National University of Computer and Emerging Sciences

Laboratory Manual

for

Data Structures Lab

Course Instructor	Ms. Zareen Alamgir
Lab Instructor(s)	Ms. Mamoona Akbar Ms. Ammara
Section	CS-C
Semester	Fall 2022

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objective:

In this lab, students will practice:

1. Hash Maps (Linear Probing, Quadratic Probing , Double Hashing)

Question 1: (HashMap with Linear Probing)

Implement a Node struct which represents an item in a hash table.

```
template <class v>struct
HashItem
{
        int key;
        v value;
        int status;

};
```

HashMap with Linear Probing Implement a HashItem struct which represents an item in a hash array. It consists of 3 variables; key(int), value(generic), and status.

status variable can have 0, 1 or 2. Whereas 0 means empty, 1 means deleted, 2 means occupied. Status variable will be used by 'get' and 'delete' methods of HashMaps implemented in HashMap class. The default value assigned to a HashItem is 0 (empty). Now implement a HashMap class (template based):

```
template <class v>
class HashMap
{
 protected:

        HashItem<v>* hashArray;

        int capacity;

        int currentElements;

        virtual int getNextCandidateIndex(int key, int i) // The reason why getNextCandidateIndex is pure
        virtual, because this function will return nextCandidateIndex according to scheme while insertion when
        collision will occur so the implementation may vary. A collision occurs when a hash function will
        give an index value which is already occupied. In this case different hashing schemes can be applied.
        In Linear probing we simply add 1 in our answer of hash function to see if the next index is vacant and
        we will keep checking till a vacant index is found.

        void doubleCapacity()  // A method which doubles the capacity of hashArray and rehashes the    existing
        items. Use getNextCandidateIndex method to resolve collision.


 public:
        HashMap(); // Default constructor and should assign capacity 10 to hashArray

        HashMap(int const capacity); // parameterized constructor that creates hashArray of size   capacity. If
        capacity is less than 1 return error via assert(capacity>1)

        void insert(int const key, v const value, HashItem<v>* hashArray);
```

1. The insert method inserts the value at its appropriate location. Find the first candidate index of the key using: index= key mod capacity

2. 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 in i. getNextCandidateIndex will be continued to call until we find a valid index. Initially i will be 1.
3. 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.

bool deleteKey(int 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).

v* get(int const key) const; // this method returns a pointer to the corresponding value of the key. If the key is not found, it returns nullptr.

~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. Set PRIME = 7.

Create a main to test these functions properly.

```
int main()

{

   cout<<"LINEAR PROBING\n";

   HashMap<string> *map=new HashMap<string>;
   map->insert(89, "hassan", map->getHashArray());
   map->insert(18, "ali", map->getHashArray());
   map->insert(49, "ayaan", map->getHashArray());
   map->insert(58, "ahsan", map->getHashArray());
   map->insert(69, "baber", map->getHashArray());

   cout<<"QUADRATIC PROBING\n";

   QHashMap<string> *Qmap=new QHashMap<string>;

   Qmap->insert(89, "hassan", Qmap->getHashArray());
   Qmap->insert(18, "ali", Qmap->getHashArray());
   Qmap->insert(49, "ayaan", Qmap->getHashArray());
```

```
    Qmap->insert(58, "ahsan", Qmap->getHashArray());
    Qmap->insert(69, "baber", Qmap->getHashArray());


    cout<<"DOUBLE HASHING\n";
    DHashMap<string> *Dmap=new DHashMap<string>;

    Dmap->insert(89, "hassan", Dmap->getHashArray());
    Dmap->insert(18, "ali", Dmap->getHashArray());
    Dmap->insert(49, "ayaan", Dmap->getHashArray());
    Dmap->insert(58, "ahsan", Dmap->getHashArray());
    Dmap->insert(69, "baber", Dmap->getHashArray());
    return 0;

}
```

Output:

```
LINEAR PROBING

index: 9    ,key: 89    ,value: hassan
index: 8    ,key: 18    ,value: ali
index: 0    ,key: 49    ,value: ayaan
index: 1    ,key: 58    ,value: ahsan
index: 2    ,key: 69    ,value: baber
Calling get function: key found at index 2  baber

QUADRATIC PROBING
index: 9    ,key: 89    ,value: hassan
index: 8    ,key: 18    ,value: ali
index: 0    ,key: 49    ,value: ayaan
index: 2    ,key: 58    ,value: ahsan
index: 3    ,key: 69    ,value: baber

DOUBLE HASHING
index: 9    ,key: 89    ,value: hassan
index: 8    ,key: 18    ,value: ali
index: 6    ,key: 49    ,value: ayaan
index: 3    ,key: 58    ,value: ahsan
index: 0    ,key: 69    ,value: baber
```