**Solving the collision problem using Linked Lists.**

Start with the project posted with the homework.

[1] Make only one change to Program.cs:

Comment out this one line:

theHashTable.PrintState();

You do not have to implement that method. (You MAY of course, I had mine write out how many Link List items were in each array element.)

Now move to OurHashTable.cs

[2] Define a new class, I called mine LLnode, with just 2 properties, an int called key and a string called value. We will store objects of this type in our Linked List to hold the key and the value.

[3] Delete this definition

string[] hashTable; // Relying on new string array has all values = null

And replace it with this one

// this new array of linked lists is what you will use instead of the string[] above

LinkedList<LLnode>[] betterHashTable; // new hash table storage is an array of Linked lists

We are going to use the .NET LinkedList. The only methods that you will need from this built-in linked list,

\* this constructor new LinkedList<LLnode>()

\* the .AddFirst(LLnode object) (to add a new node to the Linked List)

\* the .ElementAt(index) (to get back an element out of the Linked List located at that index -- yes, that IS the name of the method!, I would have named it ReturnElementAt(index) but no one asked me!)

[4] So now the constructor for our OurHashTable class should be:

public OurHashTable(int pSize)

{

betterHashTable = new LinkedList<LLnode>[pSize]; // create an array of LLs

}

[5] Now you need to modify this method so that it correctly deals with collisions

public bool AddItem(int key, string value)

- refer to the HashTable ppt, slides 11 and 12, to figure out how to make this improved hash table deal with collisions

- First instantiate a new object instance of your LLnode, and set its properties. The key value gets the **key** value, NOT the **hashed key** value!

- You will still hash the key to get the pointer to which array element to store the data in.

- You will have to then see if that array element is null, or not. If it is null, you need to instantiate a new .NET linked list of type LLnode in that array element. After you create this new LL in that array element ..

- Then add your newly created LLNode object to this new linked list.

- If that array element is not null, that means there is already a LinkedList of LLnodes there, with one or more node. Just add your newly created LLNode object to ***top*** of that linked list

==================================

At this point you should be able to put a break point in at the end of this method, and then watch your code adding linked lists and nodes to those linked lists. I suggest you verify all is well before moving to the next step.

[6] Now modify this method public string GetItem(int key)

It needs to use the passed in key, hash that key, and go to that betterHashTable array element addressed by that hashkey. If that element is null, return an empty string “”, as there is no value in the table for that key.

If there is a linked list in that array element, you need to cycle through it using the .NET LL method .ElementAt(pointer) and loop through all members of this linked list until you find a match, or not. Check each one to see if the **passed in search key** matches the LLnode object’s **key** property (not the hash key). If you get a match, return the LLnode object’s value property. If you get to the end of the linked list and did not get a match, again, return an empty string “”, as there is no value in the table for that key.

At this point, your project should work just like before you started modifying it, EXCEPT you should never get lost data due to a hash collision! You should be able to insert 14 items into the table of size 15 over and over again, and never lose data.