Implementing own hash table with open addressing linear probing

Hash table - All elements are inserted in the table itself, so size of table is equal to or greader than number of elements.

insert(k) - keep probing until an empty slot is found. Once it is found insert k.

search(k) - keep probing until slot's key doesn't become equal to k or an empty slot is reached

Process is simple, user gives a (key, value) pair set as input and based on the value generated by hash function an index is generated to where the value corresponding to the particular key is stored

So, time complexity is close to O(1)

```
tempelate<typename K, typename V>
class HashNode                 // HashNode class
{
   V value;
   K key;
   HashNode(K key, V value)
   {
        this->value=value;
        this->key=key;
```

```
    }
};

class HashMap                         // Hash Map class
{
    HashNode<k,v> ** arr;
    int capacity;
    int size        // current size
    ~~HashNode~~
    HashMap ( )
    {
        capacity = 20;        // Define an initial capacity
                              // to hash table
        size = 0;
        arr = new HashNode<k,v> *[capacity];
        for(int i=0; i<capacity; i++)
            arr[i] = NULL;
    }

    int hashCode(k key)
    {
        return key % capacity;
    }
    void insertNode(k key, V value)
    {
        HashNode<k,v> *temp = new HashNode<k,v>(key, value);
        int hashIndex = hashCode(key);

        // Find next free space
        ~~While(arr[hashIndex] != NULL~~
    while (arr[hashIndex] != NULL && arr[hashIndex]->key != k
           && arr[hashIndex]->key != -1)
    {
        hashIndex++;
        hashIndex %= capacity;
    }

    // If new node to be inserted increases the
       current size
    if (arr[hashIndex] == NULL || arr[hashIndex]->key ==
        size++;

    arr[hashIndex] = temp;
}

// Function to search the value for a given key
V get (int key)
{
    int hashIndex = hashCode(key);
    int counter = 0;
    while (arr[hashIndex] != NULL)
    {
        int counter = 0;
        if (counter++ > capacity)
            return NULL;
        if (arr[hashIndex]->key == key)
            return arr[hashIndex]->value;
        hashIndex ++;
        hashIndex %= capacity;
    }
```

```
    return NULL;          // If not found return null
}
```