

These must be completed and shown to your lab TA either by the end of this lab, or at the start of your next lab. You may work in groups of up to two people.

1. Complete the methods marked with TODO of the `CDate` class in `CDate.cc` (available under Lab 3 on the course web page). Look at `CDate.h` to give you an overview of the methods and instance variables. The only file you will need to change for this question is `CDate.cc`. Compile the code using `make` and run it using `./dates`. When you complete the code correctly, you should see the following output:

```
2015/1/1
0/0/0
0/0/0
0/0/0
2000/2/29
0/0/0
2014/12/31
0/0/0
2010/11/30
0/0/0
2012/2/29
2014/9/5
All tests passed.
```

Note that the test cases and `main()` function are located in `dates.cc`. Feel free to add more test cases to the bottom of the file, but don't make any changes to the ones that are already there.

Some helpful links on comparing strings and using switch statements in C++:

<http://www.cplusplus.com/reference/string/string/compare>

<http://www.cplusplus.com/doc/tutorial/control/#switch>

2. The second half of this lab deals with a recursive data structure called the Linked List. Complete the methods of the Linked List program (available under Lab 3 on the course web page). Compile using `make lists` and do an initial test run using `./lists`.

When you complete the functions correctly, the unit tests will all pass. Drawing pictures of the possible configurations that must be handled in each method will help. You will need to complete the following functions:

```
/**
 * Delete the last Node in the linked_list
 * PRE: head is the first Node in a linked_list (if NULL, linked_list is empty)
 * POST: the last Node of the linked_list has been removed
 * POST: if the linked_list is now empty, head has been changed
 */
void delete_last_element( Node*& head );

/**
 * Removes an existing Node (with key=oldKey) from the linked_list.
 * PRE: head is the first node in a linked_list (if NULL, linked_list is empty)
 * PRE: oldKey is the value of the key in the Node to be removed
 * POST: if no Node with key=oldKey exists, the linked_list has not changed
 * POST: if a Node with key=oldKey was found, then it was deleted
 * POST: other Nodes with key=oldKey might still be in the linked_list, but
 * POST: if the linked_list is now empty, head has been changed
```

```

*/
void remove( Node*& head, int oldKey);
/**
 * Insert a new Node (with key=newKey) after an existing Node (with key=oldKey)
 * If there is no existing Node with key=oldKey, then no action.
 * PRE: head is the first Node in a linked_list (if NULL, linked_list is empty)
 * PRE: oldKey is the value to look for (in the key of an existing Node)
 * PRE: newKey is the value of the key in the new Node (that might be inserted)
 * POST: If no Node with key=oldKey was found, linked_list has not changed
 * POST: Else a new Node (with key=newKey) is right after Node with key=oldKey.
 */
void insert_after( Node* head, int oldKey, int newKey );
/**
 * Create a new linked_list by merging two existing linked_lists.
 * PRE: list1 is the head of a linked_list (if NULL, then it is empty)
 * PRE: list2 is the head of another linked_list (if NULL, then it is empty)
 * POST: A new linked_list is returned containing new Nodes with the keys from
 * the Nodes in list1 and list2, starting with the key of the first Node of
 * list1, then the key of the first Node of list2, etc.
 * When one list is exhausted, the remaining keys come from the other list.
 * For example: [1, 2] and [3, 4, 5] would return [1, 3, 2, 4, 5]
 */
Node* interleave( Node* list1, Node* list2 );

```

When you complete the code correctly, you should see the following output:

```

<A> List 1: [3, 2, 1]
<B> List 2: [6, 7, 8, 9, 10]
<C> List 1: [3, 2]
<D> List 1: [3]
<E> List 1: []
<F> List 1: []
<G> List 1: [11, 12]
<H> List 1: [11, 12]
<I> List 1: [11, 12, 12]
<J> List 1: [11, 12]
<K> List 4: [11, 6, 12, 7, 8, 9, 10]
<L> List 4: [6, 6, 7, 7, 8, 8, 9, 9, 10, 10]
<M> List 4: [11, 12]
<N> List 4: [6, 7, 8, 9, 10]
<O> List 4: []
All tests passed.

```

3. Show your work to your TA either by the end of this lab or at the start of your next lab, or you will not receive credit for the lab!
4. **(Optional)** If you used a recursive approach to implement the `interleave` method, create another implementation using an iterative approach. If you used an iterative approach, now use a recursive approach.