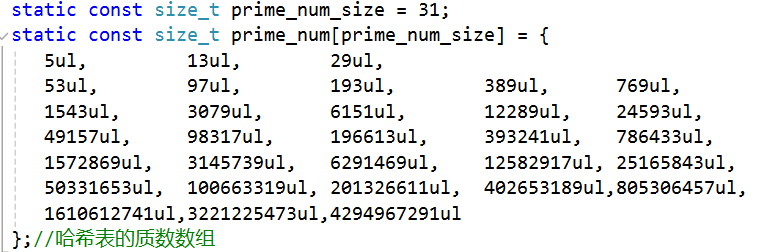
//7.遍历：遍历整个散列表，对每个节点(非空）调用函数

//8.哈希函数：通过hash函数计算出索引值;冲突越少，查找、插入、删除操作的效率越高

//9.重载[]运算符：通过键查找值，如果键不存在，则插入新值

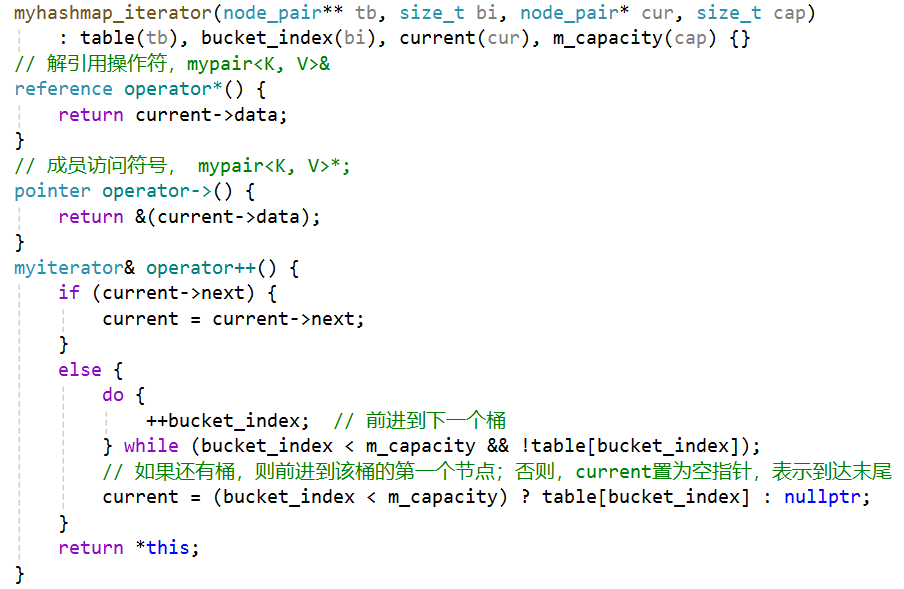
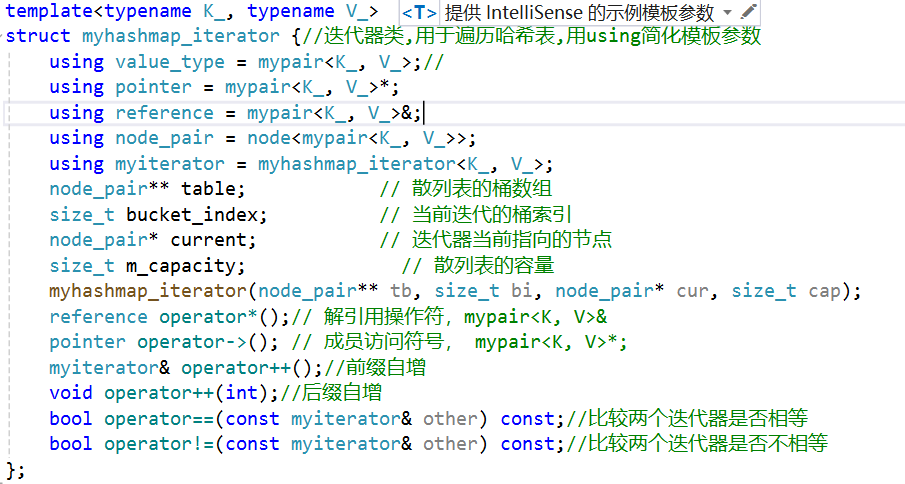


