More dataTable issues

Physical storage issues

This is an EXTERNAL STORAGE structure, so all processing of the file handles it as a FILE (except you have to read a record into memory to be able to look at it). There is never more than a single record in memory at once. You MIGHT, however, have storage allocated for 2 records in memory, depending on how you're doing the processing:

- One for reading a record into the file (which might be a good record OR it might be an empty location)
- One for preparing the record to be written to the file

Exception for HeaderRec data

Since the headerRecord fields are constantly being used and, for the 2 counters, constantly being changed, they should be stored in memory during the entire run. Just as you're handling nameIndex as internal storage:

- At the start of UserApp (i.e., dataTable's constructor), the 3 fields should be read into memory for fast access
- At the end of Setup and at the end of UserApp (i.e., dataTable's finishUp method), the 3 fields should be written out onto the file (after a seek to byte 0)

Fix sizeOfDataRec & sizeOfHeaderRec

so byteOffset calculations are correct , since dataTypes (& thus field-sizes have changed)

2 symptoms of an empty storage location in the homeArea:

1) All 0 bits (i.e., 0's in int/short/long/double/float fields & null-char's in charArray fields) [NOTE: Is this overkill? Could you just check a single field and be 100% sure?]
2) Read fails (so the read didn't work, so the read didn't bring data in from the file into memory, so what is sitting in that memory location? Good-looking garbage! Beware!)

FAQ:

- Is there an array of dataRecords in memory? NO. dataTable an EXTERNAL structure.
- Should the file space be initialized before inserting any records? NO. See note on 2 symptoms of an empty location, below.
- How are locations "given out"?
  - For records which go in the HOME area:
    the homeAddress is calculated by the HashFunction
  - For records which go in the COLLISION area:
    the nextEmptyRRN location, which is: MAX_N_HOME_LOC + nColl + 1
- Will it be necessary to test whether a location is empty or not?
  - Yes, for the HOME area
  - No, for the COLLISION area, since you're just writing to the nextEmptyRRN
- Will there be any empty locations in the "middle" of the file?
  - Yes, there probably will be in the HOME area
  - No, there won't be in the COLLISION area since delete's a dummy stub
- Do nHome and nColl need initializing? Yes, they're counters

Chains (linked lists) – 1 per synonym family

- HP is stored in the link field of the record in the HOME area
  HOWEVER, the record in the HOME area is NOT PART OF THE CHAIN.
  This is just a handy place to store the HP since there's only 1 HP per synonym family.
- New collision nodes are always inserted on the FRONT of the chain,
  NOT THE BACK (and the chains are NOT an ordered lists)
  (We want efficient insertions – we don't want to "travel down the chain".
  And nobody said anything about "fairness".)
- Inserting in a linked list requires these 3 steps:
  1. Physically store the node in nextEmptyRRN location in collision area - that is, at MAX_N_HOME_LOC + nColl + 1
  2. Hang the node in linked list (at the FRONT)
     a. this new node's link = HP link value
     b. HP = this new node's storage location, which is nextEmptyRRN
  3. Increment nColl
- How do you know where the nextEmpty is?
  - It was initially MAX_N_HOME_LOC + 1
    (that is, 21, since HOME area uses 1-20)
  - When you did the physical storing & incrementing above, you added 1 to nColl. So nextEmptyRRN = MAX_N_HOME_LOC + nColl +1