ASSIGNMENT 1

AIM:

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

SOURCE CODE:

```
#include <iostream>
#include <queue>
using namespace std;

// Structure for each node in BST
struct Node {
    string key;
    string value;
    Node* left;
    Node* right;
```

```
    Node(string k, string v) {
        key = k;
        value = v;
        left = right = nullptr;
    }
};

// Dictionary class using BST
class Dictionary {
private:
    Node* root;

    // Helper functions
    Node* insert(Node* root, string key, string value) {
        if (!root) return new Node(key, value);

        if (key < root->key)
            root->left = insert(root->left, key, value);
        else if (key > root->key)
            root->right = insert(root->right, key, value);
        else
            root->value = value; // Handle duplicate by updating value

        return root;
    }

    Node* search(Node* root, string key) {
        if (!root || root->key == key)
            return root;

        if (key < root->key)
            return search(root->left, key);
        else
            return search(root->right, key);
    }

    Node* findMin(Node* root) {
        while (root->left) root = root->left;
        return root;
    }
```

```
Node* deleteNode(Node* root, string key) {
    if (!root) return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else {
        // Node with one or no child
        if (!root->left) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (!root->right) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        // Node with two children: get inorder successor
        Node* temp = findMin(root->right);
        root->key = temp->key;
        root->value = temp->value;
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}

void inOrder(Node* root) {
    if (root) {
        inOrder(root->left);
        cout << root->key << " : " << root->value << endl;
        inOrder(root->right);
    }
}

void levelOrder(Node* root) {
    if (!root) return;
```

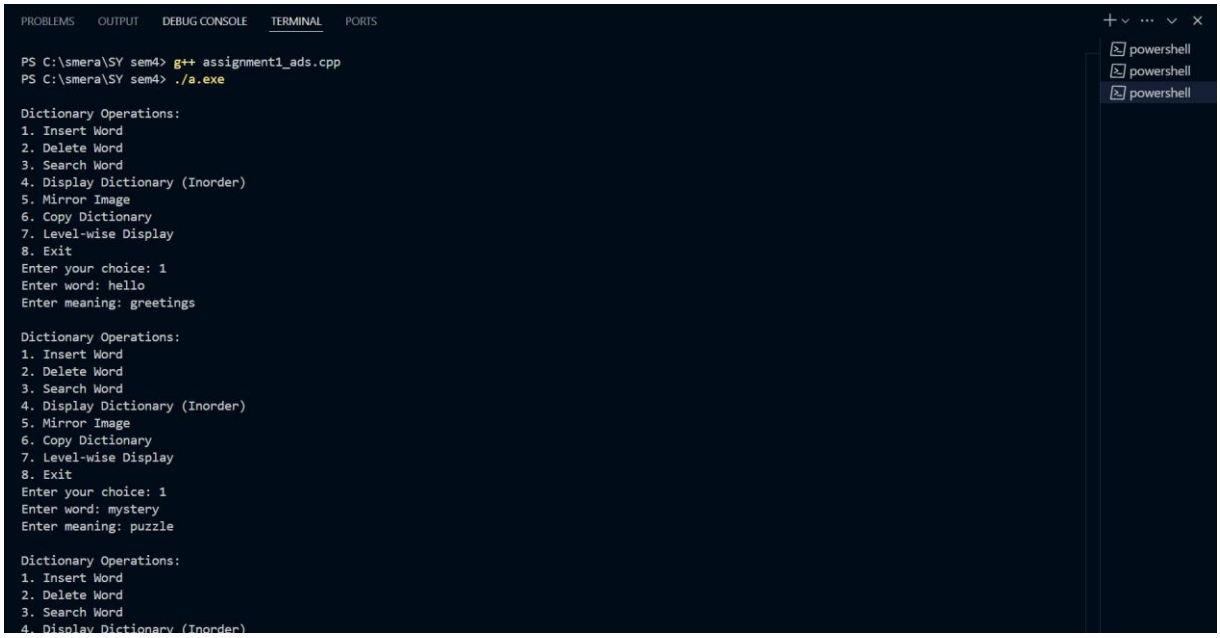
```
        queue<Node*> q;
        q.push(root);

        while (!q.empty()) {
            Node* temp = q.front();
            q.pop();
            cout << temp->key << " : " << temp->value << " | ";

            if (temp->left) q.push(temp->left);
            if (temp->right) q.push(temp->right);
        }
        cout << endl;
    }

    Node* mirror(Node* root) {
        if (!root) return nullptr;

        Node* mirrored = new Node(root->key, root->value);
        mirrored->left = mirror(root->right);
        mirrored->right = mirror(root->left);

        return mirrored;
    }

    Node* copyTree(Node* root) {
        if (!root) return nullptr;

        Node* newRoot = new Node(root->key, root->value);
        newRoot->left = copyTree(root->left);
        newRoot->right = copyTree(root->right);

        return newRoot;
    }

public:
    Dictionary() { root = nullptr; }

    void insert(string key, string value) {
        root = insert(root, key, value);
    }
```

