```
/
    ********************************************************************************
    **************

 ******************                        File: avl.h
     ******************


     ********************************************************************************
     **************/
#ifndef __AVL_H
#define __AVL_H

/*
 File: avl.h
 Interface definition of the AVL tree data structure.
 */

struct AVLTree;
struct AVLTreeIterator;

/* In C, pointer to "function" that takes argument types and returning type  is
    specified as:
 type (*function)(argtypes);
 */


typedef int (*comparator)(void * , void * );
typedef void (*printer)(void *);

/* Initialize binary search tree structure. */
void initAVLTree(struct AVLTree *tree);

/* Alocate and initialize search tree structure. */
struct AVLTree *newAVLTree();

/* Deallocate nodes in BST. */
void clearAVLTree(struct AVLTree *tree);

/* Deallocate nodes in AVL and AVL structure. */
void freeAVLTree(struct AVLTree *tree);

/*-- AVL Bag interface --*/
int  isEmptyAVLTree(struct AVLTree *tree);
int     sizeAVLTree(struct AVLTree *tree);

void     addAVLTree(struct AVLTree *tree, void *val, comparator compare);
int containsAVLTree(struct AVLTree *tree, void *val, comparator compare);
void  removeAVLTree(struct AVLTree *tree, void *val, comparator compare);

/* Utility function to print a tree */
void printTree(struct AVLTree *tree, printer printVal);

/* Iterator Interface */
struct AVLTreeIterator *createAVLTreeItr(struct AVLTree *tree);
void initAVLTreeItr( struct AVLTree *tree, struct AVLTreeIterator *itr);
```

```
void *nextAVLTreeItr(struct AVLTreeIterator *itr);
int hasNextAVLTreeItr(struct AVLTreeIterator *itr);
# endif



/
    ********************************************************************************
    **************

 ******************                          File: avl.c
     ******************


     ********************************************************************************
     **************/

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "avl.h"

struct AVLNode {
   void * val;
   struct AVLNode *left; /* Left child. */
   struct AVLNode *rght; /* Right child. */
   int hght;
};

struct AVLTree { /* Height-Balanced Binary Search Tree. */
   struct AVLNode *root;
   int cnt;
};

/*Iterator Structs*/
#ifdef ITERATOR
struct AVLTreeIterator {
    struct DynArr *stk;
    struct AVLTree *tree;
};
#endif

void initAVL(struct AVLTree *tree)
{
   tree->root = 0;
   tree->cnt = 0;

}

struct AVLTree *newAVLTree(){
   struct AVLTree *tree;

   tree = (struct AVLTree *)malloc(sizeof(struct AVLTree));
   assert(tree != 0);

   initAVL(tree);
```

```
   return tree;

}

/* Prototypes for private functions */
int _height(struct AVLNode *cur);
void _setHeight(struct AVLNode *cur);
int _bf(struct AVLNode *cur);

struct AVLNode *_balance(struct AVLNode *cur);
struct AVLNode *_rotateLeft(struct AVLNode *cur);
struct AVLNode *_rotateRight(struct AVLNode *cur);

/*------------------------------------------------------------------------
    */
void _freeAVLTree(struct AVLNode *node) {
  if (node != 0) {
    _freeAVLTree(node->left);
    _freeAVLTree(node->rght);
     free(node);
  }
}
/*------------------------------------------------------------------------
    */
void clearAVLTree(struct AVLTree *tree) {
  _freeAVLTree(tree->root);
  tree->root = 0;
  tree->cnt  = 0;

}
/*------------------------------------------------------------------------
    */
void freeAVLTree(struct AVLTree *tree) {
  clearAVLTree(tree);
  free(tree);
}
/*------------------------------------------------------------------------
    */
int isEmptyAVLTree(struct AVLTree *tree) { return tree->cnt == 0 ; }


 int sizeAVLTree(struct AVLTree *tree) { return tree->cnt; }



struct AVLNode *_addNode(struct AVLNode *cur, void * val, comparator compare) {
struct AVLNode *newNode;
   if(cur == 0) /* Base Case */
   {
      /* Create a new one and return it */
      newNode = (struct AVLNode *)malloc(sizeof(struct AVLNode));
      assert(newNode != 0);
      newNode->left = newNode->rght = 0;
      newNode->val = val;
      newNode->hght = 0; /* or SetHeight on new Node */
      return newNode;  /* No need to balance here! */
```

```
   }
   else { /* recursive case */

      if((*compare)(val, cur->val) < 0)
         cur->left = _addNode(cur->left, val, compare); /* functional approach,
             rebuild subtree */
      else cur->rght = _addNode(cur->rght, val, compare);
   }
   /* must balance the tree on way up from the recursion */
   return _balance(cur);  /* return the 'rebuilt' tree as come back from
       recursion */
}

void addAVLTree(struct AVLTree *tree, void * val, comparator compare) {
   tree->root = _addNode(tree->root, val, compare); /* call the recursive
       helper function */
   tree->cnt++;
}
// Iterative version of contains
int containsAVLTree(struct AVLTree *tree, void * val, comparator compare) {
   struct AVLNode *cur;
   cur = tree->root;

   while(cur != 0)
   {
      if((*compare)(cur->val, val) == 0)
         return 1;
      else if((*compare)(val, cur->val) < 0)
               cur = cur->left;
      else cur = cur->rght;
   }
   return 0;
}



void * _leftMost(struct AVLNode *cur) {
   while(cur->left != 0)
