Assignment 6

Title: The Dictionary ADT

Problem Statement: Implement all the functions of a dictionary (ADT) using hashing. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique Standard Operations: Insert(key, value), Find(key), Delete(key) (Use linear probing with and without replacement. Calculate the average search cost for both.)

Learning Objective: To understand implementation of all the functions of a dictionary (ADT) and standard operations on Dictionary.

Learning Outcome: At the end of this assignment students will able to perform standard operations on Dictionary ADT.

Concepts related Theory:

The Dictionary ADT: A dictionary is an ordered or unordered list of key-element pairs, where keys are used to locate elements in the list. Dictionary is a data structure, which is generally an association of unique keys with some values. One may bind a value to a key, delete a key (and naturally an associated value) and look up for a value by the key. Values are not required to be unique. Example: consider a data structure that stores bank accounts; it can be viewed as a dictionary, where account numbers serve as keys for identification of account objects. A Dictionary (also known as Table or Map) can be implemented in various ways: using a list, binary search tree, hash table, etc. In each case: the implementing data structure has to be able to hold key-data pairs and able to do insert, find, and delete operations paying attention to the key.

Hashing: Hashing is a method for directly referencing an element in a table by performing arithmetic transformations on keys into table addresses. This is carried out in two steps:

1. Computing the so-called hash function H: K -> A.
2. Collision resolution, which handles cases where two or more different keys hash to the same table address.

Implementation of Hash table:

Hash tables consist of two components: a *bucket array* and a *hash function*.

A hash table is a collection of items which are stored in such a way as to make it easy to find them later. Each position of the hash table, often called a slot, can hold an item and is named by an integer value starting at 0. For example, we will have a slot named 0, a slot named 1, a slot named 2, and so on.

Consider a dictionary, where keys are integers in the range [0, N-1]. Then, an array of size N can be used to represent the dictionary. Each entry in this array is thought of as a "bucket". An element *e* with key *k* is inserted in A[k]. Bucket entries associated with keys not present in the dictionary contain a special NO_SUCH_KEY object. If the dictionary contains elements with the same key, then two or more different elements may be mapped to the same bucket of A. In this case, we say that a *collision* between these elements has occurred. One easy way to deal with collisions is to allow a sequence of elements with the same key, k, to be stored in A[k]. Assuming that an arbitrary element with key k satisfies queries findItem(k) and removeItem(k), these operations are now performed in O(1) time, while insertItem(k, e) needs only to find where on the existing list A[k] to insert the new item, e. The drawback of this is that the size of the bucket array is the size of the set from which key are drawn, which may be huge.

Class Definition:

```
class node
{
	char key[10];
	string meaning;
public:
	node(char* k)
	{
		strcpy(key, k);
		getline(cin, meaning);
	}
	void show()
	{
		cout << key << " : " << meaning << "\n";
	}
	friend class Hash;
};

class Hash
{
	node* arr[26];
	int chain[26];
public:
	Hash()
	{
		for (int i = 0; i < 26; i++)
		{
			arr[i] = NULL;
			chain[i] = -1;
		}
	}
	void create_wo_rep(int);
	void insert_wo_rep();
	void create_wrep(int);
	void insert_wrep();
	void disp();
	void del();
	void find();
};
```

