1.)

/\*

    Construct a expression tree from postfix expression

    ===================================================

    Functions:

        1.Create

        2.Inorder Traversal (Recursive)

        3.Preorder Traversal (Recursive)

        4.Postorder Traversal (Recursive)

        5.Inorder Traversal(N.Recursive)

        6.Preorder Traversal(N.Recursive)

        7.Postorder Traversal(N.Recursive)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*/

#include<iostream>

using namespace std;

#include<stdlib.h>

#define MAX 30

//Check the Character is Operand or Not

int isoperand(char ch)

{

    if((ch>='A'  && ch<='Z')||(ch>='a'  && ch<='z')||(ch>='0'  && ch<='9'))

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

//Check the Character is Operator or Not

int isoperator(char ch)

{

    if(ch=='$'||ch=='^'||ch=='+'||ch=='-'||ch=='\*'||ch=='/')

        return 1;

    else

        return 0;

}

//Declare a self Referential Structure for Tree Nodes

struct Treenode

{

    Treenode \*lchild;

    char data;

    Treenode \*rchild;

};

//Declare Class

class ET

{

        Treenode \*root;

    public:

        ET();//Constructor

        void create(char postfix[MAX]);

        void inorder();//Wrapper Function as accessing private member root is not accessible in the main function

        void inorder(Treenode \*);

        void preorder();//Wrapper Function

        void preorder(Treenode \*);

        void postorder();//Wrapper Function

        void postorder(Treenode \*);

        void inorder\_nrc();

        void preorder\_nrc();

        void postorder\_nrc();

};

//Initialize root to NULL

ET::ET()

{

    root=NULL;

}

//For Building Expression Tree from prefix Expression

void ET::create(char prefix[MAX])

{

    //Declare Stack

    Treenode \*stack[MAX];

    int top=-1;

    int i,len,val;

    char ch;

    Treenode \*temp;

    //Calculate length of the prefix expression

    for(i=0;prefix[i]!='\0';i++);

    len=i-1;

    //Scan the prefix expression from right to left

    for(i=len;i>=0;i--)

    {

        ch=prefix[i];

        //Create and Initialize a new node in the memory

        temp=new Treenode;

        temp->lchild=NULL;

        temp->data=ch;

        temp->rchild=NULL;

        //If the character is operand push the address on stack

        if(isoperand(ch))

        {

            stack[++top]=temp;

        }

        //If the character is operator pop two addresses from stack and built a sub tree,

        //Push the parents address back on the stack

        else if(isoperator(ch))

        {

            temp->lchild=stack[top--];

            temp->rchild=stack[top--];

            stack[++top]=temp;

        }

        else

        {

            cout<<"\nWrong expression tree";

            cout<<"\nNode cannot be created";

            exit(0);

        }

    }

    //Once scanning the expression from right to left is over pop the address from stack and store it as root address

    root=stack[top--];

}

void ET::inorder()

{

    if(root)

        inorder(root);

    else

        cout<<"\nEmpty expression tree";

}

void ET::inorder(Treenode \*root)

{

    //LDR

    if(root)

    {

        //Traverse Left

        inorder(root->lchild);

        //Print Data

        cout<<root->data<<" ";

        //Traverse Right

        inorder(root->rchild);

    }

}

void ET::preorder()

{

    if(root)

        preorder(root);

    else

        cout<<"\nEmpty expression tree";

}

void ET::preorder(Treenode \*root)

{

    //DLR

    if(root)

    {

        //Print Data

        cout<<root->data<<" ";

        //Traverse Left

        preorder(root->lchild);

        //Traverse Right

        preorder(root->rchild);

    }

}

void ET::postorder()

{

    if(root)

        postorder(root);

    else

        cout<<"\nEmpty expression tree";

}

void ET::postorder(Treenode \*root)

{

    //DLR

    if(root)

    {

        //Traverse Left

        postorder(root->lchild);

        //Traverse Right

        postorder(root->rchild);

        //Print Data

        cout<<root->data<<" ";

    }

}

void ET::inorder\_nrc()

{

    Treenode \*curr=root;

    Treenode \*stack[MAX];

    int top=-1;

    while(1)

    {

        while(curr!=NULL)

        {

            stack[++top]=curr;

            curr=curr->lchild;//Traverse Left

        }

        if(top!=-1)

        {

            curr=stack[top--];

            cout<<curr->data<<" "; //Print Data

            curr=curr->rchild;  //Traverse Right

        }

        else

            break;

    }

}

void ET::preorder\_nrc()

{

    Treenode \*curr=root;

    Treenode \*stack[MAX];

    int top=-1;

    while(1)

    {

        while(curr!=NULL)

        {

            cout<<curr->data<<" "; //Print Data

            stack[++top]=curr;

            curr=curr->lchild;  //Traverse Left

        }

        if(top!=-1)

        {

            curr=stack[top--];

            curr=curr->rchild;  //Traverse Right

        }

        else

            break;

    }

}

void ET::postorder\_nrc()

{

    Treenode \*curr=root;

    Treenode \*stack[MAX];

    int top=-1,flag[MAX],f;

    while(1)

    {

        if(curr!=NULL)

        {

            stack[++top]=curr;

            flag[top]=0;

            curr=curr->lchild;  //Traverse Left

        }

        else

        {

                if(top!=-1)

                {

                    f=flag[top];

                    curr=stack[top--];

                    if(f==0)

                    {

                        stack[++top]=curr;

                        flag[top]=1;

