Data Structures and Algorithms

CSC - 221

:Group Members:

SHAYAN HASSAN ABBASI (167)
ZEESHAN ALI (197)

Department of Computer Science

BAHRIA UNIVERSITY, ISLAMABAD

MANUAL DICTIONARY

SEMESTER PROJECT:-

CODE:

```
#include <iostream> // Including Input Ouput Library Directory
#include <conio.h>
#define count 10 // Defining Symbolic Constant Named "count" Having Value 10
using namespace std; // To Access Definition Of STD Without Having To Specify Namespace
Everytime

class Node // Node Class
{
public: // To Access Data Members Everywhere In Program
    Node* right = NULL; // Right Node
    Node* left = NULL; // Left Node
    string word; // Initialize String Type Data
    string meaning; // Initialize String Type Data
}; // Object Of Struct Class

class Queue // Queue Class
{
    Node* Q[20]; // Queue For 20 Datas
    int Front, Rear; // Front And Rear Of Queue

public: // To Access Everywhere In Program
    Queue() // Constructor
    {
        Front = Rear = -1; // Initialize Front And Rear Both -1 Because It Starts From -1
    }

    void insert(Node* n1) // Function To Insert Data In Queue (EnQueue)
    {
        Q[Rear++] = n1; // Inserting From Rear Of Queue
    }

    Node* Delete() // Function To Delete Data From Queue (DeQueue)
    {
        if (Front == Rear) // If The Queue Is Empty
        {
            return NULL; // Return NULL
        }
        else // Otherwise
        {
            return (Q[Front++]); // Removing From Front Of Queue
        }
    }

    bool isEmpty() // Function To Check Weather Queue Is Empty Or Not
    {
        if (Front == Rear) // If Front And Rear Are At Same Position
            return true; // Queue Is Empty
        else // Otherwise
```

```
            return false; // Queue Is Not Empty
    }
}; // Class Ends

class Dictionary // Main Class Dictionary
{
public: // To Access Everywhere In Program
    Node* Create() // Function To Create Dictionary
    {
        Node* root = NULL; // Initiallize Root Of Binary Tree To Null
        Node* current = NULL; // Current Node Of Binary Tree
        Node* parent = NULL; // Parent Node Of Binary Tree
        char ch; // Character Type Data

        do { // Do-While Loop
            Node* temp = new Node(); // Creating New Node Temp
            temp->left = NULL; // Initiallize Left Node To Null
            temp->right = NULL; // Initiallize Right Node To Null
            cout << "Enter Word: ";
            cin >> temp->word; // Taking Word From User
            cout << "Enter Meaning: ";
            cin >> temp->meaning; // Taking Meaning Of Word From User
            if (root == NULL) // If Root Node Is Null
            {
                root = temp; // Make Temp Root Of Binary Tree
            }
            else // If Root Is Not Empty
            {
                current = root; // Current Node Will Be Consider Root
                while (current)
                {
                    parent = current; // Consider Current Node As Parent
                    if (current->word < temp->word) // If Current Node's Data Is Less Than
New Data
                        current = current->right; // Traverse To Right Side Of Binary Tree
                    else // Otherwise
                        current = current->left; // Traverse To Left Side Of Binary Tree
                }
                if (parent->word < temp->word) // If Parent's Data Is Less Than New Data
                    parent->right = temp; // Put It On Right Node Of Parent
                else // Otherwise
                    parent->left = temp; // Put It On Left Node Of Parent
            }

            cout << "Do You Want To Continue? (Y/N) ";
            cin >> ch; // Taking Choice From User
        } while (ch == 'y' || ch == 'Y'); // Loop Runs As Far User Keeps Typing 'y' OR 'Y'
        return root; // Return Root Value
    }

    void Add(Node* root) // Function To Add New Word In Dictionary
    {
        Node* current = NULL; // Current Node
        Node* parent = NULL; // Parent Node
        Node* temp = new Node(); // New Node
        temp->left = NULL; // Initiallize Left Of New Node To Null
        temp->right = NULL; // Initiallize Right Of New Node To Null
        cout << "Enter Word: ";
