//HASH TABLE IMPLEMENTATION CODE:

```
//IMPLEMENTATION OF HASH TABLE
#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
const int T_S = 5;
class HashTable {
   public:
      int k;
      int v;
      HashTable(int k, int v) {
         this->k = k;
         this->v = v;
      }
};
class DelNode:public HashTable {
   private:
      static DelNode *en;
      DelNode():HashTable(-1, -1) {}
   public:
      static DelNode *getNode() {
         if (en == NULL)
            en = new DelNode();
         return en;
      }
};
DelNode *DelNode::en = NULL;
class HashMapTable {
   private:
      HashTable **ht;
   public:
      HashMapTable() {
         ht = new HashTable* [T_S];
         for (int i = 0; i < T_S; i++) {
            ht[i] = NULL;
         }
      }
      int HashFunc(int k) {
         return k % T_S;
      }
      void Insert(int k, int v) {
         int hash_val = HashFunc(k);
         int init = -1;
         int delindex = -1;
         while (hash_val != init && (ht[hash_val]  ==
DelNode::getNode() || ht[hash_val] != NULL && ht[hash_val]->k !=
k)) {
            if (init == -1)
               init = hash_val;
            if (ht[hash_val] == DelNode::getNode())
               delindex = hash_val;
               hash_val = HashFunc(hash_val + 1);
         }
         if (ht[hash_val] == NULL || hash_val == init) {
            if(delindex != -1)
               ht[delindex] = new HashTable(k, v);
            else
               ht[hash_val] = new HashTable(k, v);
         }
         if(init != hash_val) {
            if (ht[hash_val] != DelNode::getNode()) {
               if (ht[hash_val] != NULL) {
                  if (ht[hash_val]->k== k)
                     ht[hash_val]->v = v;
               }
            } else
            ht[hash_val] = new HashTable(k, v);
         }
      }
      int SearchKey(int k) {
         int hash_val = HashFunc(k);
         int init = -1;
         while (hash_val != init && (ht[hash_val] ==
DelNode::getNode() || ht[hash_val] != NULL && ht[hash_val]->k!=
k)) {
            if (init == -1)
               init = hash_val;
               hash_val = HashFunc(hash_val + 1);
         }
         if (ht[hash_val] == NULL || hash_val == init)
            return -1;
         else
            return ht[hash_val]->v;
      }
      void Remove(int k) {
         int hash_val = HashFunc(k);
         int init = -1;
         while (hash_val != init && (ht[hash_val] ==
DelNode::getNode() || ht[hash_val] != NULL && ht[hash_val]->k!=
k)) {
            if (init == -1)
               init = hash_val;
               hash_val = HashFunc(hash_val + 1);
         }
         if (hash_val != init && ht[hash_val] != NULL) {
            delete ht[hash_val];
            ht[hash_val] = DelNode::getNode();
         }
      }
      ~HashMapTable() {
         delete[] ht;
```

```cpp
      }
};
int main() {
   HashMapTable hash;
   int k, v;
   int c;
   while(1) {
      cout<<"1.Insert element into the table"<<endl;
      cout<<"2.Search element from the key"<<endl;
      cout<<"3.Delete element at a key"<<endl;
      cout<<"4.Exit"<<endl;
      cout<<"Enter your choice: ";
      cin>>c;
      switch(c) {
         case 1:
            cout<<"Enter element to be inserted: ";
            cin>>v;
            cout<<"Enter key at which element to be inserted: ";
            cin>>k;
            hash.Insert(k, v);
         break;
         case 2:
            cout<<"Enter key of the element to be searched: ";
            cin>>k;
            if(hash.SearchKey(k) == -1) {
               cout<<"No element found at key "<<k<<endl;
               continue;
            } else {
               cout<<"Element at key "<<k<<" : ";
               cout<<hash.SearchKey(k)<<endl;
            }
         break;
         case 3:
            cout<<"Enter key of the element to be deleted: ";
            cin>>k;
            hash.Remove(k);
         break;
         case 4:
            exit(1);
         default:
            cout<<"\nEnter correct option\n";
      }
   }
   return 0;
}
```