

Homework #4

TAs' email: dsata AT csie DOT ntu DOT edu DOT tw

RELEASE DATE: 04/09/2013

DUE DATE: 05/02/2013, noon

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.

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.

4.1 Linked List, Stack, Queue, Vector, Tree

- (1) (15%) If a singly linked list is wrongly constructed, there would be a cycle within the linked list. **Use any search engine or consult any friend** to find an $O(N)$ time and $O(1)$ space algorithm that detects whether there is a cycle in a size- N linked list (hint: usually called Floyd's Cycle-Finding Algorithm). Learn and explain the algorithm clearly to the grading TA in your own words. Also, cite the website (or the person) that you learn the algorithm from.
- (2) (15%) Use any pseudo code to write down an algorithm that uses one queue (with **enqueue** and **dequeue** operations) and *at most* $O(1)$ of additional memory to simulate one stack.
- (3) (15%) Use any pseudo code to write down an algorithm that uses one fixed-size dense array (of sufficient size to hold the two stacks) and *at most* $O(1)$ of additional memory to simulate **two** stacks. You need to describe how **pop_stack_1**, **pop_stack_2**, **push_stack_1**, **push_stack_2** works with respect to what's being stored in the array.
- (4) (15%) Consider having two stacks for storing integers, a **red** one and a **blue** one with the **push** and **pop** operations only. Illustrate how to implement the **int pop_mth(color s, int m)** operation that pops the m -th element from the top (and only that element) from stack s . Note that **pop_mth(s, 1)** should be equivalent to the usual pop from stack s , and after the operation, all other elements should remain in their original orders.
- (5) (15%) Consider an implementation of the vector ADT using an extendable array, but instead of doubling the size of the underlying array when its capacity is reached, we "triple" the size of the underlying array — that is, changing the capacity from N to $3N$. Show that performing a sequence of n push operations still runs in $O(n)$ time in this case.
- (6) (15%) Assume that the in-order traversal of one binary tree to be GDHBIELJMACKF and its post-order traversal to be GHDILMJEBKFCA. Please draw the binary tree.

- (7) (Bonus 15%) The following code claims to reverse a stack “in place” (without using other stacks). Do you think the claim is true? Why or why not?

```

void reverse(Stack s){
    if (s.isEmpty()) return;
    int top = s.pop();
    reverse(s);
    pushDown(s, top);
}
void pushDown(Stack s, int value){
    if (s.isEmpty()) s.push(value);
    else{
        int tmp = s.pop();
        pushDown(s, value);
        s.push(tmp);
    }
}

```

4.2 Run-length Encoding

Run-length encoded vector is a very simple form of data compression in which *runs* of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. For example, consider a raw vector

$$x = (1, 1, 1, 1, 1, 4, 4, 4, 2, 2, 2, 8, 8)$$

Its run-length encoding is a list of pairs like

$$[(5, 1), (3, 4), (4, 2), (2, 8)]$$

which means there is five “1”, followed by three “4”, followed by four “2”, followed by two “8.”

- (1) (110%) You are asked to develop a container that stores an integer vector in its run-length encoding. You can choose any underlying container for storing the encoding. Your container should be “compatible” with `std::vector<int>` in the sense that the following member functions are supported:

- (constructor): The class must be named RLV and be of the form

```

template <class InputIterator>
    RLV (InputIterator first, InputIterator last)

```

The `InputIterator` can be anything that “points to” integers.

- `void insert(int pos, int value)`: insert *value* at *pos-th* position of the equivalent raw vector.
- `void erase(int pos)`: delete element from the equivalent raw vector at *pos-th* position.
- `void push_back(int value)`: insert *value* to the last position of the equivalent raw vector.
- `int pop_back()`: delete element from the equivalent raw vector at the last position and return it; return `MAXINT` if there is nothing to be popped.
- `int operator[] (int pos)`: return the element at *pos-th* position in the equivalent raw vector; return `MAXINT` if any error happens.
- `int size()`: return the size (length) of the raw vector. For instance, the *x* above is of size 14.
- `operator<<`: output the run-length encoded vector, like `5 1 3 4 4 2 2 8`, to an *ostream* object.
- `void clear()`: Removes all elements from the vector, leaving the container with a size of 0.

If the position is invalid (see below), please output a new line with message “**OHOHOH.\n**” (including the period and the change-of-line), and do nothing else.

- (1) `insert(int pos,int value)`: *pos* less than 0 or greater than (raw vector size) is invalid.
- (2) `erase(int pos)`: *pos* less than 0 or greater than (raw vector size - 1) is invalid.
- (3) `pop_back(int value)`: invalid when the equivalent raw vector is empty.
- (4) `operator[] (int pos)`: *pos* less than 0 or greater than (raw vector size - 1) is invalid.

We need to have an additional function over `vector` called `size_compressed()`.

- `int size_compressed()`: return the size (length) of the pair-list. For instance, the pair-list of *x* above is of size_compressed 4.

Grading

The TAs will be using an automatic judge with batches of input data. Therefore, it is important that your program follows some output format:

- All the **OHOHOH.\n** outputs should be sent to the outputstream `cout` immediately. Please search the web if you have no idea what this means.
- No new lines should be fed to output stream when using “`;;`” operator. That is, “`cout << rlv << endl;`” produces only one new line.
- Please do not have redundant spaces between outputs nor at the end of the output stream, i.e. “1 2 3 4” instead of “1 2 3 4 ” or “1 2 3 4”.
- When outputting an empty vector, the output should be fully empty.
- Please do output **OHOHOH.\n** immediately wherever needed.

Toy Program/Output

To be announced on the course website.

Submission File (Program) and Written Copy

You need to submit three files, `RLV.h`, `RLV.cpp` and the `Makefile`. You should declare the member functions in `RLV.h` and implement them in `RLV.cpp`. The TAs will use some demo programs named `main.cpp` and the command `make` and `make run` to test your program. Therefore you should compile `RLV.cpp` with a demo program named `main.cpp` and redirect standard output to a file name `output` in your makefile. Please upload your program as a single ZIP compressed file to CEIBA before the deadline at 3:30pm on Tuesday (04/30/2013). The zip file should be like `b86506054.zip`, where the file name should be changed to your own school ID. The ZIP file should contain the following items:

- `RLV.h`, `RLV.cpp`
- a `Makefile` to compile your code and run your program.
- an optional `README`, anything you want the TAs to read before grading your code

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 write down the number of medals that you want on the first page of your printed copy (something like “use 2 medals”). Use your medals wisely—usage cannot be retracted.