Data Structures and Algorithms- Lab 12

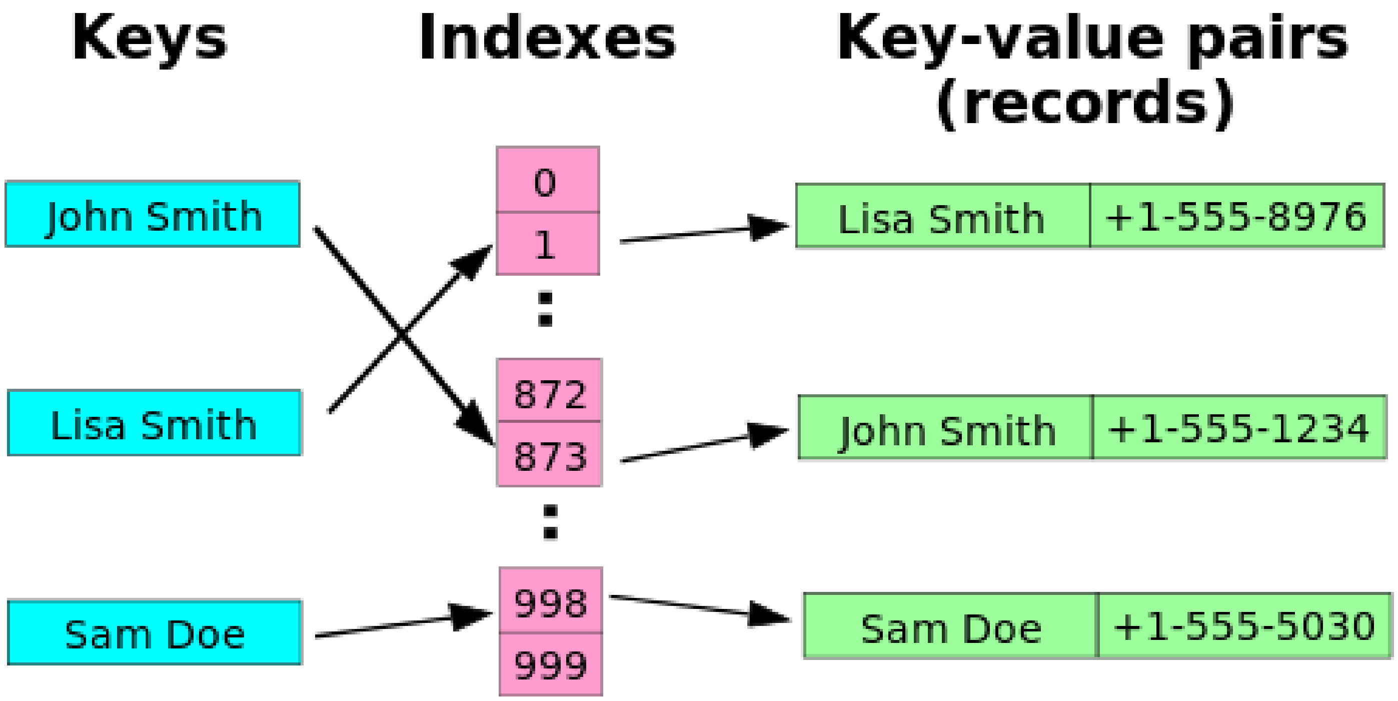
Iulia-Cristina Stanica
iulia.stanica@gmail.com

Roadmap
Hash Tables

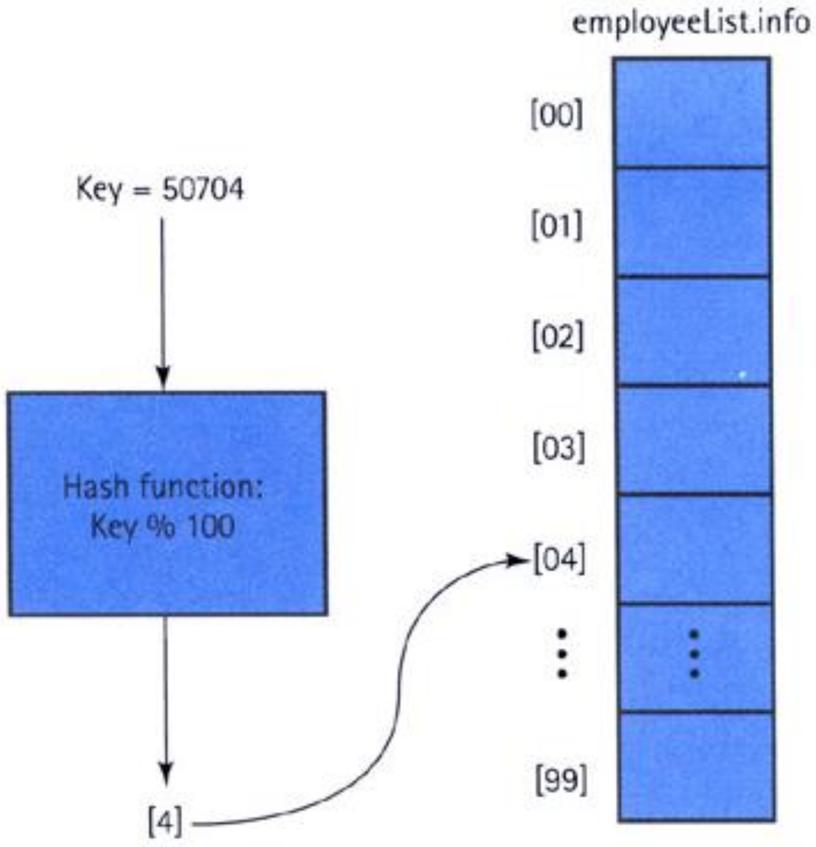
- Hash Tables
- Collision-handling
- C++ implementation

