**CSCI 360 Hashing and Hash Tables**

Up to this point, you have been building tables sequentially and performing a linear search when a record was to be processed. This process is alright for smaller tables, but it becomes a waste of time when a table is large. So, a new random search, known as **hashing**, will be introduced.

Hashing can be used to both build and search a table. The basic idea behind hashing is to take a key and convert it through some fixed process to a **hash key** in the range of 0 to n-1, where n is the maximum number of slots in the table. The hash key will then be used to either store or find an item in the table.

There is the possibility that two keys could resolve to the same hash key. This situation is known as a **COLLISION**. When this occurs, the item will be stored in the next available slot in the table, assuming that the table is not already full.

The process of finding the next available slot when a collision occurs is known as a **LINEAR PROBE**. The linear probe is implemented via a linear search from the point of collision. If the physical end of table is reached during the linear search, the search will wrap around to the top of the table and continue from there.

If an empty slot is not found before reaching the point of collision, the table is full.

The fixed process to convert a key to a hash key is known as a **hash function**. This function will be used whenever access to the table is needed.

The most common method of determining a hash key is the **division method of hashing**. The formula that will be used is:

hash key = key % number of entries in the table

The above formula uses the remainder of the division of the key by the number of entries in the table as the hash key

For example, if we assume a table that can hold 8 items:

hash key = key % number of entries in the table

4 = 36 % 8

2 = 18 % 8

0 = 72 % 8

3 = 43 % 8

6 = 6 % 8

2 = 10 % 8

Assembler does not allow subscripting/indexing so a displacement into the table must be calculated using the hash key. The formula for calculating a displacement into the table is:

displacement = hash key \* length of a table entry

The displacement can then be used to calculate the table entry address.

table entry address = displacement + beginning address of the table

**Hash Routine Pseudocode**

searchKey is the key field of the entry to insert or the key that is  
 being searched for  
  
Calculate the table entry address  
Save the table entry address  
  
Set the return code to -1  
Do while (the return code is -1)  
 If an empty table slot is found  
 Set the return code to 4  
 Return the address of this table slot|  
 Else  
 If table entry's key is equal to searchKey  
 Set the return code to 0  
 Return the address of this table slot  
 Else  
 Increment the table pointer to the next entry  
 If the physical end of table is reached  
 Wrap around to the beginning of the table  
 Endif  
 If the table pointer is equal to saved table entry address

Set the return code to 8  
 Endif

Endif

Endif

Enddo

The return codes that are set in the pseudocode have a different meaning depending on whether an item is being inserted into the table or if the table is being searched for an item.

Return code meanings when **INSERTING** into the table:

Return Code Meaning  
 0 A duplicate key has been found. DO NOT insert.  
 4 An empty slot has been found. OK to insert.  
 8 Table is full. DO NOT insert.

Return code meanings when **SEARCHING** the table for a key:  
  
 Return Code Meaning  
 0 Successful Search. Key found.  
 4 Unsuccessful Search. Empty slot found.  
 8 Unsuccessful Search. Entire table has been searched.