   {
      cur = cur->left;
   }
    return cur->val;
}

struct AVLNode *_removeLeftmost(struct AVLNode *cur) {
struct AVLNode *temp;

   if(cur->left != 0)
   {
      cur->left = _removeLeftmost(cur->left);
      return cur;                              /*This  balance can be
          removed ...had _balance(cur)*/
   }

   temp = cur->rght;
   free(cur);
```

```
   return temp;
}

struct AVLNode *_removeNode(struct AVLNode *cur, void * val, comparator compare
    ) {
   struct AVLNode *temp;

   if((*compare)(val, cur->val) == 0)
   {
      if(cur->rght != 0)
      {
         cur->val =  _leftMost(cur->rght);
         cur->rght =_removeLeftmost(cur->rght);
      }
      else {
         temp = cur->left;
         free(cur);
         return temp;  /* Note that height could not have changed on temp */
      }
   }

   else if((*compare)(val, cur->val) < 0)
          cur->left  = _removeNode(cur->left, val, compare);
   else  cur->rght = _removeNode(cur->rght, val, compare);

   return _balance(cur);
}

void removeAVLTree(struct AVLTree *tree, void * val, comparator compare) {
   if (containsAVLTree(tree, val, compare)) {
      tree->root = _removeNode(tree->root, val, compare);
      tree->cnt--;
   }
}


/* utility function to determine the height of a node */
int _height(struct AVLNode *cur) {
   if (cur == 0) return -1;
   return cur->hght;
}

/* utility function to set the height of a node */
void _setHeight(struct AVLNode *cur) {
   int lh = _height(cur->left);
   int rh = _height(cur->rght);
   if (lh < rh) cur->hght = rh + 1;
      else cur->hght = lh + 1;
}

/* utility function to compute the balance factor of a node
 * computed as rght - left
 * */
int _bf(struct AVLNode *cur) {
   return _height(cur->rght) - _height(cur->left);
}
```

```
/* utility function to balance a node*/

struct AVLNode *_balance(struct AVLNode *cur) {
    /*compute the balance factor for the node*/
   int cbf = _bf(cur);
   if (cbf < -1) { /* cur is heavy on the left */
       if (_bf(cur->left) > 0) /* Check for double rotation. ie. is the
           leftChild heavy on the right */
           cur->left = _rotateLeft(cur->left);  /*yes, left child was heavy on
               right, so rotate child left first */
       return _rotateRight(cur); /* Rotate cur to the right*/
   }else if (cbf > 1) {
            if (_bf(cur->rght) < 0)
                cur->rght = _rotateRight(cur->rght);
            return _rotateLeft(cur);
            }
           else {  //If no rotations necessary, still have to set height for
               current!
                   _setHeight(cur);
                    return cur;
               }
}


struct AVLNode *_rotateLeft(struct AVLNode *cur) {

   struct AVLNode *newTop;

   newTop = cur->rght;
   cur->rght = newTop->left;
   newTop->left = cur;
   /* reset the heights for all nodes that have changed heights*/
   /* Note that subtrees under it are all ok...because we've worked from the
       bottom up */
   _setHeight(cur);
   _setHeight(newTop);
   return newTop;

}

struct AVLNode *_rotateRight(struct AVLNode *cur){
  struct AVLNode *newTop;

   newTop = cur->left;
   cur->left = newTop->rght;
   newTop->rght = cur;

   _setHeight(cur);
   _setHeight(newTop);
   return newTop;
}

/*-------------------------------------------------------------------------
    */
```

```
void _printTree(struct AVLNode *cur, printer printVal) {
    if (cur == 0) return;

    printf("(");
    _printTree(cur->left, printVal);
    /* Note that you must change this to work for your particular 'data'
        pointer */

    (*printVal)(cur->val);printf("[%d]",cur->hght);

    _printTree(cur->rght, printVal);
    printf(")");
}
/*-------------------------------------------------------------------------
    */

void printTree(struct AVLTree *tree, printer printVal){
    _printTree(tree->root, printVal);
}


#ifdef ITERATOR

struct AVLTreeIterator *createAVLTreeItr(struct AVLTree *tree)
{
    struct AVLTreeIterator *itr = malloc(sizeof(struct AVLTreeIterator));
    assert(itr != 0);

    initAVLTreeItr(tree, itr);
    return itr;
}
void initAVLTreeItr(struct AVLTree *tree, struct AVLTreeIterator *itr)
{
    itr->tree = tree;
    itr->stk = createDynArr(10);
}
void *nextAVLTreeItr(struct AVLTreeIterator *itr)
{
    return ((struct AVLNode *)(topDynArr(itr->stk)))->val;
}

int hasNextAVLTreeItr(struct AVLTreeIterator *itr){
    struct AVLNode *n;

    if (isEmptyDynArr(itr->stk))
    {
        _slideLeft(itr->tree->root, itr);
    }else {
        n = topDynArr(itr->stk);
        popDynArr(itr->stk);
        _slideLeft(n->rght, itr);
    }
    if(!isEmptyDynArr(itr->stk))
        return 1;
    else return 0;
}
```

```
void _slideLeft (struct AVLNode *cur, struct AVLTreeIterator *itr){
    while(cur!= 0)
    {
        pushDynArr(itr->stk, cur);
        cur = cur->left;
    }
}

#endif
```

```
/
    ********************************************************************************
    **************

 ******************                    File: structs.h
     ******************


     ********************************************************************************
     **************/
#ifndef STRUCTS_H
#define STRUCTS_H
/*
 *  structs.h
 *
 *  Created by Ron Metoyer on 11/5/10.
