Name: **Muhammad Anas Baig**

Enrollment No.: **01-134152-037**

Section: **BS(CS)-4A**

LAB-JOURNAL-9

Exercise 1:

Implement the following functions and insert a given set of integers into a hash table.

```
//Hash Function to generate the index
int hash(int key) ;

//A function that accepts the hash table and key to be inserted
and inserts the "key" at appropriate location in the table. Use
linear probing to resolve collisions. The returned values is the
index at which the key is inserted.
int linear_probing(int HashTable[], int key) ;


//A function that inserts values in the table and resolves
collisions using quardatice probing.
```

```
int quardatic_probing(int HashTable[], int key) ;
```

HINTS :

```
Quardatic probing can be implemented like :
for (i = 0 ; i% MAX != pos ; i++)
   pos = (pos + i * i) % MAX ;
```

In the main program, take a number of values from the user and insert them in the hash table using linear probing. Insert the same values in another hash table using quardatic probing.

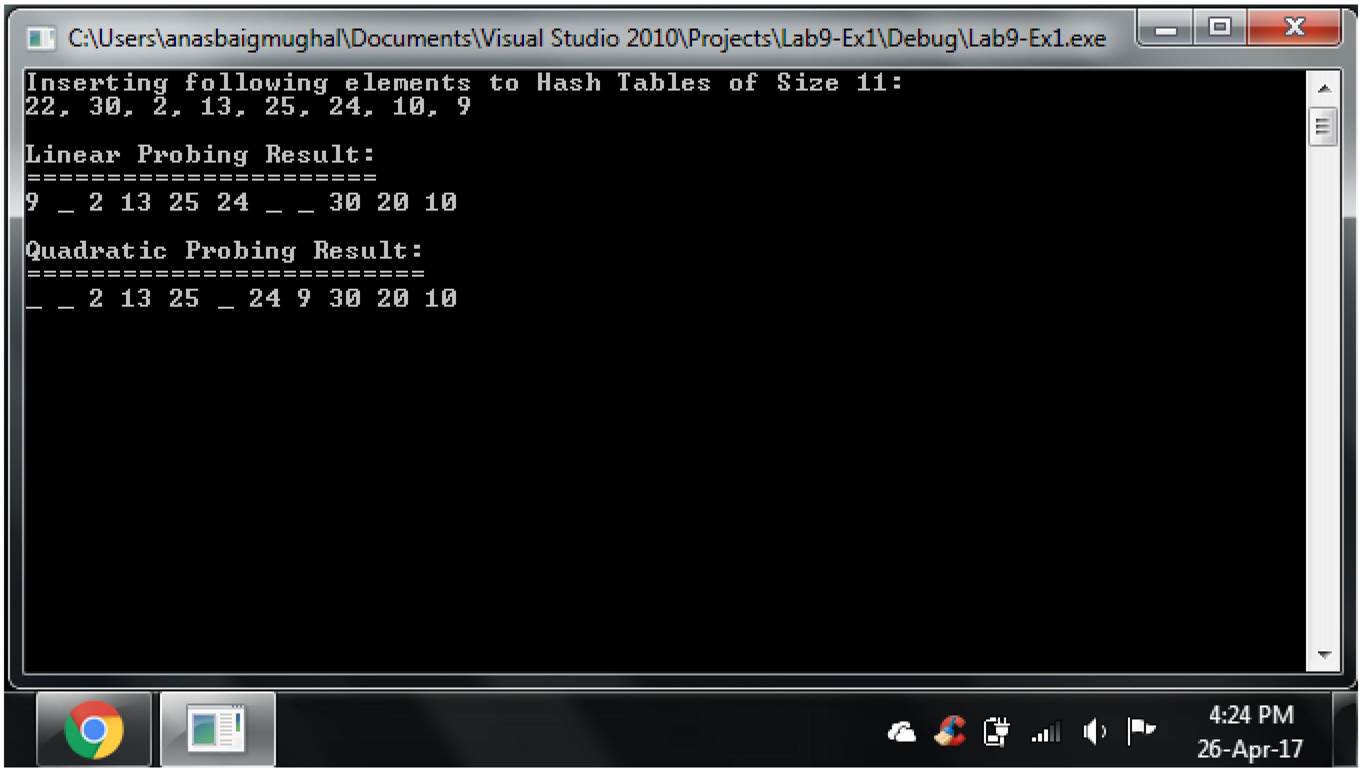
Solution:

```
#include "conio.h"
#include <iostream>
const int size=11;
using namespace std;

int _hash(int key)
{
    return (key%size);
};

int linearProbing( int hashTable[], int key )
```

```cpp
{
    bool spaceFound = false; //flag is set if empty location is found in the remaining part of array or not
    for( int i=0; ( _hash( key ) + i )<size; i++ ) //checks for the empty location in the remaining half of the array
    {
        if( hashTable[ _hash( key ) + i ] == -1 ) //if empty location is found
        {
            hashTable[ _hash( key ) + i ] = key;
            spaceFound = true; //sets flag to true i.e. empty location found so no need to further check
            return( _hash( key ) + i ); //exits from loop
        }
    }
    if( spaceFound != true  ) //if empty location is not found in the remaining half of the array
    {
        for( int i=0; i< _hash(key); i++ ) //checks for the empty location in the previous half of the array
        {
            if( hashTable[ i ] ==  -1 ) //if empty location is found
            {
                hashTable[ i ] = key;
                spaceFound = true; //sets flag to true i.e. empty location found so no need to further check
                return (i);
            }
        }
    }
    if( spaceFound == false )
    {
        cout<<"ARRAY OVERFLOW!!! No space to insert new data."<<endl;
        return (-1);
    }
};

int quadraticProbing( int hashTable[], int key )
{
    int j = 0;
    bool spaceFound = false; //flag is set if empty location is found in the remaining part of array or not
    for( int i=0; ( _hash( key ) + (i*i) )<size; i++ ) //checks for the empty location in the remaining half of the array
    {
        if( hashTable[ _hash( key ) + (i*i) ] == -1 ) //if empty location is found
        {
            hashTable[ _hash( key ) + (i*i) ] = key;