        cin >> temp->word; // Taking Word From User
```

```
    cout << "Enter Meaning: ";
    cin >> temp->meaning; // Taking Meaning Of Word From User
    current = root; // Current Node Will Be Consider As Root
    while (current)
    {
        parent = current; // Consider Current Node As Parent
        if (current->word < temp->word) // If Current Node's Data Is Less Than New Data
            current = current->right; // Traverse To Right Side Of Binary Tree
        else // Otherwise
            current = current->left; // Traverse To Left Side Of Binary Tree
    }
    if (parent->word < temp->word) // If Parent's Data Is Less Than New Data
        parent->right = temp; // Put It On Right Node Of Parent
    else // Otherwise
        parent->left = temp; // Put It On Left Node Of Parent
    cout << "Word Successfully Added To Dictionary\n";
}

void Delete(Node*& root, string word) // Function To Delete Data From Binary Tree
{
    if (root == NULL) // If Tree Is Empty
    {
        return; // Return
    }
    if (word < root->word) // If Data Is Less Than Current Data
    {
        Delete(root->left, word); // Traverse To Left Side
    }
    else if (word > root->word) // If Data Is Greater Than Current Data
    {
        Delete(root->right, word); // Traverse To Right Side
    }
    else // If Data Is Found
    {
        if (root->left == NULL && root->right == NULL) // If It's A Leaf Node
        {
            delete root; // Delete It
            root = NULL; // Make It Null
            return; // Return
        }
        else if (root->left == NULL) // If It Has Only Right Child
        {
            Node* temp = root; // Make New Node
            root = root->right; // Make Right Of Node As Root
            delete temp; // Delete Previous Node
        }
        else if (root->right == NULL) // If It Has Only Left Child
        {
            Node* temp = root; // Make New Node
            root = root->left; // Make Left Of Node As Root
            delete temp; // Delete Previous Node
        }
        else // If It Has Two Children
        {
            Node* temp = root->right; // Make New Node
            while (temp->left) // Traverse To Left Most Node
                temp = temp->left; // Make It Parent
            root->word = temp->word; // Replace Data
            root->meaning = temp->meaning; // Replace Meaning
```

```
            Delete(root->right, temp->word); // Delete Right Of Root
        }
    }
}

void Update(Node* root) // Function To Update Meaning Of Any Word
{
    Node* current = NULL; // Current Node
    string w; // String Type Data
    cout << "Enter Word To Update: ";
    cin >> w; // Take Word From User
    current = root; // Consider Current As Root Node
    while (current) // Loop Will Keep Executing As Long As Current Is Non-Zero
    {
        if (current->word == w) // If Word Is Found
        {
            cout << "Enter New Meaning: ";
            cin >> current->meaning; // Take New Meaning From User
            cout << "Meaning Updated Successfully\n";
            return; // Return
        }
        else // Otherwise
        {
            if (current->word < w) // If Current Node's Data Is Less Than Word
                current = current->right; // Traverse To Right Side Of Binary Tree
            else // Otherwise
                current = current->left; // Traverse To Left Side Of Binary Tree
        }
    }
    cout << "Word Not Found\n"; // If All Conditions Fails
}

void Display(Node* root, Node* ptr, int level) // Function To Display Tree Like Structure
{
    if (ptr != NULL) // If Ptr Is Not Null
    {
        level += count; // Increment In Level
        Display(root, ptr->right, level + 1); // Display Right Side Of Binary Tree
        cout << endl;
        if (ptr == root) // If Root Found
        {
            cout << "Root->";
        }
        else // Otherwise
        {
            for (int i = count; i < level; i++) // Loop For Designing
                cout << " ";
        }
        cout << ptr->word << "->" << ptr->meaning << "\n"; // Display Word And Meaning
        Display(root, ptr->left, level + 1); // Display Left Side Of Binary Tree
    }
}

void InOrder(Node* root) // InOrder Traversal (Depth-First)
