**CSC 221: DATA STRUCTURE (Hashing)**

### What is Hashing?

Hashing is an algorithm that calculates a fixed-size bit string value from a file. A file basically contains blocks of data. Hashing transforms this data into a far shorter fixed-length value or key which represents the original string. The hash value can be considered the distilled summary of everything within that file.

A good hashing algorithm would exhibit a property called the avalanche effect, where the resulting hash output would change significantly or entirely even when a single bit or byte of data within a file is changed. A hash function that does not do this is considered to have poor randomization, which would be easy to break by hackers.

A hash is usually a hexadecimal string of several characters. Hashing is also a unidirectional process so you can never work backwards to get back the original data.

A good hash algorithm should be complex enough such that it does not produce the same hash value from two different inputs. If it does, this is known as a hash collision. A hash algorithm can only be considered good and acceptable if it can offer a very low chance of collision.

### What are the benefits of Hashing?

One main use of hashing is to compare two files for equality. Without opening two document files to compare them word-for-word, the calculated hash values of these files will allow the owner to know immediately if they are different.

Hashing is also used to verify the integrity of a file after it has been transferred from one place to another, typically in a [file backup](https://www.2brightsparks.com/resources/articles/best-practices-for-data-backup.html)program like SyncBack. To ensure the transferred file is not corrupted, a user can compare the hash value of both files. If they are the same, then the transferred file is an identical copy.

In some situations, an encrypted file may be designed to never change the file size nor the last modification date and time (for example, [virtual drive](https://www.2brightsparks.com/resources/articles/backing-up-to-a-virtual-drive-using-syncBack-containers.html) container files). In such cases, it would be impossible to tell at a glance if two similar files are different or not, but the hash values would easily tell these files apart if they are different.

### Types of Hashing

There are many different types of hash algorithms such as RipeMD, Tiger, xxhash and more, but the most common type of hashing used for file integrity checks are MD5, SHA-2 and CRC32.

**MD5** - An MD5 hash function encodes a string of information and encodes it into a 128-bit fingerprint. MD5 is often used as a checksum to verify data integrity. However, due to its age, MD5 is also known to suffer from extensive hash collision vulnerabilities, but it’s still one of the most widely used algorithms in the world.

**SHA-2** – SHA-2, developed by the National Security Agency (NSA), is a cryptographic hash function. SHA-2 includes significant changes from its predecessor, SHA-1. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.

**CRC32** – A cyclic redundancy check (CRC) is an error-detecting code often used for detection of accidental changes to data. Encoding the same data string using CRC32 will always result in the same hash output, thus CRC32 is sometimes used as a hash algorithm for file integrity checks. These days, CRC32 is rarely used outside of [Zip](https://www.2brightsparks.com/resources/articles/data-compression.html) files and [FTP](https://www.2brightsparks.com/resources/articles/essential-ftp-features.html) servers.

**Application of Hash Tables:**

Some applications of Hash Tables are:

1. **Database System:** Specifically, those that are required efficient random access. Usually, database systems try to develop between two types of access methods: sequential and random. Hash Table is an integral part of efficient random access because they provide a way to locate data in a constant amount of time.
2. **Symbol Tables:** The tables utilized by compilers to maintain data about symbols from a program. Compilers access information about symbols frequently. Therefore, it is essential that symbol tables be implemented very efficiently.
3. **Data Dictionaries:** Data Structure that supports adding, deleting, and searching for data. Although the operation of hash tables and a data dictionary are similar, other Data Structures may be used to implement data dictionaries.
4. **Associative Arrays:** Associative Arrays consist of data arranged so that nth elements of one array correspond to the nth element of another. Associative Arrays are helpful for indexing a logical grouping of data by several key fields.