            spaceFound = true; //sets flag to true i.e. empty location found so no need to further check
            return( _hash( key ) + i ); //exits from loop
        }
        j++;
    }
    if( spaceFound != true  ) //if empty location is not found in the remaining half of the array
    {
        for( int i=j; _hash(_hash( key )+(i*i)) < _hash(key); i++ ) //checks for the empty location in the previous half of the array
        {
            if( hashTable[ _hash(_hash( key )+(i*i)) ] ==  -1 ) //if empty location is found
            {
                hashTable[ _hash(_hash( key )+(i*i)) ] = key;
                spaceFound = true; //sets flag to true i.e. empty location found so no need to further check
                return (i);
            }
        }
    }
    if( spaceFound == false )
    {
        cout<<"ARRAY OVERFLOW!!! No space to insert new data."<<endl;
        return (-1);
    }
}
```

```

void main()
{
    int a[size];
    int b[size];

    for( int i=0; i<size; i++ )
    {
        a[i] = -1;
        b[i] = -1;
    }

    cout<<"Inserting following elements to Hash Tables of Size 11:"<<endl;
    cout<<"22, 30, 2, 13, 25, 24, 10, 9"<<endl;
    linearProbing( a, 20 );
    linearProbing( a, 30 );
    linearProbing( a, 2 );
    linearProbing( a, 13 );
    linearProbing( a, 25 );
    linearProbing( a, 24 );
    linearProbing( a, 10 );
    linearProbing( a, 9 );

    quadraticProbing( b, 20);
    quadraticProbing( b, 30);
    quadraticProbing( b, 2);
        quadraticProbing( b, 13);
        quadraticProbing( b, 25);
        quadraticProbing( b, 24);
        quadraticProbing( b, 10);
        quadraticProbing( b, 9);

        cout<<endl<<"Linear Probing Result:"<<endl;
        cout<<"======================"<<endl;
        for( int i=0; i<size; i++ )
        {
            if( a[i] == -1 )
            {
                cout<<"_ ";
            }
            else
            {
                cout<<a[i]<<" ";
            }
        }
        cout<<endl;
        cout<<endl<<"Quadratic Probing Result:"<<endl;
        cout<<"=========================="<<endl;
        for( int i=0; i<size; i++ )
        {
            if( b[i] == -1 )
            {
                cout<<"_ ";
            }
            else
            {
                cout<<b[i]<<" ";
            }
        }
        getch();
    }
```

Output:



C:\Users\anasbaigmughal\Documents\Visual Studio 2010\Projects\Lab9-Ex1\Debug\Lab9-Ex1.exe
4:24 PM
26-Apr-17

```
Inserting following elements to Hash Tables of Size 11:
22, 30, 2, 13, 25, 24, 10, 9

Linear Probing Result:
======================
9 _ 2 13 25 24 _ _ 30 20 10

Quadratic Probing Result:
=========================
_ _ 2 13 25 _ 24 9 30 20 10
```

Exercise 2:

Consider the following class to store integers in nodes of a list.

```
class Node{
    public :
         int key ;
         Node * next ;
} ;
```

