# Department of Computing

# School of Electrical Engineering and Computer Science

**CS - 250: Data Structure and Algorithms**

**Class: BSCS 10AB**

**Lab 11 : Hash Tables**

**Date: 21th December, 2021**

**Time: 10:00 am – 12:50 pm   
&  
 02:00 pm – 4:50 pm**

**Mashal Ashfaque**

**337203**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# 

# Lab 11 : Hash Tables

**Introduction**

This lab is based on the hashing with separate chaining and hashing with open addressing using linear probing , Quadratic probing and double hashing.

**Objectives**

Objective of this lab is to make students familiar with hash tables.

**Tools/Software Requirement**

Visual Studio 2012 or gcc or g++

**Tasks**

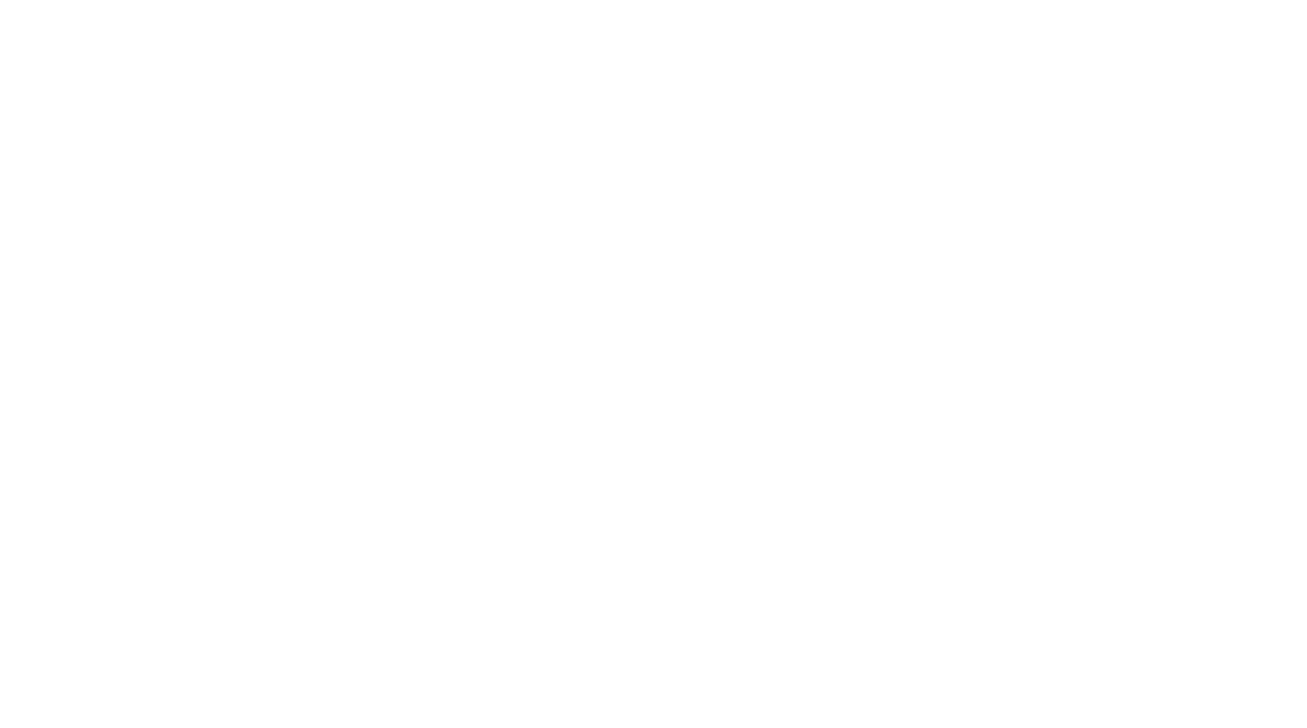
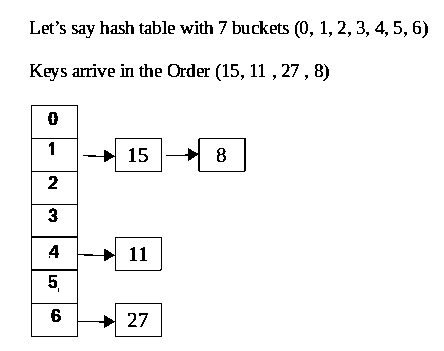
**Part A :** **Hashing With Seperate Chaining**

To write a C program to implement the concept of hashing using separate chaining.

