Homework #6
RELEASE DATE: 05/20/2014
DUE DATE: 06/03/2014, 17:00

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, either C or C++ or a mixture of them is allowed.

Note that this is the last homework of this semester and is therefore your last chance of using the MEDALs, if you still have any!

This homework set comes with 200 points and 20 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

6.1 Hashing, Skip List, Binary Search Tree

(1) (20%) Assume that we have millions of IDs and some duplicates may exist in these IDs. Describe a scheme on how the hashing mechanism may be used to facilitate the search for the duplicated IDs.

(2) (20%) Do Exercise R-9.16 of the textbook.

(3) (20%) Do Exercise R-10.6 of the textbook.

(4) (20%) Do Exercise R-10.13 of the textbook.

(5) (20%) Do Exercise C-10.4 of the textbook.

(6) (20%) Do Exercise C-10.15 of the textbook.

(7) (Bonus 20%) Do Exercise C-9.15 of the textbook.

6.2 Balanced Binary Search Trees

(1) (30%) `libavl` (`http://adtinfo.org/`) is a useful library for binary search trees. For instance, the following short code constructs an AVL tree of 16 integers and print it out.

```
#include <stdio.h>
#include <stdlib.h>
#include "avl.h"
void postorder_integer_avl(const struct avl_node *node){
  if (node == NULL)
    return;
  printf ("%d ", *((int *)node->avl_data));
  if (node->avl_link[0] != NULL || node->avl_link[1] != NULL){
      putchar('(');
      postorder_integer_avl(node->avl_link[0]);
      putchar(',');
      putchar(' ');
      postorder_integer_avl(node->avl_link[1]);
      putchar(')');
  }
}

int int_compare(const void *pa, const void *pb, void *param)
{
  int a = *(const int *)pa;
  int b = *(const int *)pb;

  if (a < b) return -1;
  else if (a > b) return +1;
  else return 0;
}

int main(){
  struct avl_table *tree;
  tree = avl_create(int_compare, NULL, NULL);

  int i;
  for(i=0;i<16;i++){
    int* element = (int*)malloc(sizeof(int));
    *element = i;
    void **p = avl_probe(tree, element);
  }

  postorder_integer_avl(tree->avl_root);
  puts("");
  return 0;
}
```

Note that the manual of `libavl` may be difficult to read; the code above comes from modifying the `avl-test.c` in `libavl`. When the code is compiled with `avl.c`, it correctly outputs an AVL tree.

```
7 (3 (1 (0 , 2 ), 5 (4 , 6 )), 11 (9 (8 , 10 ), 13 (12 , 14 (, 15 ))))
```

Write a program `hw6_2_1` that inserts the following 32 last names of professors in NTU CSIE (that can be compared lexicographically)

> Chuang, Chou, Chang, Chao, Chen, Cheng, Chu, Fu, Fuh, Hsiang, Hsu, Hsueh, Hung, Jang, Kao, Kuo, Lai, Lee, Liao, Lin, Liou, Liu, Lu, Lyuu, Ouhyoung, Oyang, Shih, Tsai, Tseng, Wang, Wu, Yang

using the order they are provided to the height-bounded binary search tree (`bst.c`), AVL tree (`avl.c`) and Red-Black tree (`rb.c`). Output the resulting trees with a format similar to the output above. **Print the output and attach it to your written part for the TA to grade.**

(2) (50%) There are 5 extremely hard homework sets in this course so far. Now, Hsuan-Tien wants to inquire the rank based on the scores on those sets. Nevertheless, the TAs haven't finished grading the homework sets from all students yet. So the score sheet keeps being updated. When TAs finish grading all homework sets from one single student, they will put the scores on the score sheet immediately. Therefore, the rank might change dynamically. Please help Hsuan-Tien solve this task with a modified AVL tree library.

The rank of students is calculated from the following criteria:

(a) the average score of 5 homework sets, rounded down to the closest integer

(b) if there is a tie, the score of homework 2

(c) if there is still a tie, the score of homework 3

(d) if there is still a tie, the score of homework 5

(e) if there is still a tie, the score of homework 4

(f) if there is still a tie, the score of homework 1

(g) if there is still a tie, the lexicographical order of the student ID

Yes, the student getting the highest score would be of rank 1.

The operations we need are as follows:

(a) `push` [ID] [score_1] [score_2] [score_3] [score_4] [score_5]
- action: add the score records of student 'ID' to the data structure

(b) `search` [R]
- action: search for the student of rank R

Note that student's ID is a string with length less than 20, and every student has a unique ID, which means there will be at most one `push` operation for each ID. Each score is an integer between 0 and 220. R is an integer between 1 and the number of students that have been graded. There will be no more than 1000000 operations.

For each of the "`search` [R]" operation, please output the ID for the student of rank R.

Sample Input

```
push b04902000 0 0 0 0 0
push b05902000 150 130 180 200 40
push b06902000 150 130 180 200 40
push b03902000 40 20 50 100 60
search 1
search 3
push b02902000 180 190 140 130 200
search 1
search 2
```

Sample Output

```
b05902000
b03902000
b02902000
b05902000
```

Time Limit: 5 seconds.

Hint: TA Yen-Chieh has already modified the AVL tree library to help you solve this problem. In particular, you can inquire the number of nodes in left sub-tree and right sub-tree of a node A by using $A \rightarrow avl\_cnode[0]$ and $A \rightarrow avl\_cnode[1]$ respectively.

Submission File (Program) and Written Copy

Please push your program to your forked repository `<user_name>/dsa14hw6` (on GitHub) before the deadline at 5:30pm on Tuesday (06/03/2014). We will use the latest time that you pushed to the repository as your submission time. Please **DO NOT PUT BINARY FILES** in your repository. As usual, the TAs will use CSIE R217 machines to test your code. Your repository should contain the following items:

- `hw6_2_1.c` or `hw6_2_1.cpp`
- `hw6_2_2.c` or `hw6_2_2.cpp`
- an optional `README`, anything you want the TAs to read before grading your code

The TAs will use the `Makefile` provided in the repository to test your code. Please make sure that your code can be compiled with the `Makefile` on CSIE R217 linux machines.
For all the problems that require illustrations, please submit a written (or printed) copy in class or to CSIE R217 before the deadline.

MEDAL USAGE: If you want to use the gold medals for this homework, please visit `http://main.learner.csie.ntu.edu.tw/php/dsa14spring/login_medal.php` and submit the request **before the deadline + 4 days** (6/7). **Note that this is the last homework of this semester and is therefore your last chance of using the MEDALs, if you still have any!**