HASHING CONCEPTS

Hashing needs 2 algorithms:

1. A **hash function** (HF) – where the key field and MaxN value are supplied, and the homeAddress is returned
2. A **collision resolution algorithm** (CRA) – determines where collisions are stored and how they'll be accessed. [A record is a collision if it can NOT be stored in its homeAddress because another record is already there].

There should ALWAYS be a physically separate hashFunction method with that name, even if it's only a couple lines of code. That allows for easily changing the function itself without affecting anything else in the program

All records which hash to the same homeAddress are called a "synonym family". The first record hashing to a particular homeAddress is stored in that home location. Subsequent records in that synonym family (whether during the initial loading in SetUp or during transaction processing in UserApp) are collisions and must be stored elsewhere, depending on the CRA used.

Static hashing (*vs. dynamic hashing*) means that the storage size is determined by compile time (*rather than at run-time, which dynamic hashing does, and thus would be gradually increased, as needed*). So it's a fixed size, specified by a MAX value

- "Storage size" refers to the number of storage locations allocated, either
  - in the entire table/file, when "EMBEDDED collision area" is used, including storage for both the home records and the collision records
  - in the home area of the table/file, when "SEPARATE collision area" is used, including just the storage for the home records – since typically there are relatively few collision records
- These MAX values are hard-coded as named constants in their appropriate class, as MAX_N_LOC or MAX_N_HOME_LOC respectively. Throughout the rest of the program, only the named constants are used, not their integer values.

###################### COLLISION RESOLUTION ######################

The **collision resolution algorithms** include 2 different decisions as to where collision records are stored:

1. Embedded (in amongst the home records) OR
   Separate (in a physically separate area from the home records)
2. Using a Linear algorithm (with wrap-around) OR
   Chaining algorithm OR
   others. . .

Using a separate overflow for collision storage could either be implemented as:

- A physically separate file/table from the one used for the home area
- OR as the next part of the same file/table after the final storage location allocated for the home area

Linear (or Progressive overflow) with wrap-around with **Embedded** Overflow

- home area & collision area share locations: 1 to MAX_N_LOC
- wrap-around → add 1 **mod** MAX_N_LOC

Chaining with **Separate** Overflow

- 2 separate areas:
  - home area: locations 0 to MAX_N_HOME_LOC – 1
  - collision area: locations MAX_N_HOME_LOC → ????
- A chain is a linked lists (of just a single synonym family)
  - only collision records are part of a chain
  - the record in homeAddress of that synonym family is NOT part of chain
  - the HP of each chain is stored in link field of record in homeAddress
- There will be MAX_N_HOME_LOC distinct chains, 1 per synonym family (though some may be empty)
- The links of the linked list (& the HPs) are:
  - subscripts, if this is a hash TABLE (using an array)
  - RRNs, if this is a hash FILE
- -1 is used for a link which "points nowhere" (including for HPs which point nowhere)
- Typically, for efficiency sake,
  - the chain is not maintained as an ordered list
  - new collisions inserted on front of chain (not on the back)

Space management for Chaining/Separate: - steps for inserting:

1. physically store node in nextEmpty location in collision area – i.e., at MAX_N_HOME_LOC + nColl
2. insert node on FRONT of chain (for this synonym family)
   a. this new node's link = HP (for this synonym family)
   b. HP (for this synonym family) = this new node's storage location (see step 1 for its address)
3. increment nColl