CMPUT 275 - Tangible Computing **Winter 2020**
Weekly Exercise #5

Dynamic Hashing

In our hash table implementation, the number of buckets is fixed at the time the table is instantiated. If you end up inserting many more items than there are buckets, the table becomes congested and we no longer get the "typical" $O(1)$ time when adding, looking up, or removing an item.

Dynamic Hashing

Your task is to extend the hash table created in class to implement **dynamic hashing**, so that it will resize the table dynamically based on the number of items in the hash table. You will also modify the constructor so that the user can not specify the table size (ie there is no parameter) when they instantiate a new hash table. When a hash table is instantiated by the user the default size should be 10. The user should **not** be able to specify the number of buckets in the constructor. As discussed in class, the basic approach you should take is whenever the number of items exceeds the number of buckets (i.e., the table starts to become congested):

1. Allocate a new array of buckets (each holding a linked list) that is twice the size of the current array of buckets.
2. For all items in the old hash table, you should recompute the bucket they should occupy in the new array of buckets, and add the item to the list in that new bucket.

To save space, you also need to shrink the table when it becomes too sparse. The basic approach you should take is whenever the number of items is less than 1/4 of the number of buckets and the number of buckets is greater than the default size of 10:

1. Allocate a new array of buckets (each holding a linked list). If the old number of buckets is B, then you should allocate $\max(B/2, 10)$ buckets (using integer division).
2. For all items in the old hash table, you should recompute the bucket they should occupy in the new array of buckets, and add the item to the list in that new bucket

Note: to avoid memory leaks it is important that your implementation deallocates any storage that is no longer being used.

Your solution must build off of the file `hash_table.h` that is posted with this weekly exercise in eClass. This is a slightly cleaned up version of the hash table seen in class. You must modify the constructor so that it does not accept any parameters. You may add new private methods to the class if you desire, but no other public methods should be added.

Your final implementation should work with the file `exercise5.cpp` that is posted with this weekly exercise **without modification**. We will be testing that your submitted `hash_table.h` file works with the original version of `exercise5.cpp`.

Makefile

In this weekly exercise, you must include a custom `Makefile` with the following targets:

- The main target `exercise5` which simply links `exercise5.o` to generate an executable called `exercise5`. This should be the topmost target, so it can be built simply by typing `make`.
- The target `exercise5.o` which compiles the object.
- The target `clean` which removes the object `exercise5.o` and the executable `exercise5`.

Make sure all dependencies are properly listed for each target. Recall that with template code, you cannot separate the implementation from the header. So it is acceptable to put implementations of functions with template arguments in a `.h` header file.

In this particular weekly exercise, you only have a single `.cpp` file. As per the instructions above, you should still compile it to an object `.o` file in one target and link it to the final executable in another target in the `Makefile`.

A Sample Interaction

We should be able to compile and run your solution in the following way demonstrated below. We promise that we will not test it with a remove (`R`) command that tries to remove something that is not already in the table. Note how your solution should put the executable in the same directory as the `Makefile` itself.

```
> make
<OUTPUT FROM COMPILING/LINKING>
> ./exercise5
I 14
I 19
I 27
I 32
I 19
S
size is 4
Q 27
found
Q 101
not found
Q 19
found
R 19
Q 19
not found
Q 32
found
R 27
S
size is 2
I 19
Q 19
found
STOP
```

Submission Details:

Compress all the following files in a compressed archive called `exercise5.tar.gz` or `exercise5.zip`. Submit only this `.tar.gz` or `.zip`.

- `exercise5.cpp` - Unmodified but included so we can compile your submission immediately by typing `make` after decompressing. This must be exactly the same file that we posted to eClass with this exercise.
- `hash_table.h` - Your implementation of the dynamic hash table.
- `linked_list.h` - Unmodified linked list implementation.
- `dynarray.h` - Unmodified dynamic array implementation.
- `Makefile` - Generates the `exercise5` executable.
- `README`

Make sure to submit a clean compressed archive (i.e., no object files or executables should be included in your submission).