



Software Engineering
Rochester Institute
of Technology

C: Hashmaps

SWEN-250

Personal Software Engineering



Data Structures

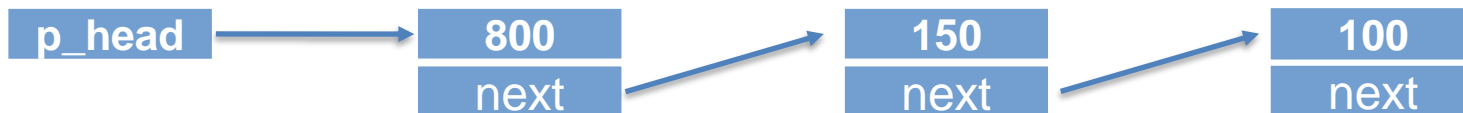
- Data Structures are ways of organizing information
 - So far, we have looked at:
 - Arrays
 - Linked Lists
 - (and structs for groups of data)
- Arrays and lists are sequential organizations of data

Recap

Array

| | | | | | | |
|-----|-----|-----|----|-----|-----|-----|
| 0 | 1 | 2 | 3 | 4 | 5 | ... |
| 800 | 150 | 100 | 25 | 952 | 216 | ... |

List



In each case, you organize data, but access in a linear fashion
- By position, or sequence

More flexibility: Hash(...)

- HashTables or HashMaps
 - Associative organization of data
 - Using a 'key' and 'value' pair
 - Fast access using keys (vs. index or sequence search)
 - Dynamic addition of data



Other considerations

- **Arrays:** Designed to statically allocate memory to store sequence of data where you can access them using an index in constant time and usually static arrays can't be extended or to add more elements
- **LinkedLists:** You can add elements dynamically and each element is pointing to the next one .. in order to access an element at index i , you have to linearly go through all elements before it in order to reach the needed element
- **HashTable:** A data structure that you can easily access elements in a fast way (using its hash value) and that you can dynamically add more data .. its also an abstraction on top of one or more data structures .. it depends on your needs which makes HashTable the perfect choice or any other data structure ..

Hash(Map | Table)

- Sometimes referred to as dictionary (but slightly different implementation)
- The general approach is:
 - Instead of links or indexes, we use a key
 - However, we need to convert the key into something usable as an unique index. That is the 'hash'
 - New Entry: Key \rightarrow Hash; $\text{table}[\text{hash}] = \text{value}$
 - Lookup: Key \rightarrow Hash; $\text{value} = \text{table}[\text{hash}]$



Dynamic sizing

- Hash tables gain speed from using arrays for indexing, but need to solve the problem of arrays being fixed size ... i.e. dynamic arrays
- Hash tables are generated at size N
- When you run out of slots, you dynamically create a new array (usually size $2N$) and rehash old elements into the new array
- Other topics (beyond the scope of our current work
 - Hash collisions (we'll cover one); hash algorithms ...



Visually (pseudo-code)

```
HashMap m;  
m.add("dog", 100);  
m.add("cat", 50);  
m.add("horse", 200);
```

| Key | Value |
|-------|-------|
| dog | 100 |
| cat | 50 |
| horse | 200 |

```
hashFunction("dog") => 2;  
hashFunction("cat") => 235343;  
hashFunction("horse") => 97543;
```

Very big array!!

