On HashMaps and Implementations

Use case

Hash maps are the backbone of fast running programs. They are used to create caches, make searching really fast (for certain workloads faster than search trees), index databases for fast lookups and to create sets.

Hashes and Hashing functions

A hash function f maps a given input: $f(i) \rightarrow h$, i to a hash h. $f(i)$ has to always compute to h for the same i, otherwise the map would store values with the same key at different locations. To keep map access $O(1)$ and map insert $O(n)$, the hash function computes to an integer. This integer is then used to index into an the underlying array, instead of iterating a list or an array until the desired key is found.
Lets take a look at some common hashing applications: Java hashes strings by summing the characters of the string, while each is xored with the length minus the index of the character [1].

```
var s = "Hello World";
var h = 0;
for (int i = 0; i < s.length(); i++) {
    h += s.codePointAt(i) * 31
        ^ (s.length() - (i+1));
}
```

We will use a similar, but different algorithm for hashing our key strings: fnv-1a [2]. The key of fnva-1a is to start with a large basis for the hash, called the base, and modify it by xoring it with the current character and multiplying it with a prime number, thus we can create the first function of our naive implementation, `hash()`:

```
const size_t BASE = 0x811c9dc5;
const size_t PRIME = 0x01000193;
size_t hash(Map *m, char *str) {
    size_t initial = BASE;
    while(*str) {
        initial ^= *str++;
        initial *= PRIME;
    }
    return initial & (m->cap - 1);
}
```

The first things to notice, is the two constants required by fnva-1a, the parameter of the hash function of the `Map` type and the bitwise and in the return statement. The `m` parameter is used specifically in combination with the bitwise `&` to restrict the resulting hash to the size of the underlying array, thus eliminating out of bounds errors.

[1] String.hashCode()
[2] Fnv Hashing Wikipedia

Map Initialisation

```
typedef struct Map {
    size_t size;
    size_t cap;
    void **buckets;
} Map;
```

Pointer Insertion

Pointer Extraction

Excursion: Hash Collisions

Excursion: Performance