                        curr=curr->rchild;  //Traverse Right

                    }

                    else if (f==1)

                    {

                        cout<<curr->data<<" "; //Print Data

                        curr=NULL;

                    }

                }

                else

                    break;

        }

    }

}

int main()

{

    int ch;

    char prefix[MAX];

    ET e;

    cout<<"\nEnter a prefix expression";

    cin>>prefix;

    while(1)

    {

        cout<<"\n\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*";

        cout<<"\n1.Create a expression tree\n2.Inorder Traversal (Recursive)\n3.Preorder Traversal (Recursive)";

        cout<<"\n4.Postorder (Recursive)\n5.Inorder Traversal(Non Recursive)\n6.Preorder Traversal(Non Recursive)";

        cout<<"\n7.Post order Traversal(Non Recursive)\n8.Exit";

        cout<<"\nEnter your choice";

        cin>>ch;

        switch(ch)

        {

            case 1: e.create(prefix);

                cout<<"\nExpression Tree Created from Prefix Expression\n";

                break;

            case 2: e.inorder(); //Call the wrapper function

                break;

            case 3: e.preorder(); //Call the wrapper function

                break;

            case 4: e.postorder(); //Call the wrapper function

                break;

            case 5: e.inorder\_nrc();

                break;

            case 6: e.preorder\_nrc();

                break;

            case 7: e.postorder\_nrc();

                break;

            case 8:exit(0);

        }

    }

}

2)

#include <iostream>

#include<string>

using namespace std;

class dictionary;

class node

{

string word,meaning;

node \*left,\*right;

public:

friend class dictionary;

node()

{

left=NULL;

right=NULL;

}

node(string word, string meaning)

{

this->word=word;

this->meaning=meaning;

left=NULL;

right=NULL;

}

};

class dictionary

{

node \*root;

public:

dictionary()

{

root=NULL;

}

void create();

void inorder\_rec(node \*rnode);

void postorder\_rec(node \*rnode);

void inorder()

{

inorder\_rec(root);

}

void postorder();

bool insert(string word,string meaning);

int search(string key);

void deleted(string todel);

};

int dictionary::search(string key)

{

node \*tmp=root;

int count;

if(tmp==NULL)

{

return -1;

}

if(root->word==key)

return 1;

while(tmp!=NULL)

{

if((tmp->word)>key)

{

tmp=tmp->left;

count++;

}

else if((tmp->word)<key)

{

tmp=tmp->right;

count++;

}

else if(tmp->word==key)

{

return ++count;

}

}

return -1;

}

void dictionary::postorder()

{

postorder\_rec(root);

}

void dictionary::postorder\_rec(node \*rnode)

{

if(rnode)

{

postorder\_rec(rnode->right);

cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;

postorder\_rec(rnode->left);

}

}

void dictionary::create()

{

int n;

string wordI,meaningI;

cout<<"\n How many Word to insert?:\n";

cin>>n;

for(int i=0;i<n;i++)

{

cout<<"\n Enter Word: ";

cin>>wordI;

cout<<"\n Enter Meaning: ";

cin>>meaningI;

insert(wordI,meaningI);

}

}

void dictionary::inorder\_rec(node \*rnode)

{

if(rnode)

{

inorder\_rec(rnode->left);

cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;

inorder\_rec(rnode->right);

}

}

bool dictionary::insert(string word, string meaning)

{

node \*p=new node(word, meaning);

if(root==NULL)

{

root=p;

return true;

}

node \*cur=root;

node \*par=root;

while(cur!=NULL) //traversal

{

if(word>cur->word)

{par=cur;

cur=cur->right;

}

else if(word<cur->word)

{

par=cur;

cur=cur->left;

}

else

{

cout<<"\n Word is already in the dictionary.";

return false;

}

}

if(word>par->word) //insertion of node

{

par->right=p;

return true;

}

else

{

par->left=p;

return true;

}

}

void dictionary::deleted(string todel)

{

node \*par = NULL, \*cur = NULL, \*temp = NULL;

int flag = 0, res = 0;

if (!root) {

cout<<"BST is not present!!\n";

return;

}

cur = root;

while (1) {

res = strcasecmp(cur->word, todel);

if (res == 0)

break;

flag = res;

par = cur;

cur = (res > 0) ? cur->left : cur->right;

if (cur == NULL)

return;

}

/\* deleting leaf node \*/

if (cur->right == NULL) {

if (cur == root && cur->left == NULL) {

delete(cur);

root = NULL;

return;

} else if (cur == root) {

root = cur->left;

delete (cur);

return;

}

flag > 0 ? (par->left = cur->left) :

(par->right = cur->left);

} else {

/\* delete node with single child \*/

temp = cur->right;

if (!temp->left) {

temp->left = cur->left;

if (cur == root) {

root = temp;

delete(cur);

return;

}

flag > 0 ? (par->left = temp) :

(par->right = temp);

} else {

/\* delete node with two children \*/

struct BSTnode \*successor = NULL;

while (1) {

successor = temp->left;

if (!successor->left)

break;

temp = successor;

}

temp->left = successor->right;

successor->left = cur->left;

successor->right = cur->right;

if (cur == root) {

root = successor;

delete(cur);

return;

}

(flag > 0) ? (par->left = successor) :

(par->right = successor);

}

}

delete (cur);

return;

}

int main()

{

string word;

dictionary months;

int ch;

char ch3;

do

{

cout<<"\nEnter your choice:\n1.Create\n2.Sorting\n3.Search\n4.Remove\n5.Exit\n";

cin>>ch;

switch(ch)