**Description**

In [hashing](http://www.geeksforgeeks.org/hashing-data-structure/) there is a hash function that maps keys to some values. But these hashing function may lead to collision that is two or more keys are mapped to same value. Chain hashing resolves collisions. The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Let’s create a hash function, such that our hash table has ‘N’ number of buckets. To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function.



Example: hashIndex = key % noOfBuckets

Insert: Move to the bucket corresponds to the above calculated hash index and insert the new node at the end of the list.

**ALGORITHM**

Step 1:Start

Step 2: Create Table size Step 3: Create hash function

Step 4: To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function.

Step 5: Display hash entry.   
Step 6: Stop

**VIVA (PRE & POST LAB) QUESTIONS**

* 1. If several elements are competing for the same bucket in the hash table, what is it called?
  2. How to insert a node in hash table?
  3. What is sizeof()function?
  4. How to delete a node from hash table.

# Part B: Hashing With Open Addressing Using Linear Probing, Quadratic Probing And Double Hashing

To write a C program to implement the concept of hashing using the following open addressing based collision resolution techniques: a) linear probing, b) quadratic hashing, and c) double hashing.

**Description**

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored within the hash table. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data, if needed).

**Insert(k):** Keep probing until an empty slot is found. Once an empty slot is found, insert k. **Search(k):** Keep probing until slot’s key doesn’t become equal to k or an empty slot is reached.

**Delete(k):** Delete operation is interesting. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as “deleted”.

Insert can insert an item in a deleted slot, but the search doesn’t stop at a deleted slot.

Open Addressing is done following ways:

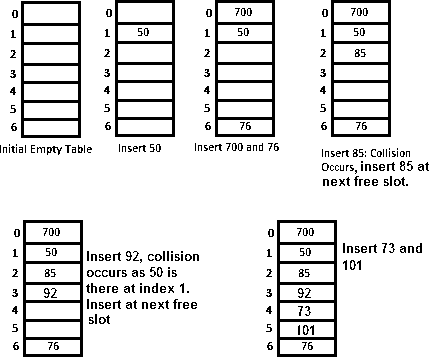
***a)* Linear Probing:** In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also. let **hash(x)** be the slot index computed using hash function and **S** be the table size

If slot hash(x) % S is full, then we try (hash(x) + 1) % S

If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S

If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S

Let us consider a simple hash function as “key mod 7” and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



**Clustering:** The main problem with linear probing is clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search an element.

***b) Quadratic Probing :*** We look for i2‘th slot in i’th iteration.

let hash(x) be the slot index computed using hash function.

If slot hash(x) % S is full, then we try (hash(x) + 1\*1) %S

If (hash(x) + 1\*1) % S is also full, then we try (hash(x) + 2\*2) % S

If (hash(x) + 2\*2) % S is also full, then we try (hash(x) + 3\*3) % S

***c)*** [***Double Hashing***](https://www.geeksforgeeks.org/double-hashing/)***:*** we use another hash function hash2(x) and look for i\*hash2(x) slot in i’th rotation.

let hash(x) be the slot index computed using hash function.

