2) (5 pts) ALG (Hash Tables)

Consider the following problem: Suppose we have a rather large text file that contains up to 50 million strings of 9-digit integers (each separated by a space), and we know that **exactly one** of those integers occurs twice. (The rest, we know for certain, occur only once.) Suppose our goal is to find the single 9-digit integer that occurs twice. Some clever cheddar of a computer scientist thinks they have come up with a great solution to this problem. They have a rock-solid implementation of hash tables (written in C) that has been thoroughly tested by the open-source community and is known to be reliable and bug-free. This implementation of hash tables has some great features:

- The hash tables have a lovely, effective, widely-used hash function for dealing with integers.
- The hash tables expand automatically without creating any memory leaks.
- In addition to keeping track of the elements in the hash table, this implementation keeps track of a few extra pieces of information inside the HashTable struct. One of those extra pieces of information is how many collisions have occurred over the lifetime of this hash table. (That value is, of course, initialized to zero (0) by the function used to create new hash tables.)

It is this last property that has caught our clever cheddar's eye. They reason that they can loop through the input file, throw each integer into a hash table one-by-one, and as soon as a collision occurs in the hash table, they must have found the duplicate integer they were searching for! So, they write up a piece of code that effectively does this:

```
function find_that_duplicate(file):
        create a hash table, h
        for each integer, n, in file:
                insert n into h
                if h->num_collisions > 0:
                        destroy h
                        close file
                        return n
```

The idea above is just pseudocode, of course. You may assume that the person implementing this in C is careful to avoid memory leaks and segmentation faults, that there is enough memory to hold all those integers in the hash table, that the hash table implementation works well and is rock solid, and that the code has no syntax errors. You may also assume this function will **only** be called with a file that actually exists, and that we know for sure there is exactly one duplicate integer in that file. (So, we do not need to worry about what happens if there is no duplicate.)

What is wrong with this proposed solution? In your response, be as clear and precise as possible. Long, rambling word salads will not receive credit.

A collision does **not** imply equality. Two distinct elements can collide in a hash table. So, this solution does not necessarily find the duplicate integer. It could return prematurely with an incorrect integer.

Grading: 5 pts for correct answer. (They just need the part about collision not implying equality.) Award only 3 pts if they seem to be on the right track but are being imprecise and possibly cheesing it. Award 1 pt if they say, "duplicate value could end up in different indexes."