{

case 1:months.create();

break;

case 2: cout<<"\nEnter your choice\n1. Ascending order \n2.Descending Order\n";

int ch1;

cin>>ch1;

switch(ch1)

{

case 1: cout<<"\n Ascending order\n";

months.inorder();

break;

case 2: cout<<"\n Descending order:\n";

months.postorder();

break;

}

break;

case 3: {cout<<"\n Enter word to search: ";

cin>>word;

int comparisons=months.search(word);

if(comparisons==-1)

{

cout<<"\n Not found word";

}

else

{

cout<<"\n "<<word<<" found in "<<comparisons<<" comparisons";

}}

break;

case 4: string n;

cout << "\nEnter the element to be deleted:";

cin >> n;

// months.deleted(n);

break;

}

cout<<"\nDo you want to continue??\n";

cin>>ch3;

}while(ch3=='y');

return 0;

}

3)

/\*   Ass 3: Create BST ,its mirror image ,display it levelwise,display leaf nodes,find height of BST \*/

/\*   Author : Vrushali V. Kondhalkar                                                                 \*/

#include <iostream>

using namespace std;

struct  tnode

{

    int info;   // node value

    struct tnode \*left; //left child pointer

    struct tnode \*right; //right child pointer

};

class bintree

{

    tnode \*root;

    public:

         bintree()  //constructor

        {

        root =NULL;

        }

          int isEmpty() //function to check if tree is empty

          {

              if(root==NULL)

                  return 1;

              return 0;

          }

         void printGivenLevel(tnode\* root, int level);

         void printLevelOrder(); //print levelwise

         void maxDepth1(); //prints height of tree wrapper

         void plnode(); //print leaf nodes wrapper

         void printLeafNodes(tnode\* root);

         int  maxDepth(tnode\* node);

         void insert(int val); //insertion of node in tree

         void mirror1();

         void mirror(struct tnode \*);//create mirror image

};

void bintree::plnode()

{

    cout<<endl<<"leaf nodes are:";

    printLeafNodes(root);

}

void bintree:: printLeafNodes(tnode \*root)

{

    // if node is null, return

    if (root == NULL)

        return;

    // if node is leaf node, print its data

    if (root->left == NULL && root->right == NULL)

    {

        cout << root->info << " ";

        return;

    }

    // if left child exists, check for leaf

    // recursively

    if (root->left!= NULL)

       printLeafNodes(root->left);

    // if right child exists, check for leaf

    // recursively

    if (root->right!= NULL)

       printLeafNodes(root->right);

}

void bintree::insert(int v)

{

    tnode \*ctnode = new tnode;

    tnode \*parent;

    ctnode->info =v;

    ctnode->left=NULL;

    ctnode->right=NULL;

    parent=NULL;

    if(isEmpty())

    {

        root=ctnode;

    }

    else

    {

        tnode \*p=root;

        while(p!=NULL)

        {

            parent =p;

            if(v>p->info)

                p=p->right;

            else

                p=p->left;

        }

        if(v<parent->info)

            parent->left=ctnode;

        else

            parent->right=ctnode;

    }

}

void bintree::printGivenLevel(tnode\* root, int level)

{

    if (root == NULL)

        return;

    if (level == 1)

        cout<<root->info<<" ";

    else if (level > 1)

    {

        printGivenLevel(root->left, level - 1);

        printGivenLevel(root->right, level - 1);

    }

}

void bintree::printLevelOrder()

{

    int h = maxDepth(root);

    int i;

    cout<<endl<<"level wise"<<endl;

    for (i = 1; i <= h; i++)

    {

        printGivenLevel(root, i);

        cout<<endl;

    }

}

void bintree::  maxDepth1()

{

int d = maxDepth(root);

cout<<"height:"<<d;

}

    int bintree:: maxDepth(tnode\* node)

{

    if (node == NULL)

        return 0;

    else

    {

        /\* compute the depth of each subtree \*/

        int lDepth = maxDepth(node->left);

        int rDepth = maxDepth(node->right);

        /\* use the larger one \*/

        if (lDepth > rDepth)

            return (lDepth + 1);

        else

            return (rDepth + 1);

    }

}

    void bintree:: mirror1()

    {

        cout<<endl<<"mirror image is :";

        mirror(root);

        printLevelOrder();

    }

    void bintree:: mirror(struct tnode \*node)

    {

        if(node==NULL)

        {

            return;

        }

        else

        {

            struct tnode \*temp;

            mirror(node->left);

            mirror(node->right);

            temp=node->left;

            node->left=node->right;

            node->right=temp;

        }

    }

int main()

{

    bintree b;

    int ch,ch1,n,d;

    do

     {

        cout<<"1.Create Binary Search Tree (BST)"<<endl;

        cout<<"2.Insert a node in BST "<<endl;

        cout<<"3.Display BST levelwise"<<endl;

        cout<<"4.Create Mirror image of BST"<<endl;

        cout<<"5.Display leaf nodes"<<endl;

        cout<<"6.Display height of BST"<<endl;

        cout<<"Enter your choice : ";

        cin>>ch1;

        switch(ch1)

        {

            case 1: cout<<"How many nodes in BST : ";

                    cin>>n;

                    for(int i=1;i<=n;i++)

                    {

                        cout<<"Enter data for node "<<i;

                        cin>>d;

                        b.insert(d);

