```c
/* RBTree.h */
#ifndef _RBTREE_H
#define _RBTREE_H

//0--代表比较相同，>0--代表dataAddr<keyAddr，<0--代表dataAddr>keyAddr
typedef int  RBTreeCmp(const void *keyAddr, const void *dataAddr);
typedef void RBTreeFree(void *);
typedef void RBTreeTraverseOp(void *);

typedef enum {RB_RED, RB_BLACK} RBCOLOR;

typedef struct red_black_node
{
    struct red_black_node *parent;
    struct red_black_node *lc;
    struct red_black_node *rc;
    int height; //黑高度
    RBCOLOR color;
    char key[0];
}RBTREENODE;

typedef struct
{
    RBTREENODE *root;
    RBTREENODE *hot; //"命中"节点的父亲
    int size;
    int keySize;
    RBTreeCmp *cmpFn;
    RBTreeFree *freeFn;
}RBTREE;

//RBTree初始化
void RBTreeNew(RBTREE *rbTree, int keySize, RBTreeCmp *cmpFn, RBTreeFree *freeFn);
//RBTree销毁
void RBTreeDispose(RBTREE *rbTree);
//RBTree判空
int RBTreeEmpty(RBTREE *rbTree);
//RBTree规模
int RBTreeSize(RBTREE *rbTree);
//RBTree树高度
int RBTreeHeight(RBTREE *rbTree);
//RBTree中序遍历（非递归）
void RBTreeTravIn(RBTREE *rbTree, RBTreeTraverseOp *traverseOpFn);
//RBTree中序遍历（递归）
void RBTreeTravInRec(RBTREE *rbTree, RBTreeTraverseOp *traverseOpFn);
//RBTree中查找关键码所在节点，hot指向当前节点的父节点
RBTREENODE *RBTreeSearch(RBTREE *rbTree, const void *e);
//RBTree中插入关键码
RBTREENODE *RBTreeInsert(RBTREE *rbTree, const void *e);
//RBTree中删除关键码所在节点，返回值：0成功，!0失败
int RBTreeRemove(RBTREE *rbTree, void *e);
//RBTree中删除关键码所在节点（无需深度删除关键码），返回值：0成功，!0失败
int RBTreeRemoveU(RBTREE *rbTree, void *e);
#endif
```