 *  Copyright 2010 Oregon State Univ. All rights reserved.
 *
 */

struct data {
    int number;
    char *name;
};
#endif
```

```
/
    ********************************************************************************
    **************

 ******************                    File: compare.c
     ******************


     ********************************************************************************
     **************/
#include <stdio.h>
#include "assert.h"
#include "structs.h"

/*-------------------------------------------------------------------------
 very similar to the compareTo method in java or the strcmp function in c. it
 returns an integer to tell you if the left value is greater than, less than,
     or
 equal to the right value. you are comparing the number variable, letter is not
 used in the comparison.

 if left < right return -1
 if left > right return 1
 if left = right return 0
 */
int compare(void *left, void *right)
{
    struct data *ml,*mr;
    /* Casts! */
    ml = (struct data *)left;
    mr = (struct data *) right;
    /* Function to compare to 'data' types ...currently a number and name */

    if(ml->number < mr->number) return -1;
    if(ml->number > mr->number) return 1;

    return 0;
}

void printVal (void *val)
{
    struct data * myVal = (struct data *)val;
    printf("%d ", myVal->number);
}
```

```
/
    ********************************************************************************
    **************

 ********************                    File: main.c
     ********************


     ********************************************************************************
     **************/

#include<stdio.h>
#include<stdlib.h>
#include "avl.h"
#include "structs.h"

/* prototype for compare and printVal which are both in compare.c*/
int compare(void *left, void *right);
void printVal(void *node);

/* Example main file to begin exercising your tree */


int main(int argc, char *argv[])
{
    struct AVLTree *tree    = newAVLTree();

    /*Create value of the type of data that you want to store*/
    struct data myData1, myData2, myData3, myData4, myData5, myData6, myData7,
        myData8;

    //  myData1.number = 5;
    //  myData1.name = "rooty";
    //  myData2.number = 1;
    //  myData2.name = "lefty";
    //  myData3.number = 10;
    //  myData3.name = "righty";
    //  myData4.number = 30;
    //  myData4.name = "righty";
    //  myData5.number = 20;
    //  myData5.name  = "lefty";


    //      myData1.number = 30;
    //      myData1.name = "rooty";
    //      myData2.number = 20;
    //      myData2.name = "second";
    //      myData3.number = 50;
```

```
//      myData3.name = "third";
//      myData4.number = 40;
//      myData4.name = "fourth";
//      myData5.number = 60;
//      myData5.name  = "fifth";
//      myData6.number = 70;
//      myData6.name  = "sixth";


//          myData1.number = 50;
//          myData1.name = "rooty";
//          myData2.number = 22;
//          myData2.name = "second";
//          myData3.number = 80;
//          myData3.name = "third";
//          myData4.number = 70;
//          myData4.name = "fourth";
//          myData5.number = 75;
//          myData5.name  = "fifth";
//          myData6.number = 73;
//          myData6.name  = "sixth";



            myData1.number = 75;
            myData1.name = "rooty";
            myData2.number = 70;
            myData2.name = "second";
            myData3.number = 100;
            myData3.name = "third";
            myData4.number = 60;
            myData4.name = "fourth";
            myData5.number = 80;
            myData5.name  = "fifth";
            myData6.number = 105;
            myData6.name  = "sixth";
            myData7.number = 77;
            myData7.name  = "seventh";
            myData8.number = 120;
            myData8.name  = "eighth";



/*add the values to AVL*/
addAVLTree(tree, &myData1, compare);
addAVLTree(tree, &myData2, compare);
addAVLTree(tree, &myData3,compare);
printTree(tree, printVal);
printf("\n");
addAVLTree(tree, &myData4, compare);
//printTree(tree, printVal);
//printf("\n");


addAVLTree(tree, &myData5, compare);
printTree(tree, printVal);
```

```
        printf("\n");

        addAVLTree(tree, &myData6, compare);
        addAVLTree(tree, &myData7, compare);
        addAVLTree(tree, &myData8, compare);

        printTree(tree, printVal);
        printf("\n");

        removeAVLTree(tree, &myData8, compare);
        removeAVLTree(tree, &myData4, compare);
        //
        printTree(tree, printVal);
        //printf("\n");

        return 1;

    }
```