                    }

                    break;

            case 2: cout<<"Enter data for node";

                    cin>>d;

                    b.insert(d);

                    break;

            case 3:

                    b.printLevelOrder();

                    break;

            case 4:

                    b.mirror1();

                    break;

            case 5:

                    b.plnode();

                    break;

            case 6:

                    b.maxDepth1();

                    break;

            default:

                    cout<<endl<<"Enter valid choice";

                    break;

        }

cout<<endl<<"Do you want to continue? Press 1 to continue else 0 :";

cin>>ch;

}while(ch==1);

return 0;

}

#include <iostream>

using namespace std;

class node

{

public:

    int data;

    int lbit,rbit;

    node \*left,\*right;

};

class TBT

{

    node \*root,\*dummy;

    public:

        void create(int num);

        void insert(node \*trav,node \*temp);

        void preorder();

        void inorder(node \*trav,node \*dummy);

        void postorder(node \*trav,node \*dummy);

        void display();

        TBT()

        {

            root=NULL;

            dummy=NULL;

        }

}\*root,\*dummy;

void TBT::create(int num)

{

    node \*trav,\*temp;

    temp=new node();

    temp->data=num;

    temp->lbit=0;

    temp->rbit=0;

    if(root==NULL)

    {

        dummy=new node;

        dummy->data=-9999;

        dummy->lbit=1;

        dummy->rbit=1;

        dummy->left=temp;

        dummy->right=dummy;

        temp->left=dummy;

        temp->right=dummy;

        root=temp;

    }

    else

    {

        trav=root;

        insert(trav,temp);

    }

}

void TBT::insert(node \*trav,node \*temp)

{

    if(temp->data<trav->data)

    {

        if(trav->lbit==0)

        {

            temp->left=trav->left;

            temp->right=trav;

            trav->left=temp;

            trav->lbit=1;

        }

        else

        {

            insert(trav->left,temp);

        }

    }

    if(temp->data>trav->data)

    {

        if(trav->rbit==0)

        {

            temp->right=trav->right;

            temp->left=trav;

            trav->right=temp;

            trav->rbit=1;

        }

        else

        {

            insert(trav->right,temp);

        }

    }

}

void TBT::preorder()

{

    node \*trav;

    trav=root;

    while(trav!=dummy)

    {

        cout<<trav->data<<endl;

        if(trav->lbit==1)

        {

            trav=trav->left;

        }

        else

        {

            while(trav->rbit==0 && trav->right!=dummy)

            {

                trav=trav->right;

            }

            trav=trav->right;

        }

    }

}

void TBT::inorder(node \*trav,node \*dummy)

{

    while(trav!=dummy)

    {

        while(trav->lbit==1)

        {

            trav=trav->left;

        }

        cout<<trav->data<<endl;

        while(trav!=dummy)

        {

            if(trav->rbit==1)

            {

                trav=trav->right;

                while(trav->lbit==1)

                {

                    trav=trav->left;

                }

                cout<<trav->data<<endl;

            }

            else

            {

                while(trav->rbit==0)

                {

                    trav=trav->right;

                    if(trav==dummy)

                    {

                        break;

                    }

                    cout<<trav->data<<endl;

                }

            }

        }

    }

}

void TBT::postorder(node \*trav,node \*dummy)

{

    if(root==NULL)

        cout<<"Empty Tree";

    else

    {

        int a[20],i=0;

        while (trav!=dummy)

        {

            a[i]=trav->data;

            i++;

            if(trav->rbit==1)

            {

                trav=trav->right;

            }

            else

            {

                while (trav->lbit==0 && trav->left!=dummy)

                {

                    trav=trav->left;

                }

                trav=trav->left;

            }

        }

        int n=i-1;

        for(i=n;i>=0;i--)

        {

            cout<<a[i]<<endl;

        }

    }

}

void TBT::display()

{

    int n,i,data;

    cout<<"\n Enter the number of nodes \n";

    cin>>n;

    cout<<"\n Enter the data in the Nodes \n";

    for (i=0;i<n;i++)

    {

        cin>>data;

        create(data);

    }

    cout<<"\n Preorder: \n";

    preorder();

    cout<<"\n Inorder: \n";

    inorder(root,dummy);

    cout<<"\n Postorder: \n";

    postorder(root,dummy);

}

int main()

{

    TBT T;

    T.display();

    return 0;

}

5)

// Program to print BFS traversal from a given

// source vertex. BFS(int s) traverses vertices

// reachable from s.

#include <bits/stdc++.h>

using namespace std;

// This class represents a directed graph using

// adjacency list representation

class Graph {

    int V; // No. of vertices

    // Pointer to an array containing adjacency

    // lists

    vector<list<int> > adj;

public:

    Graph(int V); // Constructor

    // function to add an edge to graph

    void addEdge(int v, int w);

    // prints BFS traversal from a given source s

    void BFS(int s);

};

Graph::Graph(int V)

{

    this->V = V;

    adj.resize(V);

}

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::BFS(int s)

{

    // Mark all the vertices as not visited

    vector<bool> visited;

    visited.resize(V, false);

    // Create a queue for BFS

    list<int> queue;

    // Mark the current node as visited and enqueue it

    visited[s] = true;

    queue.push\_back(s);

    while (!queue.empty()) {

        // Dequeue a vertex from queue and print it

        s = queue.front();

        cout << s << " ";

        queue.pop\_front();

