# Objectives

After the Lab, the student should be able to

- Implement different static hashing techniques.
- Compare between hashing techniques in space and time complexities.
- Decide which is a better hashing technique to use.

# Introduction

**Why Hashing?** The amount of Data is increasing dramatically and the old data structure and way of storing is not longer good enough. Suppose we have a very large data set stored sequentially in a file. The amount of time required to lookup an element in the file is either O(log n) or O( n) based on whether the file is sorted or not. If the file is sorted then a technique such as binary search can be used to search. Otherwise, the file must be searched linearly. Either case may not be desirable if we need to process a very large data set. Therefore we discuss a new technique called hashing that allows us to update and retrieve any entry in constant time O(1). The constant time or O(1) performance means, the amount of time to perform the operation does not depend on data size n[1].

In database, we are mainly concerned with data in files not the Hash Table data structure, However the idea is quite the same. Our database stores a set of records and each record has a unique key (it should y3any :)). To speed up the insertion, update and retrieval we save the file using a hashing function.

**[From Lecture Slides]** Hashing for disk files is called External Hashing. The file blocks are divided into M equal-sized buckets, numbered bucket0, bucket1, ..., bucketM-1. Typically, a bucket corresponds to one (or a fixed number of) disk block. One of the file fields is designated to be the hash key of the file. The record with hash key value K is stored in bucket i, where i=h(K), and h is the hashing function. E.g., h(k)= K mod M Search is very efficient on the hash key. Collisions occur when a new record hashes to a bucket that is already full.

There are numerous methods for collision resolution, including the following:

- Open addressing: Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found.
- Chaining: For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. In addition, a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location.
- Multiple hashing: The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.

So we are going to Implement those collision resolution Methods :)

[1] http://www.techtud.com/sites/default/files/public/share/Lecture17.pdf

# Requirements

1) Continue The insertion function in openAddressing.cpp (in Lab)
2) Implement Chaining resolution for collision. (at home)
3) Implement Multiple hashing resolution for collision (What is a good hashing function?) (at home)
4) Provide two different test cases to support each type of collision resolution (2 for openaddressing, 2 for chaining and 2 for Re-hashing) use the real no. of search/insert/delete query, different file size to store the same amount of data (same no. of records) . (at home)
5) Your analysis on the previous test cases (why a certain technique is better than the others)
6) Next Week: extendible and dynamic Hashing :)

# Considerations

- Code is written and tested under Linux using C/C++.
- In Order to compare different techniques: you need to keep the no. of records that a file can store constant, however you can modify the content of a record and provide explanation why you did that.
- Take care that you are dealing with files. Make sure you close the file when you finish. Also make sure to keep the data consistent between different runs of the program.
- It is not wise to write your program to read the whole document at once since usually files are too big (that wouldn't be external hashing anyway).
- The data in the stored files doesn't need to be readable by the human eye, but you need to figure out a way for your program to read it correctly.
- Cheating leads to **NEGATIVE GRADE** for both cheater and helper.
- Cheating from online resources is equivalently punished as the previous.
- Make sure your elearning account is working, you will deliver your (at home parts through it)

# Useful Functions:

pread
pwrite

[1] http://www.techtud.com/sites/default/files/public/share/Lecture17.pdf