Hashing

H1. Give an example of a bad hash function for strings (that generates many collisions). Justify why it is bad: find some strings that will hash to the same cell.

H2. A hash table has 1000 slots and 200 items have already been hashed in it. What is the load factor?

H3. In the hash table below, two items are originally hashed to the same slot. The slots examined for inserting one of them are shown by a star and the slots examined for inserting the other are shown by a plus. You can assume that the table size is very big (e.g. more than 1000) and the slots shown did not require a mod (%) operation (that is we did not have to wrap around).

a) What type of open addressing was used? Justify.
b) Give the next slots to be checked for each item (show where the next star and where the next plus will be).

Index	
…	
5	+ *
…	
8	+
9	*
…	
11	+
12	
13	*
14	+
…	
17	*
…	

H4. The images below show the occupancy of two hash tables. Both tables have **the same size**, the **same items** hashed in them, and both use **open addressing**. However they differ in the way they find an available slot in the table. Which one is a better hash table and why? (You do NOT have to deduce how they find the next available slot. You simply have to judge which one would behave better based on this image.)

P5. You want to hash integers in a table of size 9 and use quadratic probing. Give an appropriate hash function for this table (give the math function, not the C code for it). Give 4 different integers that hash to the same value. Draw the table (with all its cells) and show where these numbers are inserted (hashed) in the table.

P6. A hash table of size 11 with open addressing, probes the following cells in order to hash an item: **6, 9, 3, 10**. What kind of open addressing method does it use? (You do not need to give the exact formula, just the type (name) for the open addressing used.) Justify your answer (specify what made you draw the conclusion that you did).

P7. Assume you have a hash table of **size 20**, that uses **double hashing** with the first hash function $\mathbf{h_1(x) = x \%20}$ and the second hash function $\mathbf{h_2(x) = 1+(x\%9)}$. Give the first 4 probes (indexes) in the sequence of probes generated by double hashing for the key **x = 13**. Show the index and the calculations in the table below.

Probe	Index	Calculation that resulted in the Index in the middle column.
1st		
2nd		
3rd		
4th		

P8. You have a hash table of size M that uses linear probing and is half full.

a) What is the load factor of this table?
b) What is the average time to insert an item? (Assume no deletions were done.)
    a. All occupied cells are consecutive (e.g. left half is full, right half is empty)
    b. Occupied and empty cells alternate.
c) How long does it take to search for an item. Note that you must now differentiate between EMPTY and DELETED cells. (A DELTED cell is one where there was an item and later that item was removed from the table)

P9. We know that linear probing is bad because it creates long chains and moreover, the longer the chain, the more likely it is to grow. What exactly is so bad about long chains? Wouldn't the data get in the table anyway? Why is it worse that it is grouped in these long chains rather than be more spread out? You answer cannot be along the lines "because if the data is more spread, the table is better". It has to be a justification involving the expected time needed to insert a new item.

Added 12/5/2017:

P10.You want to hash integers in a table of size 9 and use quadratic probing. Give an appropriate hash function for this table (give the math function, not the C code for it). Give 4 different integers that hash to the same value. Draw the table (with all its cells) and show where these numbers are inserted (hashed) in the table.