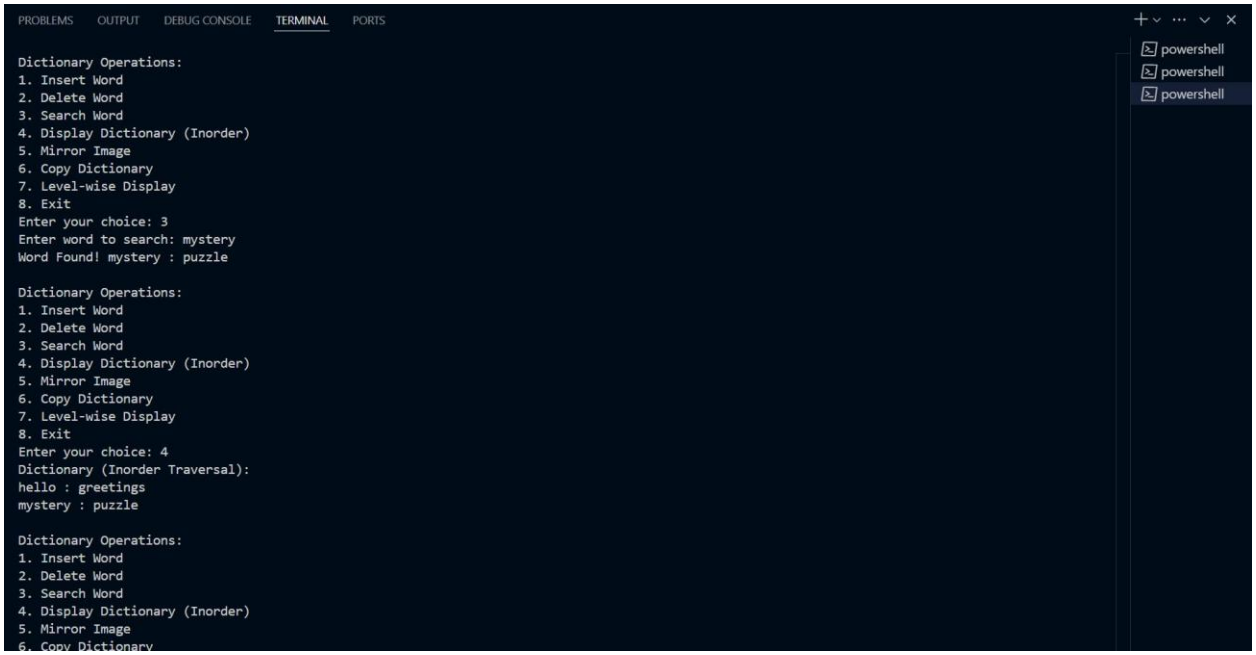
```
    void search(string key) {
        Node* res = search(root, key);
        if (res)
            cout << "Word Found! " << res->key << " : " << res->value << endl;
        else
            cout << "Word Not Found!" << endl;
    }

    void deleteWord(string key) {
        root = deleteNode(root, key);
    }

    void display() {
        if (!root) cout << "Dictionary is empty!" << endl;
        else inOrder(root);
    }

    void mirrorDictionary() {
        Node* mirroredRoot = mirror(root);
        cout << "Mirror Image of Dictionary:" << endl;
        inOrder(mirroredRoot);
    }

    void copyDictionary() {
        Node* copiedRoot = copyTree(root);
        cout << "Copied Dictionary:" << endl;
        inOrder(copiedRoot);
    }

    void levelWiseDisplay() {
        if (!root) cout << "Dictionary is empty!" << endl;
        else levelOrder(root);
    }
};

// Main function with switch case
int main() {
    Dictionary dict;
    int choice;
```