{
    if (root == NULL) // If Root Node Is Null
    {
        return; // Nothing To Traverse
    }
```

```
        else // If Root Node Is Not Null
        {
            InOrder(root->left); // Traverse Left Subtree
            cout << root->word << "->" << root->meaning << "\n"; // Print Data Of Root Node
            InOrder(root->right); // Traverse Right Subtree
        }
    }

    void PreOrder(Node* root) // PreOrder Traversal (Depth-First)
    {
        if (root == NULL) // If Root Node Is Null
        {
            return; // Nothing To Traverse
        }
        else // When Root Node Is Not Null
        {
            cout << root->word << "->" << root->meaning << "\n"; // Print Data Of Root Node
            InOrder(root->left); // Traverse Left Subtree
            InOrder(root->right); // Traverse Right Subtree
        }
    }

    void PostOrder(Node* root) // PostOrder Traversal (Depth-First)
    {
        if (root == NULL) // If Root Node Is Null
        {
            return; // Nothing To Traverse
        }
        else // When Root Node Is Not Null
        {
            InOrder(root->left); // Traverse Left Subtree
            InOrder(root->right); // Traverse Right Subtree
            cout << root->word << "->" << root->meaning << "\n"; // Print Data Of Root Node
        }
    }

    void Level(Node* root) // LevelOrder Traversal (Breadth-First)
    {
        Queue Q1; // Object Of Queue Class
        Node* n1; // Node Created
        Q1.insert(root); // Insert Data Into Queue
        while (!Q1.isEmpty()) // Loop Until Queue Is Empty
        {
            n1 = Q1.Delete(); // DeQueue
            cout << n1->word << "->" << n1->meaning << "\n"; // Display Word And Meaning
            if (n1->left) // If There Is Data On Left Side Of Binary Tree
                Q1.insert(n1->left); // EnQueue Left Sided Words
            if (n1->right) // If There Is Data On Right Side Of Binary Tree
                Q1.insert(n1->right); // EnQueue Right Sided Words
        }
    }

    void LexicographicalOrder(Node* root) // Function To Perform Lexicographical Traversal Of
A Binary Tree
    {
        if (root == NULL) // If Root Node Is Null
        {
            return; // Nothing To Traverse
        }
```

```cpp
        if (root->left) // If The Root Has Left Child
            LexicographicalOrder(root->left); // Traverse Left Subtree
        cout << root->word << "->" << root->meaning << "\n"; // Print Data Of Root Node
        if (root->right) // If The Root Has Right Child
            LexicographicalOrder(root->right); // Traverse Right Subtree
    }
};

int main() // Main Driver Program
{
    Node* root = NULL; // Root Node
    string w; // String Type Data
    Dictionary D1; // Object Of Dictionary Class
    char ch; // Character Type Data
    int i; // Integer Type Data

    do { // Do-While Loop
        cout << "\n1. Create Dictionary\n";
        cout << "2. Add New Word To Dictionary\n";
        cout << "3. Update Meaning Of Word\n";
        cout << "4. Delete A Word From Dictionary\n";
        cout << "5. Display Dictionary InOrder\n";
        cout << "6. Display Dictionary PreOrder\n";
        cout << "7. Display Dictionary PostOrder\n";
        cout << "8. Display Dictionary LevelOrder\n";
        cout << "9. Display Dictionary Lexicographical Order\n";
        cout << "10. Display Tree Like Structure\n";
        cout << "\n\nEnter Your Choice: ";
        cin >> i; // Taking Choice From User
        switch (i)
        {
        case 1: root = D1.Create(); break;
        case 2: D1.Add(root); break;
        case 3: D1.Update(root); break;
        case 4: cout << "Enter Word To Delete: ";
            cin >> w;
            D1.Delete(root, w); break;
        case 5: D1.InOrder(root); break;
        case 6: D1.PreOrder(root); break;
        case 7: D1.PostOrder(root); break;
        case 8: D1.Level(root); break;
        case 9: D1.LexicographicalOrder(root); break;
        case 10: D1.Display(root, root, 1); break;
        }

        cout << "\nPress 1 For Main & 2 To Exit ";
        cin >> ch; // Taking Choice From User
    } while (ch == '1'); // Loop Keeps Running As Far User Types '1'
    _getch();
    return 0;
}
```