Pseudo Codes:

```
void Hash::create_wo_rep(int n)
{
        char str[10];
        for(int i=0;i<n;i++)
        {
                cin.getline(str, 10);
                if(arr[(int) str[0] - 97] == NULL)
                {
                        arr[(int) str[0] - 97] = new node(str);
                }
                else
                {
                        int j = (int) str[0] - 97;
                        while(chain[j] != -1)
                        {
                                j = chain[j];
                        }
                        int p = j;
                        while (arr[j] != NULL)
                        {
                                j++;
                        }
                        arr[j] = new node(str);
                        chain[p] = j;
                }
        }
}

void Hash::insert_wo_rep()
{
        char str[10];
        cin.getline(str, 10);
        if(arr[(int) str[0] - 97] == NULL)
        {
                arr[(int) str[0] - 97] = new node(str);
        }
        else
        {
```

```
        int j = (int) str[0] - 97;
        while(chain[j] != -1)
        {
            j = chain[j];
        }
        int p = j;
        while (arr[j] != NULL)
        {
            j++;
        }
        arr[j] = new node(str);
        chain[p] = j;
    }
}

void Hash::create_wrep(int n)
{
    char str[10];
    for(int i=0;i<n;i++)
    {
        cin.getline(str, 10);
        if(arr[(int)str[0]-97] == NULL)
        {
            arr[(int)str[0]-97] = new node(str);
        }
        else
        {
            char* temp = arr[(int)str[0]-97]->key;
            if(str[0] == temp[0])
            {
                int j = (int) str[0] - 97;
                while(j != -1 && chain[j] != -1)
                {
                    j = chain[j];
                }
                int p = j;
                while (arr[j] != NULL)
                {
                    j++;
                }
                arr[j] = new node(str);
                chain[p] = j;
            }
            else
            {
                int j = (int) str[0] - 97;
                int o = j;
                while(j != -1 && chain[j] != -1)
                {
                    j = chain[j];
                }
                int p = j;
                while (arr[j] != NULL)
                {
                    j++;
                }
                arr[j] = arr[o];
                arr[o] = new node(str);
                chain[p] = j;
                chain[o] = -1;
            }
        }
    }
}

void Hash::insert_wrep()
{
    char str[10];
    cin.getline(str, 10);
    if(arr[(int)str[0]-97] == NULL)
    {
        arr[(int)str[0]-97] = new node(str);
    }
    else
    {
        char* temp = arr[(int)str[0]-97]->key;
        if(str[0] == temp[0])
        {
            int j = (int) str[0] - 97;
            while(j != -1 && chain[j] != -1)
            {
                j = chain[j];
            }
            int p = j;
            while (arr[j] != NULL)
            {
                j++;
            }
            arr[j] = new node(str);
            chain[p] = j;
        }
        else
        {
            int j = (int) str[0] - 97;
            int o = j;
            while(j != -1 && chain[j] != -1)
            {
                j = chain[j];
            }
            int p = j;
            while (arr[j] != NULL)
            {
                j++;
            }
            arr[j] = arr[o];
            arr[o] = new node(str);
            chain[p] = j;
            chain[o] = -1;
        }
    }
}

void Hash::disp()
{
    for (int i = 0; i < 26; i++)
    {
        if (arr[i] != NULL)
        {
            cout << i << ". ";
            arr[i]->show();
            cout << chain[i];
            cout << "\n";
        }
    }
}

void Hash::del()
{
    cout << "Enter key to be deleted: ";
    char str[10];
    cin.getline(str,8);
    int j = (int) str[0] - 97;
    int i = -1,flag = 0;
    while(j != -1)
    {
        if(strcmp(arr[j]->key,str) == 0)
        {
            flag = 1;
            break;
        }
        i = j;
        j = chain[j];
    }
    if(flag == 0)
    {
        cout << "Key not found.\n";
        return;
    }
    if(i != -1)
    {
        chain[i] = chain[j];
    }
    arr[j] = NULL;
}

void Hash::find()
{
    cout << "Enter key to be searched: ";
    char str[10];
    int j = (int) str[0] - 97;
```

```
        int i = -1,flag = 0;
        while(j != -1)
        {
                if(strcmp(arr[j]->key,str) == 0)
                {
                        flag = 1;
                        break;
                }
                i = j;
                j = chain[j];
        }
        if(flag == 0)
        {
                cout << "Key not found.\n";
                return;
        }
        if(i != -1)
        {
                chain[i] = chain[j];
        }
        arr[j]->show();
}
```

Test Case:

```
1. Without Replacement.
2. With Replacement.
Enter choice: 2
1. Create Dictionary.
2. Insert into Dictionary.
3. Delete from Dictionary.
4. Find from Dictionary.
5. Display Dictionary.
6. Exit.
Enter choice: 1
Enter number of Entries: 3
ask
enquire
ball
cricket
```

```
bat
cricket
Enter choice: 5
0. ask : enquire
-1
1. ball : cricket
2
2. bat : cricket
-1
Enter choice: 2
cat
pet
Enter choice: 5
0. ask : enquire
-1
1. ball : cricket
3
2. cat : pet
-1
3. bat : cricket
-1
Enter choice: 4
Enter key to be searched: cat
cat: pet
```

Conclusion: After successfully completing this assignment, Students have learned implementation of Dictionary(ADT) using Hashing and various Standard operations on Dictionary ADT .