Hash Tables
Buckets/Chaining
There are two general approaches to resolving collisions:

1. Open address hashing: if a spot is full, probe for next empty spot

2. Chaining (or buckets): keep a collection at each table entry
Resolving Collisions: **Chaining / Buckets**

Maintain a collection *(typically a Map ADT)* at each table entry:

Each collection is called a ‘bucket’ or a ‘chain’

![Diagram showing a collection of names in buckets]

- Angie → Robert
- Linda
- Joe → Max → John
- Abigail → Mark
# Hash Table Implementation: Initialization

```c
struct HashTable {
    struct LinkedList **table;  /* Hash table → Array of Lists. */
    int capacity;
    int count;
}

void initHashTable(struct HashTable *ht, int size) {
    int i;

    ht->capacity = size;
    ht->count = 0;
    ht->table = malloc(ht->capacity * sizeof(struct LinkedList *));
    assert(ht->table != 0);
    for(i = 0; i < ht->capacity; i++) ht->table[i] = newList();
}
```
void addHashTable(struct HashTable *ht, TYPE val) {
    /* Compute hash table bucket index. */
    int idx = hash(val) % ht->capacity;
    if (idx < 0) idx += ht->capacity;

    /* Add to bucket. */
    addList(ht->table[idx], val);
    ht->count++;

    /* Next step: Reorganize if load factor to large. More on this later! */
}
• **Contains**: find correct bucket using the hash function, then checks to see if element is in the linked list

• **Remove**: if element is in the table (e.g. contains() returns true, remove it from the linked list and decrement the count
Hash Table Size

• Load factor:

\[ \lambda = \frac{n}{m} \]

– Load factor represents average number of elements in each bucket

– For chaining, load factor can be greater than 1

• To maintain good performance: if load factor becomes larger than some fixed limit (say, 8) ➔ double table size
Hash Table

• Load factor:

\[ \lambda / m = n \]

- The average number of links traversed in successful searches, \( S \), and unsuccessful searches, \( U \), is

\[ S \approx \frac{\lambda}{2} \]
\[ U \approx \lambda \]

- If load factor becomes larger than some fixed limit (say, 8) \( \rightarrow \) double table size
Hash Tables: Algorithmic Complexity

• Assuming:
  - Time to compute hash function is constant
  - Chaining uses a linked list
  - Worst case analysis $\rightarrow$ All values hash to same position
  - Best case analysis $\rightarrow$ Hash function uniformly distributes the values and no collisions

• Contains operation:
  - Worst case for chaining $\rightarrow$ $O(n)$
  - Best case for chaining $\rightarrow$ $O(1)$
• Assume hash function distributes elements uniformly (a BIG if)
• And we have *collisions*
• Average case for all operations: $O(\lambda)$
  and $\lambda = n/m = O(m)/m = O(1)$
• Want to keep the load factor relatively small
• Resize table *(doubling its size)* if load factor is larger than some fixed limit *(e.g., 8)*
  – Only improves things *IF* hash function distributes values uniformly
  – How do we handle a resize?
Design Decisions

• Implement the Map interface to store values with keys (ie. implement a dictionary)

• Rather than store linked lists, build the linked lists directly
  – Link **hashTable;
Your Turn

• Worksheet 38: Hash Tables using Buckets