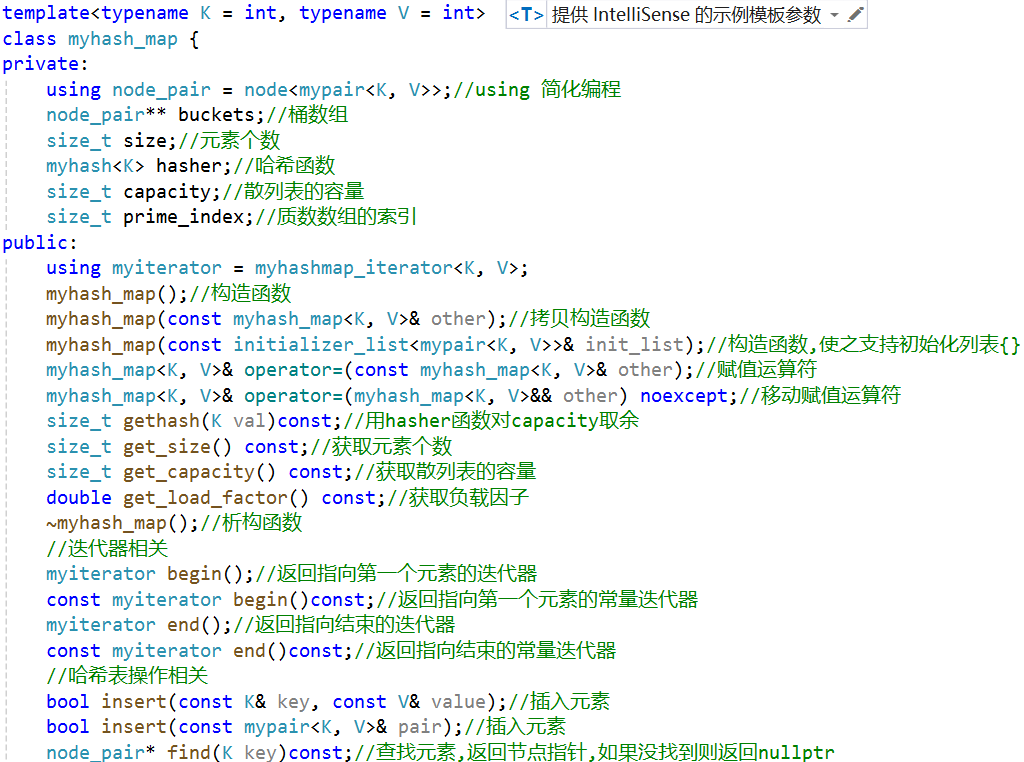


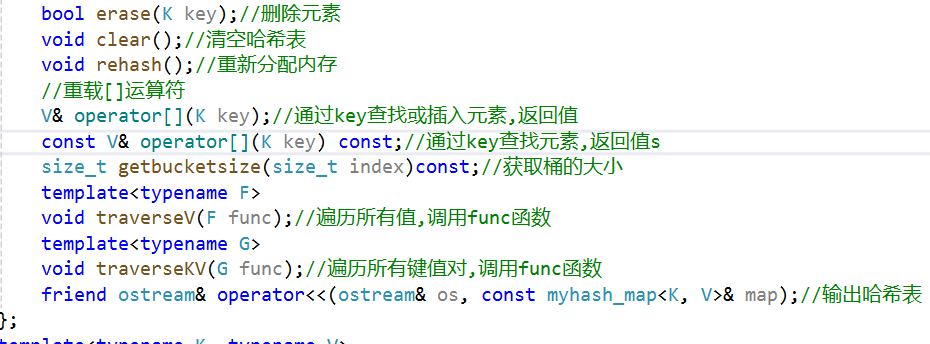


哈希函数：

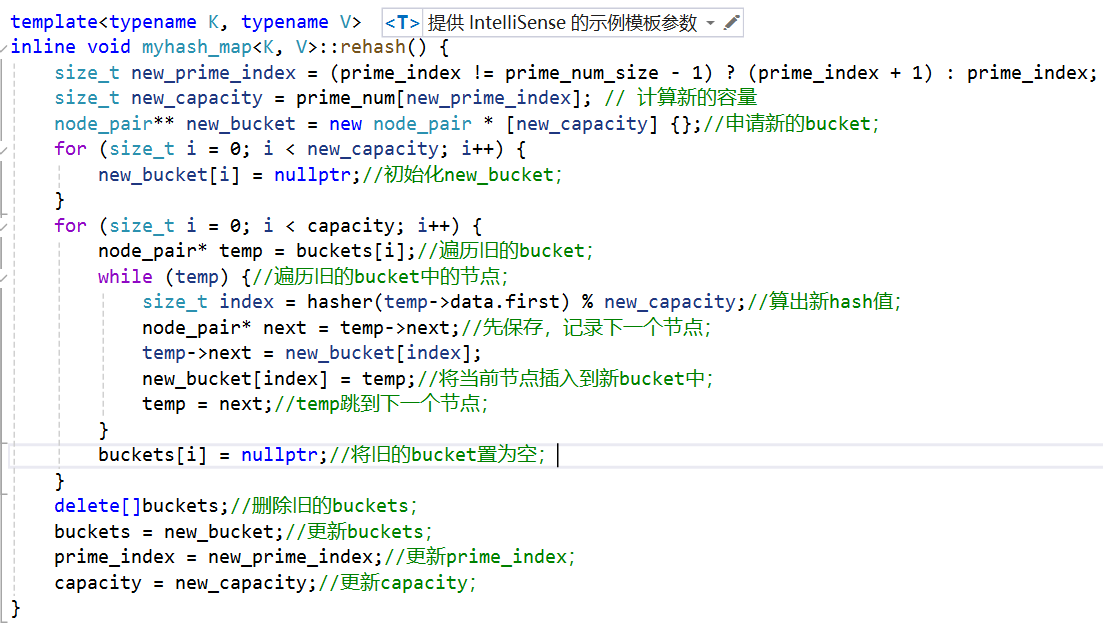




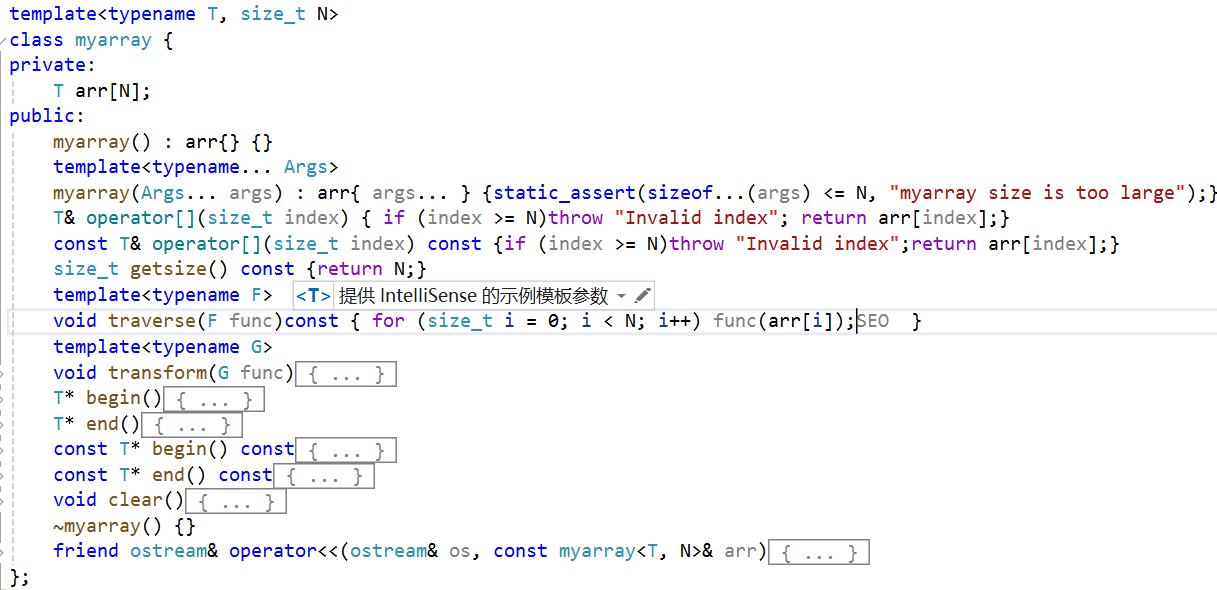


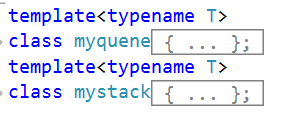


以下为重哈希(rehash)的算法，通过指针的搬移形成；