        // Get all adjacent vertices of the dequeued

        // vertex s. If a adjacent has not been visited,

        // then mark it visited and enqueue it

        for (auto adjacent : adj[s]) {

            if (!visited[adjacent]) {

                visited[adjacent] = true;

                queue.push\_back(adjacent);

            }

        }

    }

}

// Driver program to test methods of graph class

int main()

{

    // Create a graph given in the above diagram

    Graph g(4);

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "

        << "(starting from vertex 2) \n";

    g.BFS(2);

    return 0;

}

6)

// C++ program to print DFS traversal from

// a given vertex in a given graph

#include <bits/stdc++.h>

using namespace std;

// Graph class represents a directed graph

// using adjacency list representation

class Graph {

public:

    map<int, bool> visited;

    map<int, list<int> > adj;

    // function to add an edge to graph

    void addEdge(int v, int w);

    // DFS traversal of the vertices

    // reachable from v

    void DFS(int v);

};

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::DFS(int v)

{

    // Mark the current node as visited and

    // print it

    visited[v] = true;

    cout << v << " ";

    // Recur for all the vertices adjacent

    // to this vertex

    list<int>::iterator i;

    for (i = adj[v].begin(); i != adj[v].end(); ++i)

        if (!visited[\*i])

            DFS(\*i);

}

// Driver's code

int main()

{

    // Create a graph given in the above diagram

    Graph g;

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal"

            " (starting from vertex 2) \n";

    // Function call

    g.DFS(2);

    return 0;

}

7)

// C++ program for Dijkstra's single source shortest path

// algorithm. The program is for adjacency matrix

// representation of the graph

#include <iostream>

using namespace std;

#include <limits.h>

// Number of vertices in the graph

#define V 9

// A utility function to find the vertex with minimum

// distance value, from the set of vertices not yet included

// in shortest path tree

int minDistance(int dist[], bool sptSet[])

{

// Initialize min value

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// A utility function to print the constructed distance

// array

void printSolution(int dist[])

{

cout << "Vertex \t Distance from Source" << endl;

for (int i = 0; i < V; i++)

cout << i << " \t\t\t\t" << dist[i] << endl;

}

// Function that implements Dijkstra's single source

// shortest path algorithm for a graph represented using

// adjacency matrix representation

void dijkstra(int graph[V][V], int src)

{

int dist[V]; // The output array. dist[i] will hold the

// shortest

// distance from src to i

bool sptSet[V]; // sptSet[i] will be true if vertex i is

// included in shortest

// path tree or shortest distance from src to i is

// finalized

// Initialize all distances as INFINITE and stpSet[] as

// false

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of

// vertices not yet processed. u is always equal to

// src in the first iteration.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

// Update dist value of the adjacent vertices of the

// picked vertex.

for (int v = 0; v < V; v++)

// Update dist[v] only if is not in sptSet,

// there is an edge from u to v, and total

// weight of path from src to v through u is

// smaller than current value of dist[v]

if (!sptSet[v] && graph[u][v]

&& dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// print the constructed distance array

printSolution(dist);

}

// driver's code

int main()

{

/\* Let us create the example graph discussed above \*/

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

// Function call

dijkstra(graph, 0);

return 0;

}

8) Kruskal

class Graph:

    def \_\_init\_\_(self, vertices):

        self.V = vertices

        self.graph = []

    def add\_edge(self, u, v, w):

        self.graph.append([u, v, w])

    # Search function

    def find(self, parent, i):

        if parent[i] == i:

            return i

        return self.find(parent, parent[i])

    def apply\_union(self, parent, rank, x, y):

        xroot = self.find(parent, x)

        yroot = self.find(parent, y)

        if rank[xroot] < rank[yroot]:

            parent[xroot] = yroot

        elif rank[xroot] > rank[yroot]:

            parent[yroot] = xroot

        else:

            parent[yroot] = xroot

            rank[xroot] += 1

    #  Applying Kruskal algorithm

    def kruskal\_algo(self):

        result = []

        i, e = 0, 0

        self.graph = sorted(self.graph, key=lambda item: item[2])

        parent = []

        rank = []

        for node in range(self.V):

            parent.append(node)

            rank.append(0)

        while e < self.V - 1:

            u, v, w = self.graph[i]

            i = i + 1

            x = self.find(parent, u)

            y = self.find(parent, v)

            if x != y:

                e = e + 1

                result.append([u, v, w])

                self.apply\_union(parent, rank, x, y)

        for u, v, weight in result:

            print("%d - %d: %d" % (u, v, weight))

g = Graph(6)

g.add\_edge(0, 1, 4)

g.add\_edge(0, 2, 4)

g.add\_edge(1, 2, 2)

g.add\_edge(1, 0, 4)

g.add\_edge(2, 0, 4)

g.add\_edge(2, 1, 2)

g.add\_edge(2, 3, 3)

g.add\_edge(2, 5, 2)

g.add\_edge(2, 4, 4)

g.add\_edge(3, 2, 3)

g.add\_edge(3, 4, 3)

g.add\_edge(4, 2, 4)

g.add\_edge(4, 3, 3)

g.add\_edge(5, 2, 2)

g.add\_edge(5, 4, 3)

g.kruskal\_algo()

// C++ program for the above approach

#include <bits/stdc++.h>

using namespace std;

// DSU data structure

// path compression + rank by union

class DSU {

int\* parent;

int\* rank;

public:

DSU(int n)

{

parent = new int[n];

rank = new int[n];

for (int i = 0; i < n; i++) {

parent[i] = -1;

rank[i] = 1;

