

National University of Computer and Emerging Sciences



**Laboratory Manual #11**  
*for*  
**Data Structures Lab**  
**(CL 2001)**

Department of Computer Science  
FAST-NU, Lahore, Pakistan

## Introduction

### Objectives

After performing this lab, students shall be able to:

1. Hashing

### Question 1: (HashMap with Linear Probing)

Implement a HashItem struct which represents an item in a hash array.

```
template <class v>
struct HashItem
{
    int key;
    v value;
    short status;
    list <v> ll; //STL list
};
```

status variable can have 0, 1 or 2. 0 means empty, 1 means deleted, 2 means occupied. Status variable will be used by get and delete methods of HashMaps implemented in the next questions. The default value assigned to a HashItem is 0 (empty).

Now implement a HashMap class whose basic definition is as follows (You can add any helping member variables and methods):

```
template <class v>
class HashMap
{
private:
    HashItem<v>* hashArray;
    int capacity;
    int currentElements;
    virtual int getNextCandidateIndex(int key, int i)

public:
    HashMap();
    HashMap(int const capacity);
    void insert(int const key, v const value);
    bool deleteKey(k const key) const;
    v* get(k const key) const;
    ~HashMap();
};
```

1. HashMap(): constructor assigns a capacity of 10 to hashArray.
2. HashMap(int const capacity): an overloaded constructor that assigns the capacity of given capacity to hashArray. If capacity is less than 1 return error via assert(capacity>1)
3. void insert(int const key, v const value):
  - a. The insert method inserts the value at its appropriate location. Find the first candidate index of the key using:  
 $index = key \bmod capacity$

- b. To resolve hash collision, it will use the function getNextCandidateIndex(key, i) to get the next candidate index. If the candidate index also has collision, then getNextCandidateIndex will be called again with an increment of 1 (**hashing using linear probing**). getNextCandidateIndex will be continued to call until we find a valid index. Initially i will be 1.
  - c. If the loadFactor becomes 0.75, then it will call the doubleCapacity method to double the capacity of array and rehash the existing items into the new array.
4. `void doubleCapacity()`: A private method which doubles the capacity of hash array and rehashes the existing items. Use getNextCandidateIndex method to resolve collision.
  5. `virtual int getNextCandidateIndex(int key, int i)`: a private and virtual method that uses linear probing to return the next candidate index for storing the item containing key k. Linear probing means that it will simply add i to the hash value of key. This method does not check whether the candidate index has collision or not.
  6. `bool deleteKey(const key) const`: this method deletes the given key. It returns true if the key was found. If the key was not found it returns false. When the key is found, simply set the status of the hashitem containing the key to deleted (value of 1). It also uses status variable to search for the key intelligently.
  7. `V* get(const key) const`: this method returns a pointer to the corresponding value of the key. If the key is not found, it returns nullptr. It also uses status variable to search for the key intelligently.
  8. `~HashMap()`: destructor

### Question 2: (HashMap using Quadratic Probing)

Create a class **QHashMap** which inherits the HashMap class implemented in question 1.

Override the getNextCandidateIndex(int key, int i) method so that it performs quadratic probing, i.e., add the square of i to the hash value of key.

### Question 3: (HashMap using Double Hashing)

Create a class **DHashMap** which inherits the HashMap class implemented in question 1.

Override the getNextCandidateIndex(int key, int i) method so that it performs double hashing and returns the candidate index. Double hashing will be performed as follows:

*first\_value = key mod capacity*

*second\_value = (PRIME - (key mod PRIME))* (PRIME is any prime number.)

*candidate index = (first\_value + i\*second\_value) mod capacity*

### Question 4: (HashMap using with chaining)

Create a class **CHashMap** which inherits the HashMap class implemented in question 1.

Override the getNextCandidateIndex(int key, int i) method so that in case of collision it inserts the value in the STL list l1 corresponding to that key. It will return the key.

### Question 5:

Create a global function populateHash which is passed a filename as parameter and a HashMap object by pointer `void populateHash(string filename, HashMap<string> *hash)`.

The function reads <id, name> pairs line by line and populates the hash map with those pairs using the insert function. The key is id.

**Question 6:**

Now run the following main program for all including **HashMap**, **QHashMap**, **DHashMap** and **CHashMap** and compare the performance of each and suggest which hashing technique is better in our case.

```
#include <iostream>
using namespace std;
#include <string>

int main()
{
    HashMap<string> *map;
    map=new HashMap<string>; // map=new QHashMap<string>;you will use polymorphism to run
    for all the classes.
    populateHash("students.txt", map);
    cout<<*map->get(9);
    map->deleteKey(9);
    assert(map->get(9)==nullptr);
    delete map;

    map=new QHashMap<string>;
    populateHash("students.txt", map);
    cout<<*map->get(98);
    map->deleteKey(98);
    assert(map->get(98)==nullptr);
    delete map;

    map=new DHashMap<string>;
    populateHash("students.txt", map);
    cout<<*map->get(101);
    map->deleteKey(101);
    assert(map->get(101)==nullptr);
    delete map;

    return 0;
}
```