**Exercise 1:** Generate n random numbers (using rand%1000) and store them is a hash table.The value of n should be asked from the user at runtime and decide the size of array to use accordingly (hint: big prime number).

Please use the modulus function (%) as the hash function and **Open addressing (or linear probing)** as the Overflow handling technique. You have to implement the following ADT of Hashtable:

#ifndef HSH\_H

#define HSH\_H

template<class DT>

class Hashtable

{

public:

Hashtable(int size);

bool store(DT key);

bool search(DT key);

private:

int size;

DT\* arr;

};

#endif

Once all the n numbers have been stored in this hashtable, search for a particular number entered by the user and note the time it takes to carry out the search. If the number is found (or not) let the user know accordingly.

**SAMPLE SEARCH CODE**

template<class DT>

bool Hashtable<DT>::search\_in\_hashtable(DT identifier)

{

bool success=true;

int location = identifier%size;

int j;

for( j=location; arr[j]!=identifier; )

{

j=(j+1)%size;

if(j==location || arr[j]==-1)

{ success=false;//doesn't exist

break;

}

}

if(success)

{

cout<<"found at "<<j<<endl;

}

return success;

}

You may test your code using the following client

#include<iostream>

#include<Windows.h>

#include "Hashtable.h"

#include "Hashtable.cpp"

using namespace std;

int main()

{

int num\_of\_identifiers;

cout<<"Enter maximum number of keys that need to be stored in the hashtable: ";

cin>>num\_of\_identifiers;

int size\_hashtable;

cout<<"Enter size of hashtable needed to store these many identifiers (hint: use prime number): ";

cin>>size\_hashtable;

//create a hashtable of this size

Hashtable<int>\* ht=new Hashtable<int>(size\_hashtable);

for (int i=0; i<num\_of\_identifiers; i++)

{

int key=rand()%1000;

bool was\_stored = ht->store(key);

if(!was\_stored)

cout<<key<<" could not be stored as it already exists or table is full"<<endl;

}

int find\_key;

cout<<"Enter the key to search for "<<endl;

cin>>find\_key;

DWORD start, end;//measure time

start= GetTickCount();

bool found= ht->search(find\_key);

end= GetTickCount();

double cpu\_time\_used = end - start;

if(found)

cout<<"it was found in "<<cpu\_time\_used<<" milliseconds"<<endl;

else

cout<<"not found in "<<cpu\_time\_used<<" milliseconds"<<endl;

return 0;

}

**Exercise 2:** Generate n random numbers (using rand%1000) and store them is a hashtable (use an array of singly linked lists). The value of n should be asked from the user at runtime and decide the size of array to use accordingly (hint: big prime number).

template<class DT>

bool Hashtable<DT>::search(DT identifier)

{

bool success=false;

int location = identifier%size;

if(arr[location])

{

Node<DT> \*n= arr[location]->GetFirst();

while(n)

{

if(n->getData()==identifier)

{

success=true;

break;

}

n=n->GetNext();

}

}

else

success=false;

return success;

}

Please use the modulus function (%) as the hashfunction and **Chaining** as the Overflow handling technique. You may use the singly linked list code implemented in lab 3 to manage the chain held at each index of the hashtable. Following sample code is given for your guidance:

You have to implement the following ADT of Hashtable:

#ifndef HSH\_H

#define HSH\_H

#include "SList.h"

template<class DT>

class Hashtable

{

public:

Hashtable(int size);

bool store(DT key);

bool search(DT key);

private:

int size;

List<DT>\*\* arr;

};

#endif

You may test your code using the following client

#include<iostream>

#include<Windows.h>

#include "Hashtable.h"

#include "Hashtable.cpp"

using namespace std;

int main()

{

int num\_of\_identifiers;

cout<<"Enter maximum number of keys that need to be stored in the hashtable: ";

cin>>num\_of\_identifiers;

int size\_hashtable;

cout<<"Enter size of hashtable needed to store these many identifiers (hint: use prime number): ";

cin>>size\_hashtable;

//create a hashtable of this size

Hashtable<int>\* ht=new Hashtable<int>(size\_hashtable);

for (int i=0; i<num\_of\_identifiers; i++)

{

int key=rand()%1000;

bool was\_stored = ht->store(key);

if(!was\_stored)

cout<<key<<" could not be stored as it already exists or table is full"<<endl;

}

int find\_key;

cout<<"Enter the key to search for "<<endl;

cin>>find\_key;

DWORD start, end;//measure time

start= GetTickCount();

bool found= ht->search(find\_key);

end= GetTickCount();

double cpu\_time\_used = end - start;

if(found)

cout<<"it was found in "<<cpu\_time\_used<<" milliseconds"<<endl;

else

cout<<"not found in "<<cpu\_time\_used<<" milliseconds"<<endl;

return 0;

}