}

}

// Find function

int find(int i)

{

if (parent[i] == -1)

return i;

return parent[i] = find(parent[i]);

}

// Union function

void unite(int x, int y)

{

int s1 = find(x);

int s2 = find(y);

if (s1 != s2) {

if (rank[s1] < rank[s2]) {

parent[s1] = s2;

}

else if (rank[s1] > rank[s2]) {

parent[s2] = s1;

}

else {

parent[s2] = s1;

rank[s1] += 1;

}

}

}

};

class Graph {

vector<vector<int> > edgelist;

int V;

public:

Graph(int V) { this->V = V; }

// Function to add edge in a graph

void addEdge(int x, int y, int w)

{

edgelist.push\_back({ w, x, y });

}

void kruskals\_mst()

{

// Sort all edges

sort(edgelist.begin(), edgelist.end());

// Initialize the DSU

DSU s(V);

int ans = 0;

cout << "Following are the edges in the "

"constructed MST"

<< endl;

for (auto edge : edgelist) {

int w = edge[0];

int x = edge[1];

int y = edge[2];

// Take this edge in MST if it does

// not forms a cycle

if (s.find(x) != s.find(y)) {

s.unite(x, y);

ans += w;

cout << x << " -- " << y << " == " << w

<< endl;

}

}

cout << "Minimum Cost Spanning Tree: " << ans;

}

};

// Driver code

int main()

{

Graph g(4);

g.addEdge(0, 1, 10);

g.addEdge(1, 3, 15);

g.addEdge(2, 3, 4);

g.addEdge(2, 0, 6);

g.addEdge(0, 3, 5);

// Function call

g.kruskals\_mst();

return 0;

}

9) Prims:

INF = 9999999

V = 5

G = [[0, 9, 75, 0, 0],

     [9, 0, 95, 19, 42],

     [75, 95, 0, 51, 66],

     [0, 19, 51, 0, 31],

     [0, 42, 66, 31, 0]]

selected = [0, 0, 0, 0, 0]

no\_edge = 0

selected[0] = True

print("Edge : Weight\n")

while (no\_edge < V - 1):

    minimum = INF

    x = 0

    y = 0

    for i in range(V):

        if selected[i]:

            for j in range(V):

                if ((not selected[j]) and G[i][j]):

                    # not in selected and there is an edge

                    if minimum > G[i][j]:

                        minimum = G[i][j]

                        x = i

                        y = j

    print(str(x) + "-" + str(y) + ":" + str(G[x][y]))

    selected[y] = True

    no\_edge += 1

// A C++ program for Prim's Minimum

// Spanning Tree (MST) algorithm. The program is

// for adjacency matrix representation of the graph

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 5

// A utility function to find the vertex with

// minimum key value, from the set of vertices

// not yet included in MST

int minKey(int key[], bool mstSet[])

{

// Initialize min value

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

// A utility function to print the

// constructed MST stored in parent[]

void printMST(int parent[], int graph[V][V])

{

cout << "Edge \tWeight\n";

for (int i = 1; i < V; i++)

cout << parent[i] << " - " << i << " \t"

<< graph[i][parent[i]] << " \n";

}

// Function to construct and print MST for

// a graph represented using adjacency

// matrix representation

void primMST(int graph[V][V])

{

// Array to store constructed MST

int parent[V];

// Key values used to pick minimum weight edge in cut

int key[V];

// To represent set of vertices included in MST

bool mstSet[V];

// Initialize all keys as INFINITE

for (int i = 0; i < V; i++)

key[i] = INT\_MAX, mstSet[i] = false;

// Always include first 1st vertex in MST.

// Make key 0 so that this vertex is picked as first

// vertex.

key[0] = 0;

// First node is always root of MST

parent[0] = -1;

// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum key vertex from the

// set of vertices not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = true;

// Update key value and parent index of

// the adjacent vertices of the picked vertex.

// Consider only those vertices which are not

// yet included in MST

for (int v = 0; v < V; v++)

// graph[u][v] is non zero only for adjacent

// vertices of m mstSet[v] is false for vertices

// not yet included in MST Update the key only

// if graph[u][v] is smaller than key[v]

if (graph[u][v] && mstSet[v] == false

&& graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

// Print the constructed MST

printMST(parent, graph);

}

// Driver's code

int main()

{

int graph[V][V] = { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

// Print the solution

primMST(graph);

return 0;

}

10) simple index file

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <iomanip>

using namespace std;

// Define the structure of the index record

struct IndexRecord

{

    char key[10];

    long offset;

};

// Define the size of the index record

const int RECORD\_SIZE = sizeof(IndexRecord);

// Function to add a new index record

void addRecord(fstream& indexFile)

{

    IndexRecord record;

    cout << "Enter the key: ";

    cin >> record.key;

    cout << "Enter the offset: ";

    cin >> record.offset;

    // Write the record to the end of the file

    indexFile.seekp(0, ios::end);

    indexFile.write(reinterpret\_cast<char\*>(&record), RECORD\_SIZE);

}

// Function to search for an index record

void searchRecord(fstream& indexFile)

{

    char key[10];

    IndexRecord record;

    cout << "Enter the key to search for: ";

    cin >> key;

    // Search the file for the record with the matching key

    indexFile.seekg(0, ios::beg);

    while (indexFile.read(reinterpret\_cast<char\*>(&record), RECORD\_SIZE))

    {

        if (strcmp(record.key, key) == 0)