4.//myarray类，用于存放固定大小的数组



5.用mylist的友元实现的myquene和mystack类，可能有用；

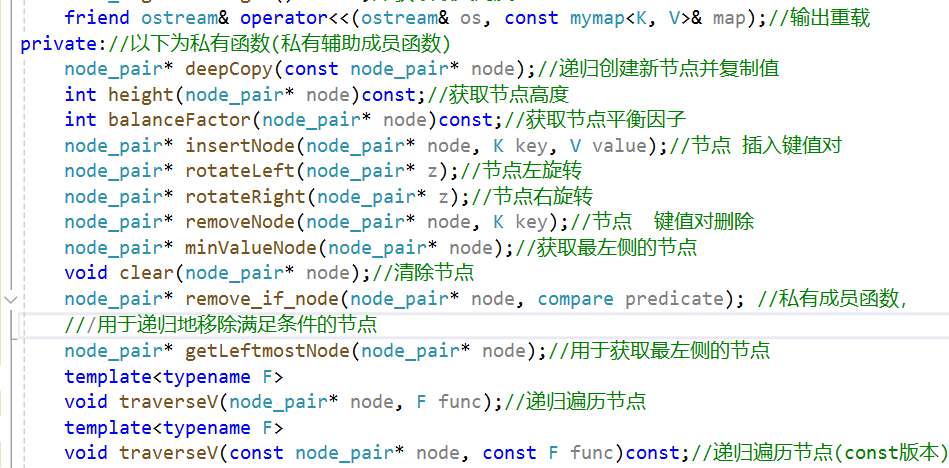
6.//用avl树实现mymap

//avl树的节点类

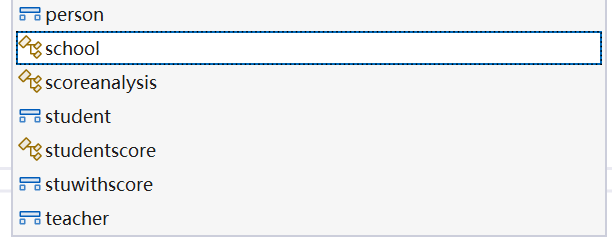


//以下为mymap类





第二部分：目前我们学校管理系统的大概框架,后续可能添加新的部分；(其中为结构体，为类)



//person类，所有人员的基类;

//teacher类，教师类，继承自person类;

//student类，学生类，继承自person类;

//school类，学校类，包含学生和老师等，用myvector和myhash\_map实现的聚合类;

//stuwithscore类，用于存放学生信息和成绩的结构体;

//studentscore类，用于存放学生的成绩的类;

//scoreanalysis类，用于分析学生的成绩的类;