If slot hash(x) % S is full, then we try (hash(x) + 1\*hash2(x)) % S

If (hash(x) + 1\*hash2(x)) % S is also full, then we try (hash(x) + 2\*hash2(x)) % S   
 If (hash(x) + 2\*hash2(x)) % S is also full, then we try (hash(x) + 3\*hash2(x)) % S

**ALGORITHM**

Step 1: Start

Step 2: Create hash table and hash function

Step 3: Assigning INT\_MIN indicates that cell is empty

Step4: INT\_MIN and INT\_MAX indicates that cell is empty. So if cell is empty loop will break and goto bottom of the loop to insert element. If table is full we should break, if not check this, loop will go to infinite loop.

Step 5: If the hash element is empty, the deletion has immediately failed.

Otherwise, check for a match between the target and data key

* If there is a match, delete the hash element (and set the flag)
* If there is no match, probe the table until either:
  + An match is found between the target key and the data's key; the data can be deleted, and the deleted flag set.
  + A completely empty hash element is found

Step 6: If the hash element is empty, the search has immediately failed.

Otherwise, check for a match between the search and data key

* If there is a match, return the data.
* If there is no match, probe the table until either:
  + An match is found between the search and data key
  + A completely empty hash element is found.

Step 7: Stop

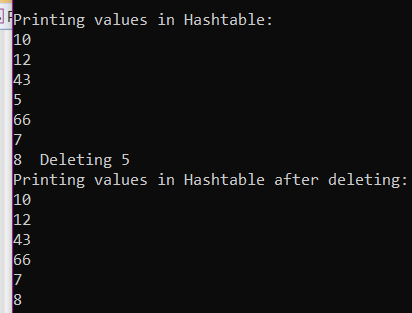
**Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. The name of word document should follow this format. i.e. **YourFullName(reg)\_Lab#.** This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.

**Linear Probing**

**Code:**

#include<iostream>

#include<limits>

using namespace std;

class Hash

{

int size;

int\* arr;

public:

Hash(int n) // constructor to store size of hashtable

{

this->size = n;

arr = new int[n];

for (int i = 0; i < n; i++)

arr[i] = INT\_MIN;

}

int Linearhashing(int x)

{

return(x % size); //hashfunction

}

void Insert\_LinearProbing(int key)

{

int index = Linearhashing(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i) % size] == INT\_MIN) //to check if cell is empty or not.

{

arr[(index + i) % size] = key;

return;

}

}

}

void delete\_Linear(int key)

{

int index = Linearhashing(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i) % size] == key)

{

cout << " Deleting " << arr[(index + i) % size];

arr[(index + i) % size] = INT\_MIN;

}

}

}

void Search\_Linear(int key)

{

int index = Linearhashing(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i) % size] == key)

{

cout << "Value has been found at index " << i + 1;

return;

}

}

cout << "Value not found";

}

void print\_Linear()

{

for (int i = 0; i < size; i++)

{

if (arr[i] != INT\_MIN) //to check if the slot is empty or not in hashtable

{

cout<<endl << arr[i] << " ";

}

}

}

};

int main()

{

int arr[] = { 10,5,8,7,66,43,12 };

int n = sizeof(arr) / sizeof(arr[0]);

Hash h1(10);

//insert into hashtable

for (int i = 0; i < n; i++)

h1.Insert\_LinearProbing(arr[i]);

cout << "\nPrinting values in Hashtable:";

h1.print\_Linear();

h1.delete\_Linear(5);

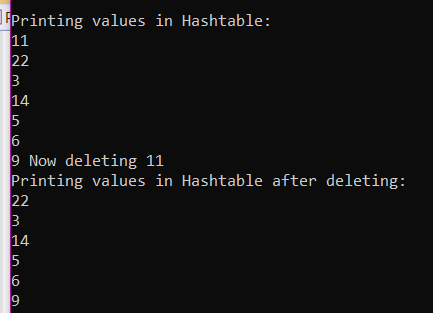
cout << "\nPrinting values in Hashtable after deleting: ";

h1.print\_Linear();

// h1.Search\_Linear(12);

}

**Quadratic Probing**



**Code**

#include<iostream>

#include<limits>

using namespace std;

class Hashing

{

int size;

int\* arr;

public:

Hashing(int n) // constructor to store size of hashtable

{

this->size = n;

arr = new int[n];

for (int i = 0; i < n; i++)

arr[i] = INT\_MIN;

}

int Quadratic\_Probing(int x)

{

return(x % size);

}

void insert\_quadratic(int key)

{

int index = Quadratic\_Probing(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i \* i) % size] == INT\_MIN) //to check if empty or not.

