Hash Function/Table Lab

Assigned: 2/10/2020
Due: end of lab

This is an individual lab. The purpose of this lab is to apply what we learned with congruences and modularity to create a hash functions and ultimately, a hash table. You must demo the lab to the instructor and submit this paper upon completion. Make sure your name is on this paper.

Background:
In computer science and software engineering, the more efficiently a program runs, the better in quality it is (or at least that's how it can be classified). Searching for a certain value in a large database or list can take a long time especially if that value is stored in a space that is the last place that you look. See the example below. This is where a hash function comes into play.

If you tried to find `32532` in the array below, it would take a long time because `32532` is at the end of the array. You would have to check around 20483960 memory addresses before finding it.

Index	0	1	2	3	4	5	6	…	20483958	20483959	20483960	20483961
Value	#	#	#	#	#	#	#	…	49842	428	32532	35246

A hash function is a function that generates a key for a value. This key allows us to (1) strategically place our data and (2) easily find our data when we need to refer to it. There are many types of hashing, but we're going to use **modular hashing**. Let's look at a simplified example:

293587
85023840
54234914
39482905
9850293
209524
98520398
623407

Let's use the array on the left as our example. There are 8 numbers in our array and we want to arrange them in a way that allows us to find them as quickly as possible. Because there are 8 spots in the array, we are going to use mod 8 to define these locations for us. For example:
`293587`: 293587 mod 8 = 3. Thus we'll put this number in index 3
`85023840`: 85023840 mod 8 = 0. Thus we'll put this number in index 0
Continue this pattern….

Finally we'll have an array that looks like below. This is called a hash table:

0	85023840
1	39482905
2	54234914
3	293587
4	209524
5	9850293
6	98520398
7	623407

Let's say I want to find or have access to a certain number such as `98520398`.
If I wanted to use a brute force algorithm to find it in the array above, it would take me 7 tries if I started from the beginning of the array and 2 tries if I started at the end of the array.
However, if I want to find it in my hash table, I would plug it into the hash function that I used to create the hash table `x mod 8` and I would get a remainder of 7. In 1 try, I can find the value I was looking for.

****Note: This is a very simplified example. In real life situations, you have to account for much more such as collisions. Collisions are outside the scope of this lab and we will not be worrying about it.

Lab Introduction:
You may choose Java or Python to complete the lab. Be sure to check BeachBoard for the resources that you need such as the skeleton code and a video.

Instructions:
In the skeleton note the following:

- The TABLESIZE is initialized for you already (=9). Do not change it or else your output may not match the expected output
- There is an object called Character that has the following
  - Variables:
    - `(String) firstName`
    - `(String) lastName`
  - Methods:
    - `(int) key()`
      - For now, it returns the ascii value of the first letter of `lastName`
    - A "toString" method so that it prints out the details of the character
    - `String getLastName()` returns the last name (getter)
- A hash table of size 9 is already initialized (but empty)

Your task:

1. Change the body of `key()` so that it does the following:
   - Generates the proper key between 0-8 (because there are 9 spots in the hash table) through a hashing function in terms of the ascii value of the **first letter** of `lastName`
2. Before moving on to the main function implementation, look at the following characters. Next to them, write the hash values for the current implementation (bold faced names are the last names). You may want to refer to the Ascii table:

A-65	B-66	C-67	D-68	E-69	F-70	G-71	H-72	I-73	J-74
K-75	L-76	M-77	N-78	O-79	P-80	Q-81	R-82	S-83	T-84
U-85	V-86	W-87	X-88	Y-89	Z-90	a-97	b-98	c-99	d-100
e-101	f-102	g-103	h-104	i-105	j-106	k-107	l-108	m-109	n-110
o-111	p-112	q-113	r-114	s-115	t-116	u-117	v-118	w-119	x-120
y-121	z-122								

Anakin **Skywalker**: _____
Obi Wan **Kenobi**: _____
General **Grievous**: _____
Sheev **Palpatine**: _____
r2 **d2**: _____
Leia **Organa**: _____
Ahsoka **Tano**: _____
Sheev **palpatine**: _____
Boba **Fett:** _____
Mace **Windu**: _____
Asajj **Ventress**: _____
General **Hux**: _____

****Test your code with some of the names above to see if you get the correct key before moving on to step 3

3. Using the Characters above (and in that order), simulate what would happen if you tried to insert them into hash table using their respective keys. In the case there is a collision (when you try to insert a Character into a space that already has a character), then prioritize the first character and ignore the second character.

Index	0	1	2	3	4	5	6	7	8
Character									

4. On a loop, ask the user for a first name and a last name. Create a Character object with each set of inputs. Find the key of the Character input to see if it can fit in our hash table. If it does, insert it into the hash table. If it does not, output an error message saying that there's no room for that character. Continue to ask them until they fill out all spots in the hash table. Look at the example below:

```
Last name: Skywalker
First name: Anakin
Inserted into the hash table,it is
----------------------------------------
Last name: Palpatine
First name: Sheev
Inserted into the hash table,it is
----------------------------------------
Last name: Grievous
First name: General
Error: Filled that spot you already have
----------------------------------------
Last name:
```

5. Once the whole hash table has been filled, prompt the user to ask for a last name. Search through the last name using two methods: brute force and hashing. Time the two methods and print out the index they were found in and the amount of time it took to find them

```
Finished Filling Hash Table!
Last name: Organa

Using brute force
Index: 7
Time: 222ms

Using hashing
Index: 7
Time: 4ms
```

6. Demo and submit. NOTE: **<u>The sequence that I gave to you in the lab was so that you can test. This may or may not be the actual sequence that I test during the demo!</u>**