The 'key' is used to
generate a number
as an index


| | |
|--------|-----|
| 1 | - |
| 2 | 100 |
| 3 | - |
| ... | - |
| 97543 | 200 |
| ... | - |
| 235343 | 50 |
| | - |
| | |



Visually (pseudo-code)

We can make this array size more reasonable by using the modulus operator!

```
HashMap m;  
m.add("dog", 100);  
m.add("cat", 50);  
m.add("horse", 200);
```

| | |
|---|--|
| 1 | - |
| 2 | 100 |
| 3 | 50  |
| 4 | - |
| 5 | - |

```
hashFunction("dog") => 2 % 5 = 2;  
hashFunction("cat") => 235343 % 5 = 3;  
hashFunction("horse") => 97543 % 5 = 3;
```



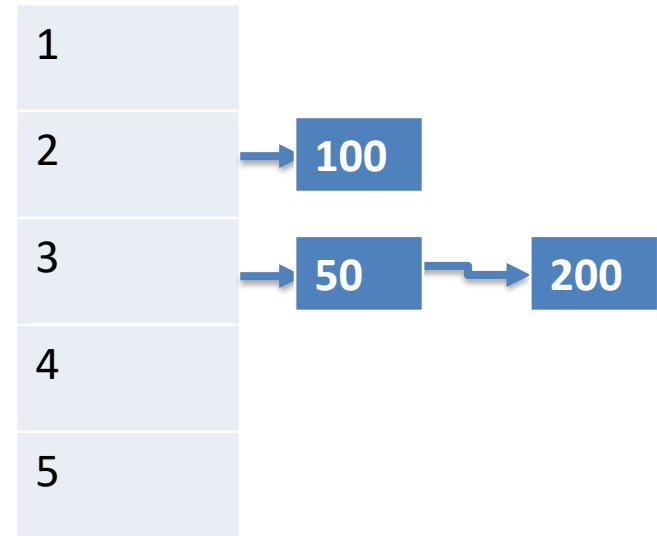
Collision
@ 3



Visually (pseudo-code)

```
HashMap m;  
m.add("dog", 100);  
m.add("cat", 50);  
m.add("horse", 200);
```

```
hashFunction("dog") => 2 % 5 = 2;  
hashFunction("cat") => 235343 % 5 = 3;  
hashFunction("horse") => 97543 % 5 = 3;
```



We have the hashes point to a LIST of values!

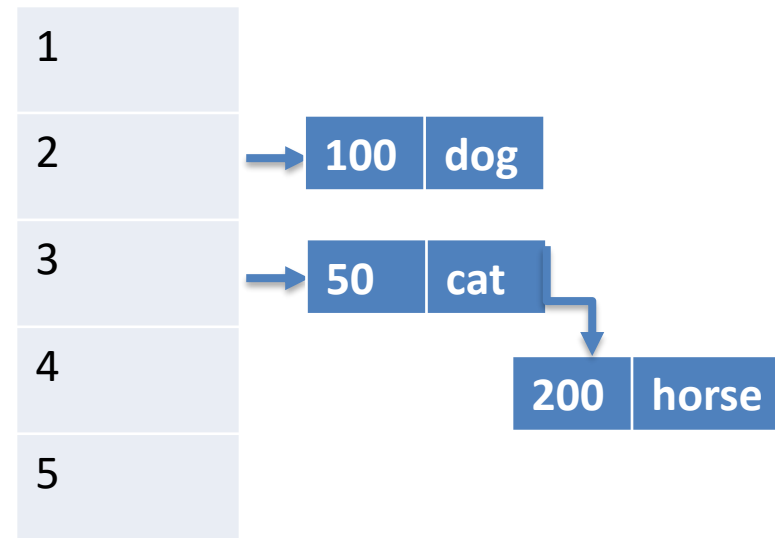
How do we know which item is the list is the right one?



Visually (pseudo-code)

```
HashMap m;  
m.add("dog", 100);  
m.add("cat", 50);  
m.add("horse", 200);
```

```
hashFunction("dog") => 2 % 5 = 2;  
hashFunction("cat") => 235343 % 5 = 3;  
hashFunction("horse") => 97543 % 5 = 3;
```



Each 'object' will hold the original key, value and reference to the next object

To find/ look up a key-value pair

- Hash the key
- Look up the first object using the hash and see if it has the same key
- If it does: Done. If not: Keep searching the list



Software Engineering
Rochester Institute
of Technology

ON TO THE ACTIVITY