**Assignment 1**

**Consider the Dictionary Implementations which allow for efficient storage and retrieval of**

**key-value pairs using binary search trees. Each node in the tree store (key, value) pair. The**

**dictionary should support the following operations efficiently:**

**1. Insert word (Handle insertion of duplicate entry)**

**2. Delete word**

**3. Search specific word**

**4. Display dictionary (Traversal)**

**5. Mirror image of dictionary**

**6. Create a copy of dictionary**

**7. Display dictionary level wise**

**CODE:**

#include <iostream>

using namespace std;

class QNode

{

public:

string key;

string value;

QNode \*left;

QNode \*right;

QNode(string key = "", string value = "")

{

this->key=key;

this->value=value;

left=right=nullptr;

}

};

class Node

{

public:

QNode \*data;

Node \*next;

Node(QNode \*data)

{

this->data=data;

next=nullptr;

}

};

class LinkedListQueue

{

public:

Node \*front;

Node \*rear;

LinkedListQueue()

{

front=rear=nullptr;

}

void add(QNode \*data)

{

Node \*temp = new Node(data);

if (rear ==nullptr)

{

front=rear=temp;

return;

}

rear->next=temp;

rear=temp;

}

void remove()

{

if (front == nullptr)

return;

Node \*temp=front;

front=front->next;

if(front==nullptr)

{

rear = nullptr;

}

delete temp;

}

Node \*peek()

{

if (front == nullptr)

{

return nullptr;

}

return front;

}

bool isEmpty()

{

return front==nullptr;

}

};

class Dictionary

{

public:

QNode \*root;

Dictionary()

{

root=nullptr;

}

QNode \*Insert\_word(QNode \*root, string key, string value)

{

if (root==nullptr)

{

return new QNode(key, value);

}

if (root->key==key)

{

cout<<"Duplicate entry! Word already exists.\n";

return root;

}

if (root->key<key)

{

root->right=Insert\_word(root->right, key, value);

}

else

{

root->left=Insert\_word(root->left, key, value);

}

return root;

}

QNode \*Successor(QNode \*curr)

{

curr=curr->right;

while(curr != nullptr && curr->left != nullptr)

{

curr=curr->left;

}

return curr;

}

QNode \*deletel(QNode \*root, string key)

{

if (root == nullptr)

{

return root;

}

if (root->key > key)

{

root->left = deletel(root->left, key);

}

else if(root->key < key)

{

root->right = deletel(root->right, key);

}

else

{

if (root->left==nullptr)

{

QNode \*temp=root->right;

delete root;

return temp;

}

else if (root->right==nullptr)

{

QNode \*temp=root->left;

delete root;

return temp;

}

QNode \*succ=Successor(root);

root->key=succ->key;

root->right=deletel(root->right, succ->key);

}

return root;

}

QNode \*search\_word(QNode \*root, string key)

{

if (root==nullptr || root->key == key)

{

return root;

}

if (root->key >key)

{

return search\_word(root->left, key);

}

return search\_word(root->right, key);

}

void displayInTr(QNode \*root)

{

if (root==nullptr)

{

return;

}

displayInTr(root->left);

cout<<root->key << " : "<< root->value << endl;

displayInTr(root->right);

}

void mirror(QNode \*root)

{

if (root == nullptr)

{

return;

}

swap(root->left, root->right);

mirror(root->left);

mirror(root->right);

}

QNode \*create\_copy(QNode \*node)

{

if (node==nullptr)

{

return nullptr;

}

QNode \*newNode=new QNode(node->key, node->value);

newNode->left=create\_copy(node->left);

newNode->right=create\_copy(node->right);

return newNode;

}

void levelOrder(QNode \*root)

{

if (root==nullptr)

return;

LinkedListQueue q;

q.add(root);

while (!q.isEmpty())

{

Node \*temp = q.peek();

q.remove();

if (temp->data != nullptr)

{

cout << temp->data->key << ": " << temp->data->value << " ";

if (temp->data->left != nullptr)

q.add(temp->data->left);

if (temp->data->right != nullptr)

q.add(temp->data->right);

}

}

cout << "\n";

}

};

int main()

{

Dictionary dict;

int choice;

string key, value;

do

{

cout << "\n1. Insert word (Handle insertion of duplicate entry) \n2. Delete word\n3. Search specific word\n4. Display dictionary (Traversal)\n5. Mirror image of dictionary\n6. Create a copy of dictionary\n7. Level Order Display\n8. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter key and value: ";

cin >> key >> value;

dict.root = dict.Insert\_word(dict.root, key, value);

break;

case 2:

cout << "Enter key to delete: ";

cin >> key;

dict.root = dict.deletel(dict.root, key);

break;

case 3:

cout << "Enter key to search: ";

cin >> key;

if (dict.search\_word(dict.root, key) != nullptr)

cout << "Word found!\n";

else

cout << "Word not found!\n";

break;

case 4:

dict.displayInTr(dict.root);

break;

case 5:

dict.mirror(dict.root);

break;

case 6:

dict.create\_copy(dict.root);

cout << "Copy created.\n";

break;

case 7:

dict.levelOrder(dict.root);

break;

case 8:

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice!\n";

}

} while (choice != 8);

return 0;

}