Review Activity 9 Sample Solutions

Stacks and Queues

1) Implement a non-recursive function named **remove_median_from_stack** that takes as input a sorted collection of integers stored in **stack p**. The function (1) iterates through the stack, (2) finds the median value inside the stack (i.e., the middle value for an odd number of elements, or the average between the two middle values for an even number of elements), (3) puts all the elements back in the original order but with the median value or values removed, and (4) returns the modified stack to the caller. That is, for input [1, 3, 5, 7, 55], **remove_median_from_stack** would return [1, 3, 7, 55].

Your function may only use one other Stack as part of its implementation; that is, the new **stack q**. You may also use a number of primitive temporary variables, but no other data structures. In your code, you may only call the **<stack>** methods, such as **bool empty(), int size(), void push (int value), void pop(), and int top()**.

```
stack<int> remove_median_from_stack (stack<int> p) {
      stack<int> q;
      // write your code here

if (p.empty())    return p;

int median1, median2;
if (p.size() % 2 = 0) { // check if the stack size is even
      median2 = p.size() / 2 - 1; // grab the middle two array elements
      median1 = p.size() / 2; // e.g., for size = 6, middle two are 2 and 3

} else {
      median1 = median2 = p.size() / 2;
}

int count = 0;
while (!p.empty()) { // move items from p to q
      int cur = p.top(); p.pop();
      if (count != median1 && count != median2)
            q.push(cur);
      ++count;
}

while (!q.empty()) { // place items back to p
      int cur = q.top(); q.pop();
      p.push(cur);
}

 return p;
}
```

2) Implement a non-recursive function named **transfer_stack** that takes as input a collection of integers stored in **stack p**, and transfers the elements from **p** into a new **stack q** while preserving the order from **p** in **q**. Once **q** includes all the data items from **p** and in the same order as in **p**, the function returns **q** and terminates.

Your function needs to use only one other **stack** to implement the algorithm; that is, the new **stack q**. You may also use a number of primitive temporary variables, but no other arrays, lists, stacks, or other data structures.

In your code, you may only call the **<stack>** methods, such as **bool empty(), int size(), void push (int value), void pop(), and int top()**. You may use helper functions if needed, but those need to be non-recursive too.

```
stack<int> transfer_stack(stack<int> p) {
      stack<int> q;
      // Implement your code here
    if (p.empty())   return q; // follows the intent of the question

    // iterate through the stack p.size() times
    int p_size = p.size(); // added as redundancy
    for (int count = 0; count < p_size; ++count) {
        // pop the top element from p, and store it as temp
        StackItem temp = p.top(); p.pop();
        // remove p.size() - count elements from p and store them onto q
        int p_size_cached = p.size();
        for (int i = 0; i < (p_ cached - count); ++i)   {
               StackItem cur = p.top(); p.pop();
               q.push(cur);
        }
        // store temp onto p
        p.push(temp);

        // move back all the elements back from q to p
        while(!q.empty()) {
               StackItem cur = q.top(); q.pop();
               p.push(cur);
        }
    }

    // move all the elements from p to q
    while(!p.empty()) {
         StackItem cur = p.top(); p.pop();
         q.push(cur);
    }
    return q;
}
```