        {

            cout << "Record found:" << endl;

            cout << "Key: " << record.key << endl;

            cout << "Offset: " << record.offset << endl;

            return;

        }

    }

    // If the key is not found, print an error message

    cout << "Record not found." << endl;

}

// Function to print all index records

void printRecords(fstream& indexFile)

{

    IndexRecord record;

    cout << setw(10) << "Key" << setw(10) << "Offset" << endl;

    cout << "-----------------------" << endl;

    // Read each record and print its key and offset

    indexFile.seekg(0, ios::beg);

    while (indexFile.read(reinterpret\_cast<char\*>(&record), RECORD\_SIZE)) {

        cout << setw(10) << record.key << setw(10) << record.offset << endl;

    }

}

int main()

{

    fstream indexFile("index.dat", ios::in | ios::out | ios::binary);

    // If the index file does not exist, create it

    if (!indexFile)

    {

        indexFile.open("index.dat", ios::out | ios::binary);

        indexFile.close();

        indexFile.open("index.dat", ios::in | ios::out | ios::binary);

    }

    int choice;

    do

    {

        cout << endl;

        cout << "1. Add a new record" << endl;

        cout << "2. Search for a record" << endl;

        cout << "3. Print all records" << endl;

        cout << "4. Exit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

            case 1:

                addRecord(indexFile);

                break;

            case 2:

                searchRecord(indexFile);

                break;

            case 3:

                printRecords(indexFile);

                break;

            case 4:

                break;

            default:

                cout << "Invalid choice. Please try again." << endl;

                break;

        }

    } while (choice != 4);

    indexFile.close();

    return 0;

}

11) employee sequential file

#include <iostream>

#include <fstream>

#include <iomanip>

#define SIZE 100

using namespace std;

class Employee

{

private:

    int empno;

    int salary;

    char name[SIZE];

public:

    void accept();

    void display();

    void modify();

    int geteno();

};

void Employee::accept()

{

    cout << "Enter the Employee number:- ";

    cin >> empno;

    cin.ignore();

    cout << "Enter the name of the Employee:- ";

    cin.getline(name, SIZE);

    cout << "Enter the salary:- ";

    cin >> salary;

}

void Employee::display()

{

    cout << "\t" << empno << "\t   " << name << "   \t" << salary << "\n";

}

void Employee::modify()

{

    cin.ignore();

    cout << "Enter new name of Employee:- ";

    cin.getline(name, SIZE);

    cout << "Enter the new salary:- ";

    cin >> salary;

}

int Employee::geteno()

{

    return empno;

}

int create(int);

void display(int);

int add(int);

void deleterec(int);

void modifyrec(int);

int main()

{

    int choice, i, flag, cntr = 0;

    while (1)

    {

        cout << "\nEmployees Database" << endl;

        cout << "1. Create" << endl;

        cout << "2. Display" << endl;

        cout << "3. Add" << endl;

        cout << "4. Delete" << endl;

        cout << "5. Modify" << endl;

        cout << "6. Exit program" << endl;

        cout << "\nEnter your choice:- ";

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            cntr = create(cntr);

            break;

        }

        case 2:

        {

            display(cntr);

            break;

        }

        case 3:

        {

            if (cntr == 0)

                cout << "\nPlease build a master table first" << endl;

            else

                cntr = add(cntr);

            break;

        }

        case 4:

        {

            if (cntr == 0)

                cout << "\nPlease build a master table first" << endl;

            else

                deleterec(cntr);

            break;

        }

        case 5:

        {

            if (cntr == 0)

                cout << "\nPlease build a master table first" << endl;

            else

                modifyrec(cntr);

            break;

        }

        case 6:

            return 0;

        default:

            cout << "Error in choice,try again" << endl;

        }

    }

    return 0;

}

int create(int cntr)

{

    int eno;

    Employee m;

    ofstream fout("recs.dat", ios::binary | ios::out);

    cntr = 0;

    char ask = 'y';

    while (ask == 'y' || ask == 'Y')

    {

        m.accept();

        fout.write((char \*)&m, sizeof(m));

        cntr++;

        cout << "\nDo you want to add another record?(y/n):- ";

        cin >> ask;

    }

    fout.close();

    return cntr;

}

void display(int cntr)

{

    int i;

    Employee m;

    if (cntr == 0)

        cout << "\nPlease build a master table first" << endl;

    else

    {

        ifstream fin("recs.dat", ios::binary | ios::in);

        cout << "\n-----------------------------------------------------------------" << endl;

        cout << "Sr No. " << setw(5) << "Employee No.  ";

        cout << "  Name  ";

        cout << "     Salary" << endl;

        cout << "\n-----------------------------------------------------------------" << endl;

        i = 1;

        while (fin.read((char \*)&m, sizeof(m)))

        {

            cout << "  " << i << "  ";

            m.display();

            i++;

        }

        fin.close();

    }

}

int add(int cntr)

{

    Employee m;

    ofstream fout("recs.dat", ios::binary | ios::app);

    cout << "Enter the details of the record you want to insert:- " << endl;

    m.accept();

    fout.write((char \*)&m, sizeof(m));

    fout.close();

    cntr++;

    cout << "Record inserted successfully" << endl;

    return cntr;

}

void deleterec(int cntr)

{

    Employee m;

    int flag;

    int eno;

    ifstream fin("recs.dat", ios::binary | ios::in);

    cout << "Enter the Employee number of the record you want to delete:- ";

    cin >> eno;

    flag = 0;

    while (fin.read((char \*)&m, sizeof(m)))

