ECE-250 – Algorithms and Data Structures (Winter 2012)

Assignment 3 (Partial)

This is only part of the assignment, to give you the opportunity to start working on these questions, related to topics that have been already covered in class. The rest of the questions will be posted by Friday or possibly Monday (most likely it will be two additional questions, and possibly a second bonus marks question).

All questions will have equal weight for the assignment grade (except for the bonus marks questions).

1 - Provide the definition of a function `swap_nodes` that swaps two elements in a doubly-linked list by readjusting links only (that is, the actual values stored in the two nodes are not touched or in any way moved). The function declaration is:

```
template <typename Type>
void swap_nodes (Node<Type> * node1, Node<Type> * node2);
```

2 - Given a hash table of size 16, with hash function $h(x) = x \bmod 16$, we want to insert prime numbers in sequence starting at 11 (i.e., 11, 13, 17, 19, 23, 29, $\cdots$) until two collisions occur — that is, include the number that causes the second collision.[1]

Show the above procedure (insertion by insertion, showing the contents of the array after each insertion) with collisions handled by:

(a) Linear probing.
(b) Double hashing, with the hash function for the jump size being $h_J(x) = (x \bmod 10)$ OR 1 (that is, we take the number modulo 10, and if it is even, we add 1, so that we can only obtain values 1, 3, 5, 7, or 9).

[1] To avoid unnecessary mistakes/oversights, you can check the list of prime numbers on Wikipedia, `http://en.wikipedia.org/wiki/List_of_prime_numbers`

10% Bonus Marks:

We want to implement an iterator class for a singly linked list with dummy or sentinel element. The linked list and node classes are declared as follows,[2] and the default constructor for List is shown below:

```
template <typename Type> class Node;

template <typename Type>
class List
{
public:
    class Iterator
    {
    public:
        Iterator (Node<Type> * node, const Node<Type> * sentinel)
            : d_node(node), d_sentinel(sentinel)
        {}

        void advance();
        Type retrieve() const;
        bool at_end() const;

    private:
        Node<Type> * d_node;
        Node<Type> * d_sentinel;
    };

    List()
        : d_head (new Node<Type>)
    {
        d_head->d_next = d_head;
    }

    Iterator begin() const;
    void insert (const Type & value, Iterator at);

private:
    Node<Type> * d_head;
};

template <typename Type>
class Node
{
public:
    Node (const Type & value = Type());

    Node<Type> * next() const;
    Type retrieve() const;

private:
    Type d_value;
    Node<Type> * d_next;
    friend class List<Type>;
};
```

Given this information (the "documentation" implied by the code sample), provide definitions for the three Iterator methods (`advance`, to move to the next element in the list; `retrieve` to get the value of the element "pointed at" by the iterator; and `at_end`, to check if we are outside the sequence of elements in the list).

Also, provide definitions for the two methods in class List: `begin`, to return an iterator pointing at the first element in the list; and `insert`, to insert a new element after the node pointed at by the

[2] The first line is a *forward declaration*, to break the mutual (circular) dependency — List needs to know about Node, and Node needs to know about List

given iterator. If the list is empty, `List::begin()` should return an iterator for which the method `at_end()` returns true.

A typical loop going through the elements in the list could look like this:

```
List<int> values;
// ...
for (List<int>::Iterator i = values.begin(); !i.at_end(); i.advance())
{
    // ...
}
```