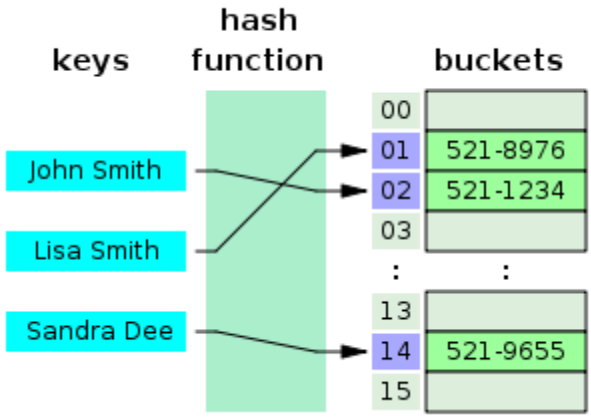
Lab 5: Hashtables and C++ Overview (Total 100 pts)

The data structures you have seen until now are perfect for dynamic allocation of unknown amount of data. However when it comes to a large amount of data the structures start to behave slowly and getting the desired information takes a lot of time. To solve this issue two alternatives exist, the first one is creating a search tree to speed up the process, which we have seen in class how to do.

The other alternative is to use a hash table. A hash table is a structure composed of arrays of all sorts of things. Each element in such array is actually a pair (key and value). Keys are indexes which range from 0 to n – 1, just like any other array. Values may be a single Object (ideally) or any other thing (more practical).

The key value, to which the Object gets mapped to, is calculated using a mathematical function known as a hash function which receives an object and associates an index (hash code) to it. Hash Collisions occur when two different objects have the same hash code, which is an undesired condition. When a hash collision occurs, multiple elements get mapped to the same position in the array. We can solve this by changing the hash function for a better one, rehashing by increasing the calculated hash by some offset, or simply using a data structure to hold the mapped values.



Example of a Hashtable

In the ideal case, a hash table is the perfect happy medium between an array and a linked list. Can expand easily as a linked list/binary search tree and search time is O(1). However, when we use data structures to hold our values we get a search time dependent of that structure; therefore, is best to go with a BST as values, that way our search time is O(log(n)). Alternatively, we can also place another Hashtable inside to achieve a complicated O(1) search time.

But how can we write a hash table in C? This course has been all in C, so even if the hash table we will use it in C++, we will still learn how to build a hash table in C:

Let's define a set of functions to use with our Hashtable (in Webcourses you will find these files implemented):

```
Hashtable* allocHT(uint elementSize, void (*releaser)(Object), int (*hasher)(Object));

void releaseHT(Hashtable* ht);

void addHT(Hashtable* ht, Object key, Object value);

Object getHT(Hashtable* ht, Object key);

void setHT(Hashtable* ht, Object key, Object newValue);

void removeHT(Hashtable* ht, Object key);

bool isEmptyHT(Hashtable* ht);

void clearHT(Hashtable* ht);

uint countKeysHT(Hashtable* ht);

uint countValuesHT(Hashtable* ht);

void keysHT(Hashtable* ht, Object* keys);

void valuesHT(Hashtable* ht, Object* values);

void printHT(FILE* out, Hashtable* ht);
```

Our first idea could be:

```
struct HTable {
        Object* keys;
        Object* values;
        …
} Hashtable;
```

But this is actually a **pretty bad solution**. Commonly, we like to have hash table and print all its entries. When this happens our memory performs bad in keeping copies of the frequently used information. For this is better to create an array of pairs (key + value).

```
struct HTPair {
        Object* key;
        BSTree* value;  //using this to handle duplicates
```

```
}

struct HTable {

        struct HTPair* entries;

        uint elementSize;

        void (*releaser)(Object);

        int (*hasher)(Object);

        int (*comparator)(Object, Object);

        …

} Hashtable;
```

TODO

Write a C++ class named Student that has a PID (uint), name (string), and academicYear (uint). Write also a class Course with a courseName (string), numberCredits (uint). In the end, write a C++ main program that creates two hash tables (map) of <string, Student> and <uint, vector<Course>>. **Only create getters, setters, and the operator<< for each class**. You will need the following libraries (plus any other that you need):

```
#include<vector>

#include<map>
```

Feel free to read at all the other data structures that C++ already has http://www.cplusplus.com/reference/stl/.

Write the following functions (in main.c):

```
//prompt for the data of a new student (read from keyboard) and add the new student to the map database, if it is not already there. You can assume that the name will be unique.

void createStudent(map<string, Student>);

//creates a new course with the information from keyboard, you can store all the courses in a vector

Course createCourse();

//add a course as one of the courses a student has taken

void addCourse(Student, Course);
```

//prints the list of students and all the courses that the student has taken

void printStudents();

Submit

Write for each class the .h (header) and .cpp (implementation) files. Also create a main.c function that tests the functions specified in the section above. Submit the 5 files.

Grading

- (10 pts) Author header comment and file comments
- (30 pts) Implementation of class Student
    - (15 pts) Header file
        - (5 pts) Class definition
        - (5 pts) Private fields
        - (5 pts) Public methods
    - (15 pts) Implementation
        - (5 pts) Getters and Setters
        - (5 pts) Constructors and Destructor (since the classes don't have any pointers just reset the values to 0 or "")
        - (5 pts) friend operator<<
- (30 pts) Implementation of class Course
    - (15 pts) Header file
        - (5 pts) Class definition
        - (5 pts) Private fields
        - (5 pts) Public methods
    - (15 pts) Implementation
        - (5 pts) Getters and Setters
        - (5 pts) Constructors and Destructor (since the classes don't have any pointers just reset the values to 0 or "")
        - (5 pts) friend operator<<
- (5 pts) Map for <string, Student>, where key is a person's name and Student is a student with that name.
- (5 pts) Map for <int, vector<Course>>, where key is the PID of a student and vector<Course> is a dynamic array of Courses. You could replace this with another map (e.g., map<string, Course>), where it maps from courseName to a Course.
- (20 pts) Main.c functions
    - (5 pts) void createStudent(map<string, Student>);
    - (5 pts) Course createCourse();
    - (5 pts) void addCourse(Student, Course);
    - (5 pts) void printStudents();