OUTPUT:

C:\Users\SHAYAN\source\repos\Project53\Debug\Project53.exe

```
1. Create Dictionary
2. Add New Word To Dictionary
3. Update Meaning Of Word
4. Delete A Word From Dictionary
5. Display Dictionary InOrder
6. Display Dictionary PreOrder
7. Display Dictionary PostOrder
8. Display Dictionary LevelOrder
9. Display Dictionary Lexicographical Order
10. Display Tree Like Structure


Enter Your Choice: 1
Enter Word: apple
Enter Meaning: fruit
Do You Want To Continue? (Y/N) y
Enter Word: ball
Enter Meaning: ball
Do You Want To Continue? (Y/N) y
Enter Word: app
Enter Meaning: application
Do You Want To Continue? (Y/N) n

Press 1 For Main & 2 To Exit
```

C:\Users\SHAYAN\source\repos\Project53\Debug\Project53.exe

```
Press 1 For Main & 2 To Exit 1

1. Create Dictionary
2. Add New Word To Dictionary
3. Update Meaning Of Word
4. Delete A Word From Dictionary
5. Display Dictionary InOrder
6. Display Dictionary PreOrder
7. Display Dictionary PostOrder
8. Display Dictionary LevelOrder
9. Display Dictionary Lexicographical Order
10. Display Tree Like Structure


Enter Your Choice: 2
Enter Word: cat
Enter Meaning: animal
Word Successfully Added To Dictionary

Press 1 For Main & 2 To Exit 1
```

C:\Users\SHAYAN\source\repos\Project53\Debug\Project53.exe

```
Enter Your Choice: 3
Enter Word To Update: ball
Enter New Meaning: sportsitem
Meaning Updated Successfully

Press 1 For Main & 2 To Exit 1

1. Create Dictionary
2. Add New Word To Dictionary
3. Update Meaning Of Word
4. Delete A Word From Dictionary
5. Display Dictionary InOrder
6. Display Dictionary PreOrder
7. Display Dictionary PostOrder
8. Display Dictionary LevelOrder
9. Display Dictionary Lexicographical Order
10. Display Tree Like Structure


Enter Your Choice: 4
Enter Word To Delete: app

Press 1 For Main & 2 To Exit 1

1. Create Dictionary
```

C:\Users\SHAYAN\source\repos\Project53\Debug\Project53.exe

```
Enter Your Choice: 5
apple->fruit
ball->sportsitem
cat->animal

Press 1 For Main & 2 To Exit 1

1. Create Dictionary
2. Add New Word To Dictionary
3. Update Meaning Of Word
4. Delete A Word From Dictionary
5. Display Dictionary InOrder
6. Display Dictionary PreOrder
7. Display Dictionary PostOrder
8. Display Dictionary LevelOrder
9. Display Dictionary Lexicographical Order
10. Display Tree Like Structure


Enter Your Choice: 6
apple->fruit
ball->sportsitem
cat->animal

Press 1 For Main & 2 To Exit 1
```

C:\Users\SHAYAN\source\repos\Project53\Debug\Project53.exe

```
Enter Your Choice: 7
ball->sportsitem
cat->animal
apple->fruit

Press 1 For Main & 2 To Exit 1

1. Create Dictionary
2. Add New Word To Dictionary
3. Update Meaning Of Word
4. Delete A Word From Dictionary
5. Display Dictionary InOrder
6. Display Dictionary PreOrder
7. Display Dictionary PostOrder
8. Display Dictionary LevelOrder
9. Display Dictionary Lexicographical Order
10. Display Tree Like Structure


Enter Your Choice: 9
apple->fruit
ball->sportsitem
cat->animal

Press 1 For Main & 2 To Exit 1
```

C:\Users\SHAYAN\source\repos\Project53\Debug\Project53.exe

```
Press 1 For Main & 2 To Exit 1

1. Create Dictionary
2. Add New Word To Dictionary
3. Update Meaning Of Word
4. Delete A Word From Dictionary
5. Display Dictionary InOrder
6. Display Dictionary PreOrder
7. Display Dictionary PostOrder
8. Display Dictionary LevelOrder
9. Display Dictionary Lexicographical Order
10. Display Tree Like Structure


Enter Your Choice: 10

                       cat->animal

            ball->sportsitem

Root->apple->fruit

Press 1 For Main & 2 To Exit 2
```