|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in CST, IT  Year 2/3 (2020/21) Semester 4/6 | **Week 6** |
| **1-2 Hours** |
| **Tutorial 6 – Hash Tables** | |

1. Devise a *perfect* hash function for the following domains. What is the range of your function?
2. The set of integers in [-100, 100]

|  |
| --- |
| h(k) = k if 0<= k <=100  k +201 if -100 <= k <= -1  Range of h(k) is [0,200]  h(k) = k+100  Range of h(k) is [0,200] |

1. The set of 3-letter English words

|  |
| --- |
| Let code(‘A’) = 0, code(‘B’) = 1,…, code(‘Z’) = 25  h(k) = code(k[0])\*26^2 + code(k[1])\*26 + code(k[2])  Range of h(k) = [0,26^3 -1 = 17575] |

1. The set of tic-tac-toe positions

|  |
| --- |
| h(position) is a 9-digit integer, where each digit corresponds to one square: blank =0 , nought=1 ,cross=2.  h(position) = 200211102 corresponds to  Range of h(k) = [0,222222222] |

1. The specification of the Dictionary ADT implemented using a Hash Table with separate chaining is given below.

|  |
| --- |
| // Dictionary.h - - Specification of Dictionary ADT  #include<string>  #include<iostream>  using namespace std;  const int MAX\_SIZE = 100;  typedef string ItemType;  typedef int KeyType;  struct Node  {  KeyType key; // search key  ItemType item; // data item  Node \*next; // pointer pointing to next item  };  class Dictionary  {  private:  Node \*items[MAX\_SIZE];  int size; // number of items in the Dictionary  public:  // constructor  Dictionary();  // destructor  ~Dictionary();  int hash(KeyType key);  // add a new item with the specified key to the Dictionary  bool add(KeyType newKey, ItemType newItem);  // remove an item with the specified key in the Dictionary  void remove(KeyType key);  // get an item with the specified key in the Dictionary (retrieve)  ItemType get(KeyType key);  // check if a specified key is in the Dictionary  bool contains(KeyType key);  // check if the Dictionary is empty  bool isEmpty();  // check the size of the Dictionary  int getLength();    //------------------- Other useful functions -----------------  // display the items in the Dictionary  void print();  }; |

Implement the following operations of the Dictionary ADT

1. **int hash(KeyType key);**

|  |
| --- |
| int Dictionary::hash(KeyType key){  Return key % MAX\_SIZE;  } |

1. **bool add(KeyType key, ItemType item);**

|  |
| --- |
| bool Dictionary::add(KeyType newKey, ItemType newItem){  int index = hash(newKey);  if (items[index] == NULL) {  items[index] = new Node;  items[index]->key = newKey;  items[index]->item = newItem;  items[index]->next = NULL;  }  else {  Node\* current = items[index];  if (current->key != newKey) { return false; }  while (current->next != NULL) {  current = current->next;  if (current->key == newKey) { //duplicate data  return false;  }  }  Node\* temp = new Node;  temp->item = newItem;  temp->key = newKey;  temp->next = NULL;  current->next = temp;  }  size++;  return true;  } |

1. **bool remove(KeyType key);**

|  |
| --- |
| Bool Dictionary::remove(KeyType key){  int index = hash(key);  if (items[index] != NULL) {  Node \*current = items[index];  if (current->key == key){  items[index] == items[index]->next;  delete current;  size--;  return true;  } else {  while (current->next != NULL) &&  (current->next->key != key) {  current = current->next;  }  if (current->next != NULL){  Node \*temp = current->next;  current->next = temp->next;  delete temp;  temp = NULL;  size--;  return true;  }  }  }  return false;  } |