And the following hash table :

```
Node * hashtable[MAX]  ;
```

Implement the following functions.

```
//Hash function that generate hash index
     int hashfunction(int key);

//Intialize the array of pointers to NULL
     void initialize();

// Insert a value in the hash table; You need to create a node,
insert the value in the node and place the node at appropriate
location.
     void insert(int k);

// Display the complete data in the hash table
     void display();
```

Solution:

Node.h File:

```
#pragma once
class node
{
public:
    int key;
    node *next;
public:
    node(void);
};
```

Node.cpp File:

```
#include "node.h"

node::node(void)
```

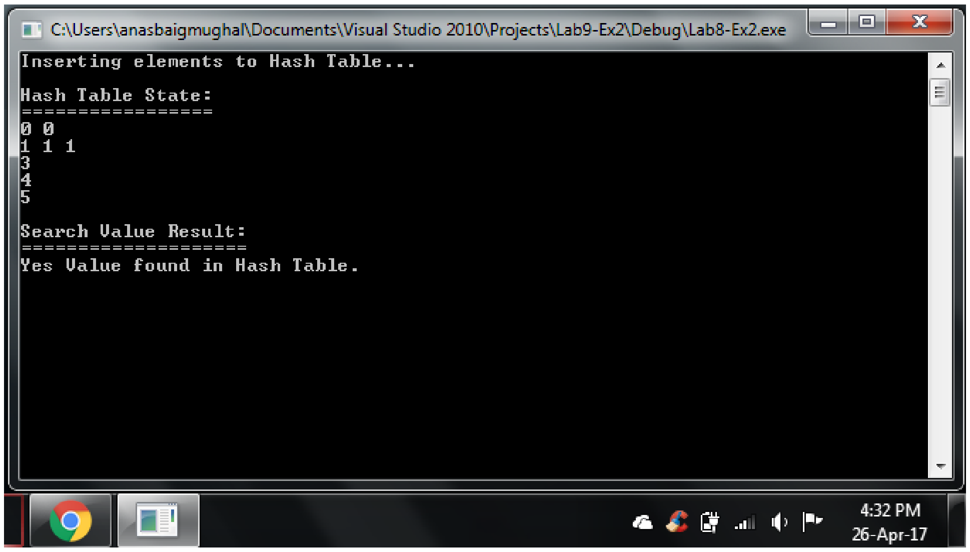
```
{
}
```

Main.cpp File:

```
#include "node.h"
#include "conio.h"
#include <iostream>
const int size=11;
using namespace std;

node *hashTable[11];

int hashFunction( int key )
{
    return ( key%size );
}

void initialize()
{
    for( int i=0; i<size; i++ )
    {
        hashTable[i] = NULL;
    }
}

void insert( int key )
{
    node *temp = hashTable[ hashFunction( key ) ];

    node *ptr = new node;
    ptr->key = 0;
    ptr->next = NULL;

    ptr->next = temp;

    ptr->key = key;

    hashTable[ hashFunction( key ) ] = ptr; //inserting new head
}

void display()
{
    for( int i=0; i<11; i++ )
    {
        node *temp = hashTable[i];
        if(hashTable[i] != NULL) //if not empty list
        {
            while(temp != NULL)
            {
                cout<<temp->key<<" ";
                temp= temp->next;
            }
            cout<<endl;
        }
    }
};

void search( int v )
{
    int index = hashFunction( v );
    bool valueFound = false;
    if(index < 11)
    {
```

```cpp
        node *temp = hashTable[index];
        while( temp != NULL )
        {
            if( temp->key == v )
            {
                cout<<"Yes Value found in Hash Table."<<endl;
                valueFound = true;
                return;
            }
            temp = temp->next;
        }
    }
    if( valueFound != true )
    {
        cout<<"SORRY!!! Value not found in Hash Table."<<endl;
    }
}

void main()
{
    initialize();
    cout<<"Inserting elements to Hash Table..."<<endl<<endl;

    insert(0);
    insert(0);
    insert(1);
    insert(1);
    insert(1);
    insert(3);
    insert(4);
    insert(5);

    cout<<"Hash Table State:"<<endl;
    cout<<"================="<<endl;
    display();

    cout<<endl;
    cout<<"Search Value Result:"<<endl;
    cout<<"===================="<<endl;
    search(1);
           getch();
       }
```

Output:


C:\Users\anasbaigmughal\Documents\Visual Studio 2010\Projects\Lab9-Ex2\Debug\Lab8-Ex2.exe

```
Inserting elements to Hash Table...

Hash Table State:
=================
0 0
1 1 1
3
4
5

Search Value Result:
====================
Yes Value found in Hash Table.
```