Homework 5: Custom Dictionary

**Due: Sunday, November 17th by 11:59PM**

For this homework, your job is to create a basic implementation of a generic Dictionary class. Your implementation is not required to handle collisions (but will need to detect them). You are not allowed to actually use C#’s built-in Dictionary<,> class – your task is to make your own.

Below you’ll find the specifics of the classes and methods you are required to implement. The exact implementation details of the class’s methods are up to you, but you *should* ***not*** *be doing any linear searches through the whole array*! Use the hash function to look up a key’s index.

Feel free to include any helper methods you may need.

# Task Overview

This is a brief overview of the tasks you must complete for this assignment. Specifics are given in the corresponding sections later in this document. You should ***read the entire document*** before starting.

* Create a CustomPair<,> class to hold Key/Value pairs
* Create a CustomDictionary<,> class that represents a generic hash table
* Implement the required methods, plus any others you think you’d need
* Optional Challenge: Handle collisions

# CustomPair<,> Class

Before implementing the Dictionary itself, you’ll need a class which can hold a key/value pair. Internally the dictionary will hold multiple of these pairs, allowing you to verify keys later in the CustomDictionary class.

This class should be generic, much like the CustomDictionary class. Make sure it takes two type specifiers: The key’s type and the value’s type. It should also have the following:

* Two fields: A key and a value
* Two properties, each with a get/set: Key and Value
* A parameterized constructor which accepts the key and the value

# CustomDictionary<,> Class

The class should be called CustomDictionary, and must be Generic, taking two type specifiers: the type of the key, and the type of the value. Since the key’s type is generic, you may need to use .Equals( ) when comparing keys to each other.

## Fields

Your class should have two fields

* **data**: An *array* of CustomPair objects – This holds the actual keys and values of the dictionary
* **count**: A count of items in the dictionary

## Hash Function

Come up with a reasonable hash function for your class. You can take advantage of C#’s built-in GetHashCode() method to get an integer hash code for any object (the result of which can be a positive or negative number). You’ll need to ensure the resulting integer falls inside the range zero to the data array’s length. This makes sense as a helper method, since you’ll need to generate hashes quite often.

## Properties

You’ll need 2 properties for your class:

* **Count** – this will get the current count of actual data in the dictionary
* An indexer property – This will allow you to get or set individual values in the dictionary.
  + **Get**: Run the key through your hash function. If there is no data at that index, or the key stored at that index doesn’t match the provided key, throw a KeyNotFoundException(). Otherwise, return the value stored at that data.
  + **Set**: Run the key through the hash function. If there is no data at the resulting index, store a new CustomPair object there and increment count. If data existed already with the same key, overwrite the old value with the new one. If there was a collision (two keys with the same hash index), throw an ArgumentException with a message about key collisions.

## Constructors

Create default and parameterized constructors, which initialize the data array and set count to zero.

* Default constructor – Initialize the array with a size of 100.
* Parameterized constructor – This should accept a parameter that determines the initial size of the array.

## Methods

Implement the following methods in your CustomDictionary class as well

## bool ContainsKey(TKey key)

Determine if the specified key exists in the data array. Remember: Don’t just search through the array!

## void Add(TKey key, TValue value)

If the key doesn’t exist in the dictionary already, add the key/value CustomPair and increment count. If the key already exists or a collision occurred, throw an ArgumentException with an appropriate message.

## bool Remove(TKey key)

Remove the data from the array (by replacing it with null) if the key exists, decrement count and then return true. If the key doesn’t exist, simply return false.

## void Clear( )

Loop through the data array and replace all values with null, then set count to zero.

# Main Method

Your main method should create a CustomDictionary<String, String>. Seed the dictionary with at least 6 initial key/value pairs - you can hardcode these: use the *Add* method for 3 of the pairs and the *set property* for the other 3. Then loop and allow the user to enter commands:

|  |  |
| --- | --- |
| **Command** | **Action** |
| quit | Ends the loop (and the program) |
| count | Print the current count of items in the dictionary. |
| add | Ask the user for a key, then a value. Store them in the dictionary.  Catch any exceptions (to prove your exceptions work) and print their messages. |
| remove | Ask the user for a key, and attempt to remove it from the dictionary.  Let the user know if it worked. |
| clear | Clear the dictionary. |
| get | Ask the user for a key. Use ContainsKey() to determine if the key exists and retrieve it.  Print the value at that key, or a message about the key not existing. |
| set | Ask the user for a key, then a value. Set the entry at the given key to the specified value. |

# Sample Run

Note: The following keys were added when the dictionary was created - pizza, hamburger, taco, milkshake, stirfry, and pretzel.

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Count  
The dictionary has 6 entries.

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Set

Key: cupcake   
Value: chocolate with caramel frosting

The key ‘cupcake’ was added

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Count  
The dictionary has 7 entries.

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Set

Key: pizza   
Value: sausage and pepperoni

The value was changed for the key ‘pizza’

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Get

Key: milkshake

Value is: ‘cookies and cream’

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Get

Key: coffee

That key does not exist.

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Add

Key: coffee

Value: vanilla latte

The key ‘coffee’ was added

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Add

Key: coffee

Value: vanilla latte

Error! That key already exists.

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Remove

Key: coffee

The key ‘coffee’ was removed

<Continued on next page>

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Remove

Key: cereal

The key ‘cereal’ is not in the dictionary.

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Clear

Dictionary was cleared

Custom Dictionary menu:  
Count Add Remove Get Set Clear Quit >> Count  
The dictionary has 0 entries.

# Optional Challenge (Up to +10 points):

As a bonus challenge, your job is to properly handle hash table collisions. Once you have the rest of the homework completed, alter your CustomDictionary class to handle cases where two different objects hash to the same index. Use buckets to handle your collision handling for this assignment.

This means that all entries with unique keys will be successfully added to the dictionary. Collisions will not need to throw exceptions anymore – instead, they’ll be handled by the dictionary and all data will be stored. You’ll need to update several methods and/or properties to get this working.

Be sure to thoroughly test your newly updated dictionary when you’re done. The easiest way to ensure you get collisions is by making your initial dictionary size very small. Adding 5 items to a dictionary with an internal array of size 4 means at least one collision will occur (and probably more).

**Only attempt this challenge if you are absolutely sure you can successfully complete ALL of these requirements in time. Save a copy of your dictionary before attempting this challenge in case you need to go back to your original version!**