Hash Tables

- **Hash table** = a data structure which allows key – element associations.



Hash function



- Used to identify the place of the element inside the table
- The elements are classified based on a certain function depending on the key: the hash function.

Collisions

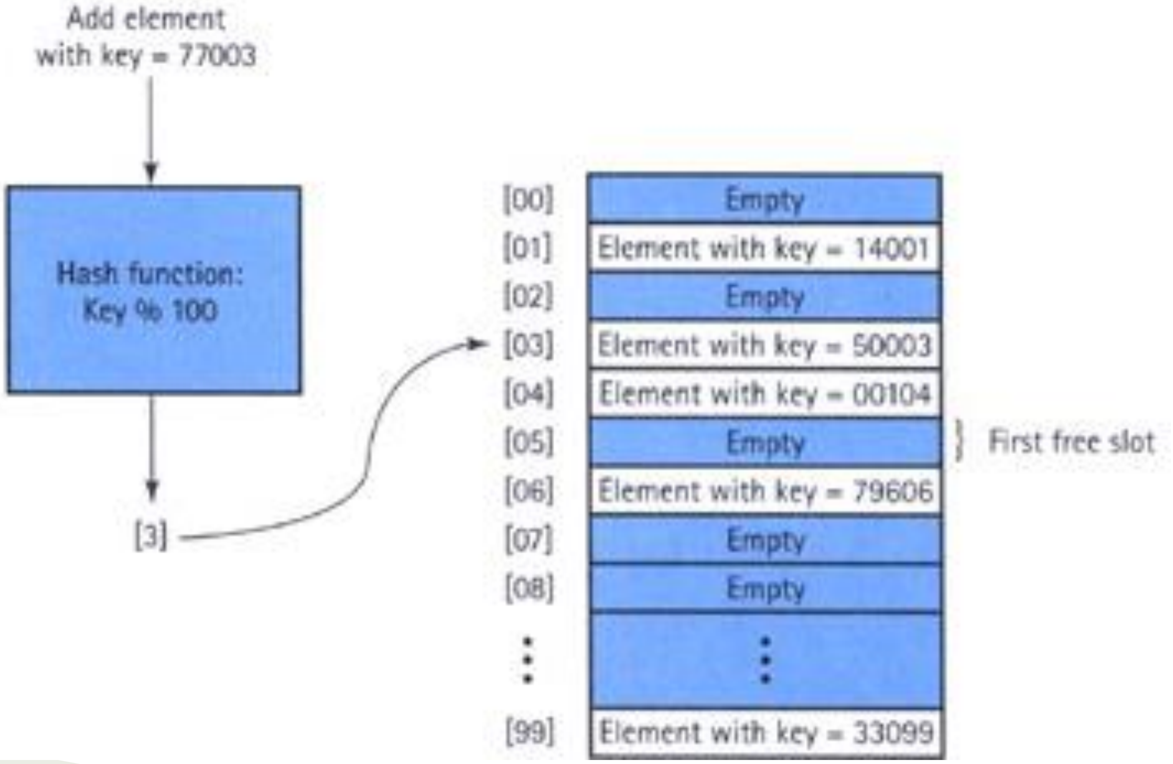
- number 01234 and ID number 91234 both "hash" to the same location: list.info [34]

- a good hash function *minimizes collisions* by spreading the elements uniformly throughout the array

Collision-handling algorithms

1. Linear Probing

- store the colliding element in the next available space.