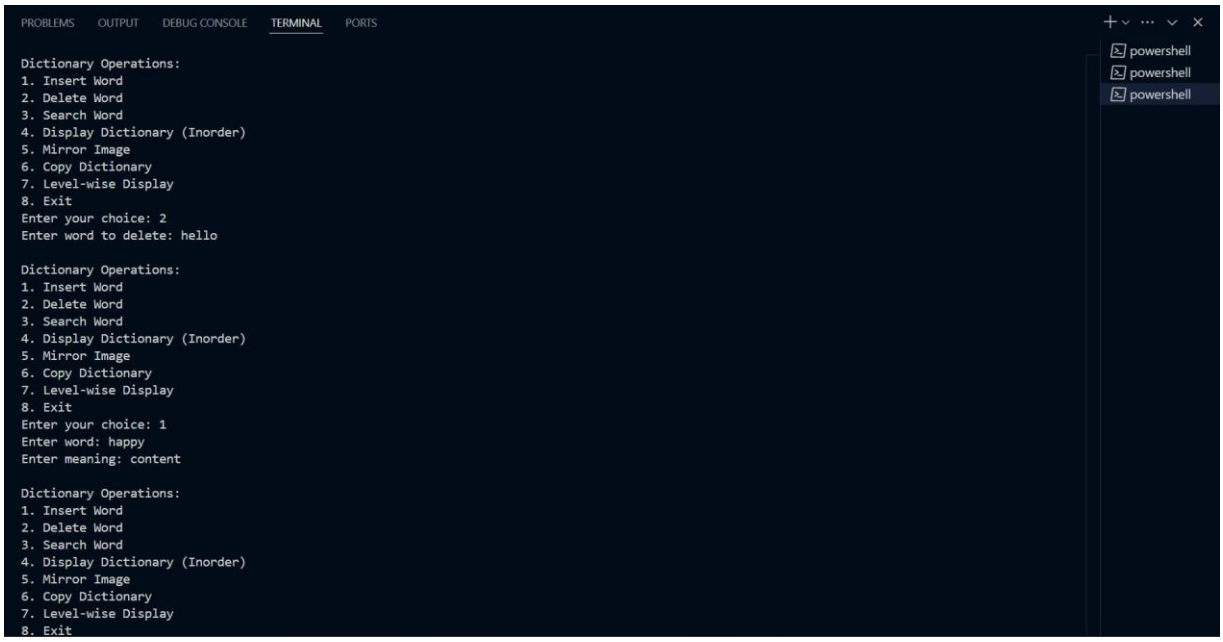
```
string key, value;

do {
    cout << "\nDictionary Operations:\n";
    cout << "1. Insert Word\n2. Delete Word\n3. Search Word\n";
    cout << "4. Display Dictionary (Inorder)\n5. Mirror Image\n";
    cout << "6. Copy Dictionary\n7. Level-wise Display\n8. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1:
            cout << "Enter word: ";
            cin >> key;
            cout << "Enter meaning: ";
            cin.ignore();
            getline(cin, value);
            dict.insert(key, value);
            break;
        case 2:
            cout << "Enter word to delete: ";
            cin >> key;
            dict.deleteWord(key);
            break;
        case 3:
            cout << "Enter word to search: ";
            cin >> key;
            dict.search(key);
            break;
        case 4:
            cout << "Dictionary (Inorder Traversal):\n";
            dict.display();
            break;
        case 5:
            dict.mirrorDictionary();
            break;
        case 6:
            dict.copyDictionary();
            break;
        case 7:
```

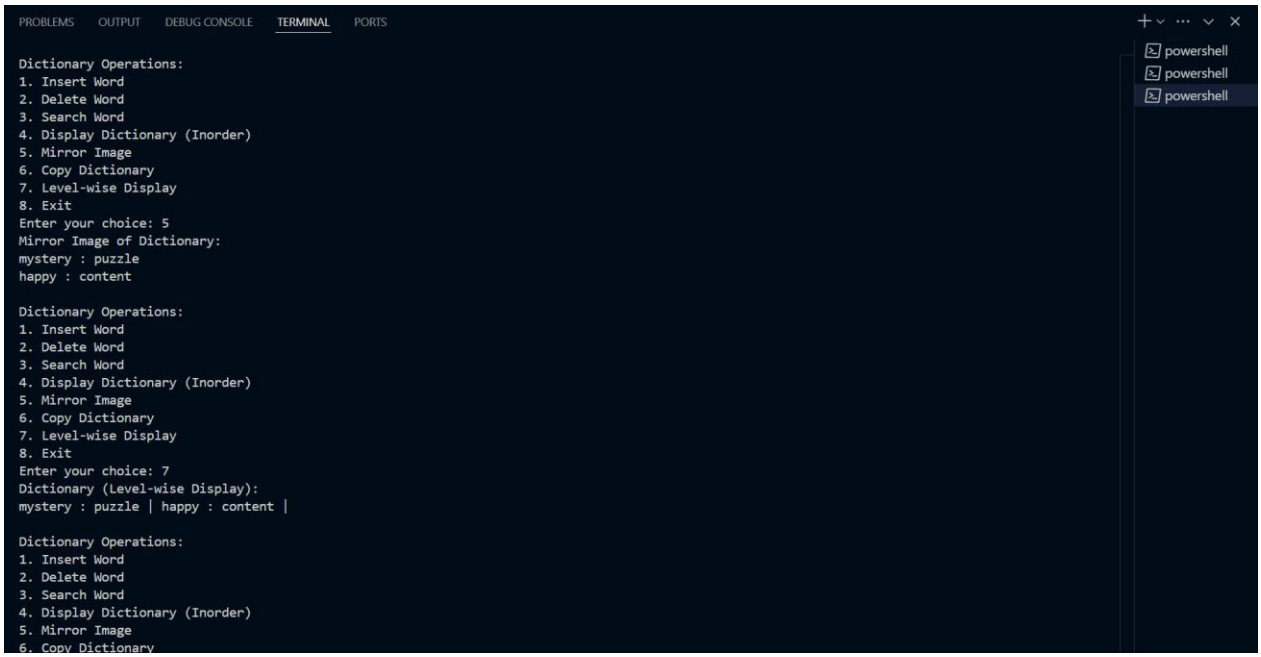
```
            cout << "Dictionary (Level-wise Display):\n";
            dict.levelWiseDisplay();
            break;
        case 8:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice! Try again.\n";
      }
   } while (choice != 8);

   return 0;
}
```