{

arr[(index + i \* i) % size] = key;

return;

}

}

}

void delete\_quadratic(int key)

{

int index = Quadratic\_Probing(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i \* i) % size] == key)

{

cout << "Now deleting " << arr[(index + i \* i) % size];

arr[(index + i \* i) % size] = INT\_MIN;

}

}

}

void Search\_quadratic(int key)

{

int index = Quadratic\_Probing(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i \* i) % size] == key)

{

cout << "Value has been found at index " << i + 1;

return;

}

}

cout << "Value not found";

}

void print\_quadratic()

{

for (int i = 0; i < size; i++)

{

if (arr[i] != INT\_MIN) //to check if the slot is empty or not in hashtable

cout << endl << arr[i] << " ";

}

}

};

int main()

{

int arr[] = { 22,5,9,3,6,11,14 };

int n = sizeof(arr) / sizeof(arr[0]);

Hashing h1(10);

//insert into hashtable

for (int i = 0; i < n; i++)

h1.insert\_quadratic(arr[i]);

cout << "\nPrinting values in Hashtable:";

h1.print\_quadratic();

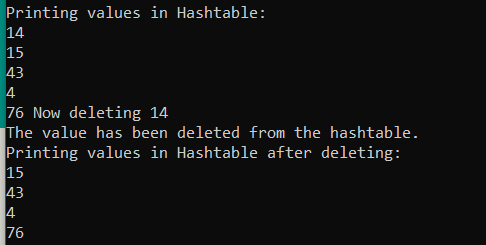
h1.delete\_quadratic(11);

cout << "\nPrinting values in Hashtable after deleting: ";

h1.print\_quadratic();

}

**Double Quadratic Probing**



#include<iostream>

#include<limits>

//#include<bits/stdc++.h>

using namespace std;

class Hash

{

int size;

int\* arr;

public:

Hash(int n) // constructor to store size of hashtable

{

this->size = n;

arr = new int[n];

for (int i = 0; i < n; i++)

arr[i] = INT\_MIN;

}

int hashdouble(int x)

{

return(x % size);

}

void insert\_double(int key)

{

int index = hashdouble(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i \* hashdouble(key)) % size] == INT\_MIN) //to check if empty or not.

{

arr[(index + i \* hashdouble(key)) % size] = key;

return;

}

}

}

void delete\_double(int key)

{

int index = hashdouble(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i \* hashdouble(key)) % size] == key)

{

cout << "Now deleting " << arr[(index + i \* i) % size];

arr[(index + i \* hashdouble(key)) % size] = INT\_MIN;

cout << "\nThe value has been deleted from the hashtable.";

}

}

}

void Search\_double(int key)

{

int index = hashdouble(key);

for (int i = 0; i < size; i++)

{

if (arr[(index + i \* hashdouble(key)) % size] == key)

{

cout << "Value has been found as index " << i + 1;

return;

}

}

cout << "Value not found";

}

void print\_double()

{

for (int i = 0; i < size; i++)

{

if (arr[i] != INT\_MIN) //to check if the slot is empty or not in hashtable

cout << "\n" << arr[i] << " ";

}

}

};

int main()

{

int arr[] = { 15,14,76,4,43 };

int n = sizeof(arr) / sizeof(arr[0]);

Hash h1(7);

//insert into hashtable

for (int i = 0; i < n; i++)

h1.insert\_double(arr[i]);

cout << "\nPrinting values in Hashtable:";

h1.print\_double();

h1.delete\_double(14);

cout << "\nPrinting values in Hashtable after deleting: ";

h1.print\_double();

}