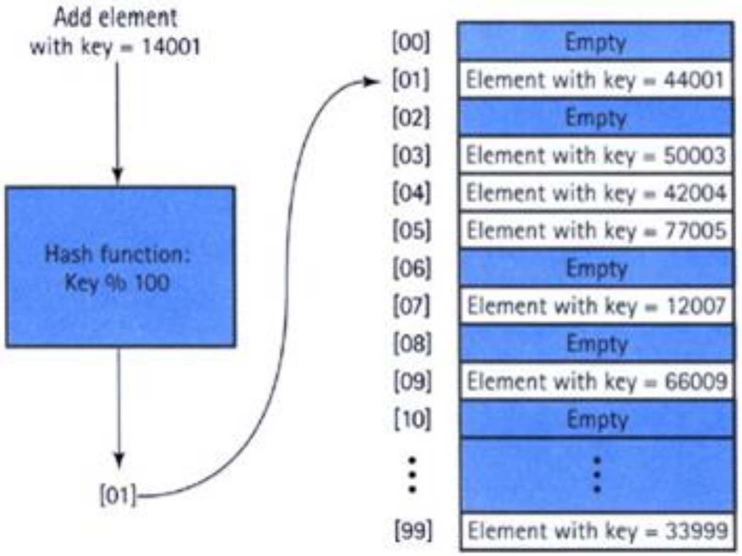

Add element with key = 77003
Hash function: Key % 100
[3]
[00] Empty
[01] Element with key = 14001
[02] Empty
[03] Element with key = 50003
[04] Element with key = 00104
[05] Empty
First free slot
[06] Element with key = 79606
[07] Empty
[08] Empty
[99] Element with key = 33099

Collision-handling algorithms

2. Rehashing

- Resolving a collision by computing a new hash location from a hash function that manipulates the original location*(HashValueOriginal + constant) % array_size*
  - Obs: the constant and the array-size must be relatively prime

(HashValue + 3) % 100

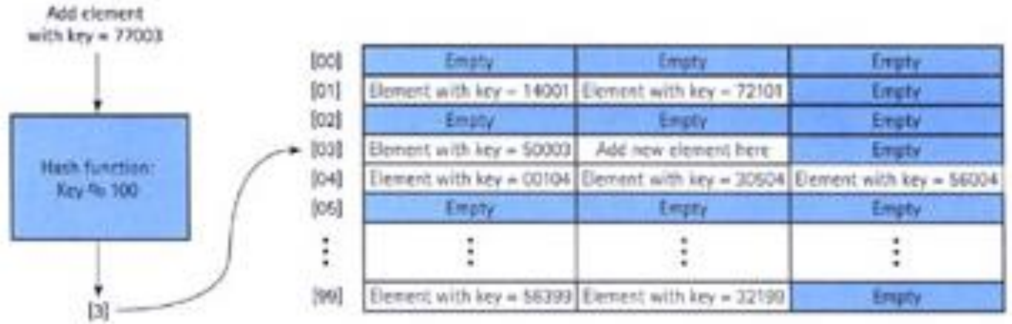


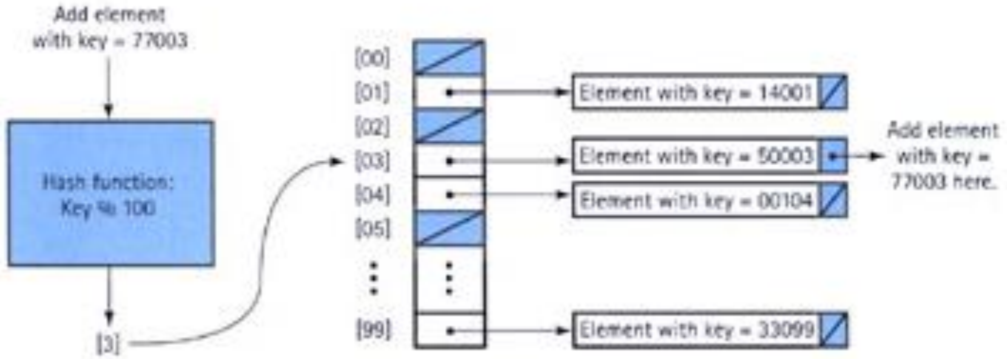
[00]	Empty
[01]	Element with key = 44001
[02]	Empty
[03]	Element with key = 50003
[04]	Element with key = 42004
[05]	Element with key = 77005
[06]	Empty
[07]	Element with key = 12007
[08]	Empty
[09]	Element with key = 66009
[10]	Empty
⋮	⋮
[99]	Element with key = 33999

Collision-handling algorithms

3. Buckets and Chaining

- bucket: a collection of elements associated with a particular hash location
- chain: a linked list of elements that share the same hash location





Exercise 1

- Use the following values:
- *66 47 87 90 126 140 145 153 177 285 393 395 467 566 620 735*
- Store the values into a hash table with 20 positions.

a) Use the division method of hashing and the linear probing method of resolving collisions.

b) Use rehashing as the method of collision resolution. Use key % tableSize as the hash function, and (key + 3) % tableSize as the rehash function

c) Store the values into a hash table with ten buckets, each containing three slots. If a bucket is full, use the next (sequential) bucket that contains a free slot.

Exercise 2

- Test the heaser hash.h in a .cpp file with the main function.

- Create a hash table which has keys of type char*. Use a hash function similar to this one:
  - for (int i = 0; i < strlen(key); i++)
  - hkey = (hkey * P + key[i]) % VMAX;

Exercise 3

- Find a hash function to convert numeric personal numbers into values between 1 and 10. Write a program to generate some random numeric personal numbers test your function.