Experiment B11

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;

struct node {
    char k[20]; //array to store keyword
    char m[20]; //array to store meaning;
    class node *left;
    class node *right;
};

class dict {
    public:
    node *root; //root poitner
    void create(); //to create bst
    void disp(node *); //to display bst;

    void insert(node *root, node *temp); // to insert new node
    int search(node *,char[]); // to search any node;
    int update(node *, char[]); // to change  alue of any node
    node *del(node *, char[]);
    node * min(node *);
};

void dict::create() {
    class node * temp;
    int ch;
    do {
    temp = new node;
    cout << "\nEnter keyword: ";
    cin >> temp->k;
    cout << "\nEnter meaning: ";
    cin >> temp->m;
    temp->left = NULL;
    temp->right = NULL;
    if(root == NULL) {
        root = temp; //if no root node then make temp as root node
    } else {
        insert(root, temp);// if root is present then call insert function to insrt node to appropriate
position
    }
    cout << "\nDo you want to add more (y=1/n=0) ";
    cin >> ch;
```

```
    } while(ch == 1); //while choice is 1, add no. of nodes
    }

    void dict::insert(node *root, node *temp) {
     //to insert new node n bst when root node is available
        if(strcmp(temp->k, root->k) < 0)
        //compare keyboard of temp node root node & if it is less than 0 then need to inset left
        {

            if(root->left == NULL) //if left node is null then insert temp otherwise call insert
function to search position in left subtree
            {
                root->left = temp;
            } else {
                insert(root->left, temp);
            }
        }
            else {
                if(root->right == NULL) {
                    root->right = temp;
                } else {
                    insert(root->right, temp);
                }
            }
        }
        void dict::disp(node *root) //to display record
        {
            if(root != NULL) {
                disp(root->left); // go towards extreme left node
                cout << "\n Key word: " << root->k; //print that node
                cout << "\t Meaning: " << root->m;
                disp(root->right);
            }
        }

        int dict::search(node *root, char k[20]) { // to search an element
            int c = 0; //to count no. of comparisons
            while(root != NULL) // until root becomes null
            {
                c++;
                if(strcmp (k, root->k) == 0) {
                    cout << "\n No of comparisons: " << c; //if matches, return 1 and print count
                    return 1;
                }
                if(strcmp(k, root->k) < 0)
                    root = root->left;
```

```
            if(strcmp(k, root->k) > 0) //if comparison is greater than zero then search towards
right subtree
            {
                root = root->right;
            }

        }
        return -1;
}

        int dict::update(node * root, char k[20]) // to update any entry
        {
            while(root != NULL) {
                if(strcmp(k, root->k) == 0) //compare search key with keyword of root node
                {
                    cout << "\n Enter new meaning of keyword" << root->k;
                    cin >> root-> m; //if found then update the meaning of specified keyword &
return 1 as found
                    return 1;
                }

                if(strcmp(k, root->k) < 0)
                    root=root->left;
                if(strcmp(k, root->k) > 0)
                    root = root->right;
            }
            return -1;
        }

        node *dict::del(node * root, char k[20])//to delete any entry
        {
            node *temp;
            if(root == NULL) { //if root node is not present

                cout<< "\nElement not found"; //no element is found
                return root;
            }

            if(strcmp(k, root->k) < 0) { //if keyword is less than root node

                root->left = del(root->left, k); //apply delete function on left element
                return root;
            }

            if(strcmp(k, root->k) > 0) { //if keyword is less than root node

                root->right = del(root->right, k); //apply delete function on right element
                return root;
```

```
        }

        if(root->right==NULL && root->left==NULL) //if that root node is leaf node
        {
            temp = root;
            delete temp;
            return NULL;

        }

        if(root->right==NULL) {
            temp = root;
            root = root->left;
            delete temp;
            return root;
        } else if(root->left ==NULL) {
            temp = root;
            root = root->right;
            delete temp;
            return root;
        }
        temp = min(root->right); // if condition get unstatisfied that node is not a leaf node ,
node is not having a left child, right child
        strcpy(root->k, temp->k);
        root->right = del(root->right, temp->k);
        return root;
    }

    node * dict::min(node *q) // find min element on extreme left //on the right subtree find
the min element
    {
        while(q->left != NULL);
        {
            q = q->left; // until reach the leaf node
        }

        return q;
    }

int main( ){
    int ch;
    dict d; // object of class dictionary
    d.root = NULL; //setting initially root to the null
    do {
        cout << "\nMenu\n1.Create\n2.Disp\n3.Search\n4.Update\n5.Delete\nEnter your
choise: ";
        cin >> ch;
        switch(ch)
```

```
{
    case 1: d.create(); // to create bst;
    break;

    case 2: if(d.root == NULL) // nothing to display in a tree
    {
        cout << "\nNo any keywrod";
    } else {
        d.disp(d.root);
    }
    break;

    case 3: if(d.root == NULL) // nthing to display in a tree
        {
        cout << "\nDictionary is empty, first add keywords then try again ";
    } else {
        cout << "\nEnter keyword which u want to search: ";
        char k[20];
        cin >> k; // take a choice to search;
        if(d.search(d.root, k) == 1) {
            cout << "\nKeyword found";
        }
        else
            cout << "\nKeyword not found ";
    }
        break;

    case 4:
        if(d.root == NULL) {
            cout << "\ndictionary is empty, first add keywords then try again ";
        } else {
            cout << "\nEnter keyword which meaning want to update";
            char k[20];
            cin >> k;
            if(d.update(d.root, k) == 1) //if function returns meaning is updated
                cout << "\nmeaning updated";
            else
                cout << "\nMeaning Not Found";
        }
        break;

    case 5:
        if(d.root == NULL) {
            cout << "\nDictionary is emtpy , first add keywords then try again ";
        } else {
            cout << "\nEnter keyword which u want to delete: ";
            char k[20];
            cin >> k;
```

```
                    if(d.root == NULL) {
                        cout << "\nno any keyword";
                    } else {
                        d.root = d.del(d.root, k);
                    }
                }
            }

        } while(ch<=5);


    return 0;
}
```