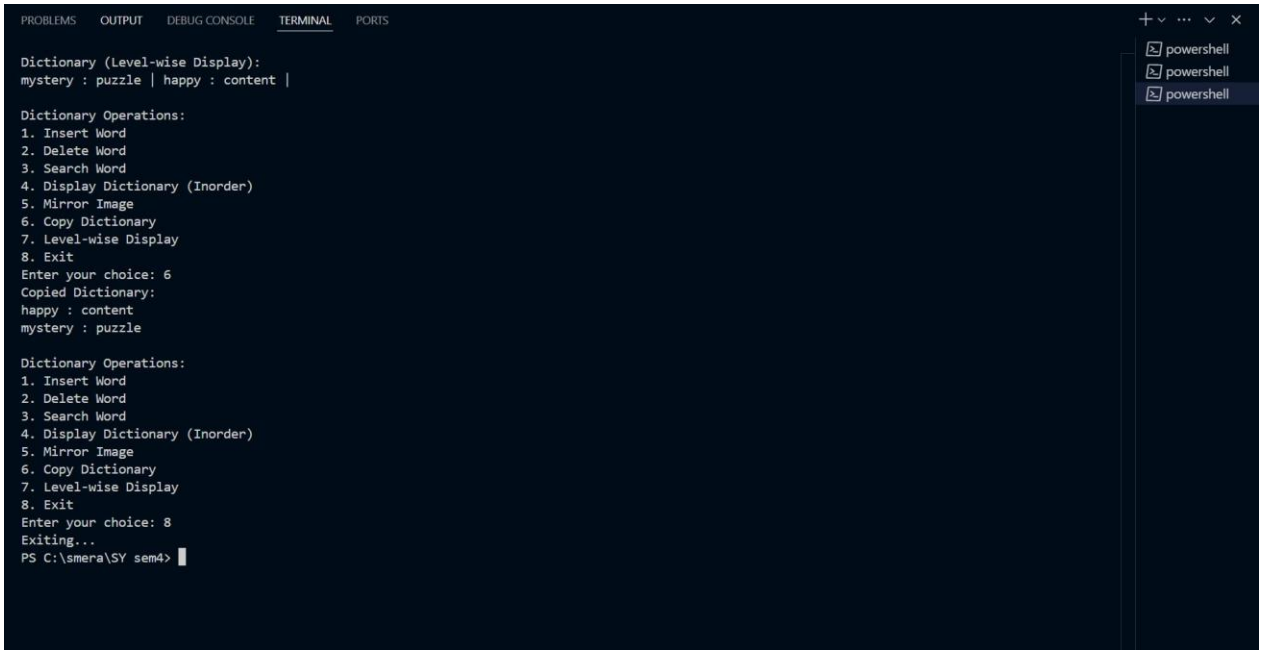
OUTPUT:




PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
powershell
powershell
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 3
Enter word to search: mystery
Word Found! mystery : puzzle
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 4
Dictionary (Inorder Traversal):
hello : greetings
mystery : puzzle
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
powershell
powershell
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 2
Enter word to delete: hello
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 1
Enter word: happy
Enter meaning: content
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
powershell
powershell
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 5
Mirror Image of Dictionary:
mystery : puzzle
happy : content
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 7
Dictionary (Level-wise Display):
mystery : puzzle | happy : content |
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
powershell
powershell
Dictionary (Level-wise Display):
mystery : puzzle | happy : content |
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 6
Copied Dictionary:
happy : content
mystery : puzzle
Dictionary Operations:
1. Insert Word
2. Delete Word
3. Search Word
4. Display Dictionary (Inorder)
5. Mirror Image
6. Copy Dictionary
7. Level-wise Display
8. Exit
Enter your choice: 8
Exiting...
PS C:\smera\SY sem4>