    {

        if (eno == m.geteno())

        {

            flag = 1;

            cntr--;

            break;

        }

    }

    fin.close();

    if (!flag)

        cout << "\nRecord with Employee number " << eno << " not found" << endl;

    else

    {

        ofstream fout("temp.dat", ios::binary | ios::out);

        ifstream fin("recs.dat", ios::binary | ios::out);

        fin.seekg(0, ios::beg);

        fin.read((char \*)&m, sizeof(m));

        while (!fin.eof())

        {

            if (m.geteno() != eno)

                fout.write((char \*)&m, sizeof(m));

            fin.read((char \*)&m, sizeof(m));

        }

        fout.close();

        fin.close();

        cout << "Record with Employee number " << eno << " deleted successfully" << endl;

        remove("recs.dat");

        rename("temp.dat", "recs.dat");

    }

}

void modifyrec(int cntr)

{

    Employee m;

    int eno;

    int flag;

    ifstream fin("recs.dat", ios::binary | ios::in);

    cout << "Enter the Employee number of the record you want to edit:- ";

    cin >> eno;

    flag = 0;

    while (fin.read((char \*)&m, sizeof(m)))

    {

        if (eno == m.geteno())

        {

            flag = 1;

            cntr--;

            break;

        }

    }

    fin.close();

    if (!flag)

        cout << "\nRecord with Employee number " << eno << " not found" << endl;

    else

    {

        ofstream fout("temp.dat", ios::binary | ios::out);

        ifstream fin("recs.dat", ios::binary | ios::out);

        fin.read((char \*)&m, sizeof(m));

        while (!fin.eof())

        {

            if (m.geteno() == eno)

            {

                m.modify();

                fout.write((char \*)&m, sizeof(m));

            }

            else

                fout.write((char \*)&m, sizeof(m));

            fin.read((char \*)&m, sizeof(m));

        }

        fout.close();

        fin.close();

        cout << "Record with Employee number " << eno << " edited successfully" << endl;

        remove("recs.dat");

        rename("temp.dat", "recs.dat");

    }

}

12)heap

/\*Read the marks obtained by students of second year in an online examination of particular subject.

Find out maximum and minimum marks obtained in that subject.

Use heap data structure.

Analyze the algorithm\*/

#include<iostream>

using namespace std;

# define max 20

class stud

{

int mks[max];

public:

    stud()

    {

    for(int i=0;i<max;i++)

      mks[i]=0;

    }

    void insertheap(int tot);

    void displayheap(int tot);

    void showmax(int tot);

    void showmin();

};

void  stud::insertheap(int tot)

{

 for(int i=1;i<=tot;i++)

 {

   cout<<"Enter marks";

   cin>>mks[i];

    int j=i;

    int par=j/2;

    while(mks[j]<=mks[par] && j!=0)

     {

        int tmp = mks[j];

        mks[j]=mks[par];

        mks[par]=tmp;

        j=par;

        par=j/2;

     }

 }

}

void stud::displayheap(int tot)

{

int i=1,space=6;

cout<<endl;

while(i<=tot)

{

    if(i==1 || i==2 || i==4 || i==8 || i==16)

    {

    cout<<endl<<endl;

    for(int j=0;j<space;j++)

         cout<<" ";

    space-=2;

    }

   cout<<" "<<mks[i];

   i++;

}

}

void stud::showmax(int tot)

{

int max1=mks[1];

for(int i=2;i<=tot;i++)

{

 if(max1<mks[i])

  max1= mks[i];

}

cout<<"Max marks:"<<max1;

}

void stud::showmin()

{

cout<<"Min marks:"<<mks[1];

}

int main()

{

int ch,cont,total,tmp;

int n;

stud s1;

do

{

cout<<endl<<"Menu";

cout<<endl<<"1.Read marks of the student ";

cout<<endl<<"2.Display  Min heap";

cout<<endl<<"3.Find Max Marks";

cout<<endl<<"4.Find Min Marks";

cout<<endl<<"Enter Choice";

cin>>ch;

switch(ch)

{

case 1:

     cout<<"How many students";

     cin>>total;

    s1.insertheap(total);

    break;

case 2:

    s1.displayheap(total);

    break;

case 3:    s1.showmax(total);

    break;

case 4:

    s1.showmin();

    break;

}

cout<<endl<<"do u want to continue?(1 for continue)";

cin>>cont;

}while(cont==1);

return 0;

}

/\* output \*/

/\*

Menu

1.Read marks of the student

2.Display  Min heap

3.Find Max Marks

4.Find Min Marks

Enter Choice1

how many students7

enter marks35

enter marks65

enter marks25

enter marks87

enter marks36

enter marks98

enter marks27

do u want to continue?(1 for continue)1

Menu

1.Read marks of the student

2.Display  Min heap

3.Find Max Marks

4.Find Min Marks

Enter Choice2

       25

     36 27

   87 65 98 35

do u want to continue?(1 for continue)1

Menu

1.Read marks of the student

2.Display  Min heap

3.Find Max Marks

4.Find Min Marks

Enter Choice3

Max marks:98

do u want to continue?(1 for continue)1

Menu

1.Read marks of the student

2.Display  Min heap

3.Find Max Marks

4.Find Min Marks

Enter Choice4

Min marks:25

do u want to continue?(1 for continue)0

\*/

linear probing without:

hashTable = [[]] \* 10

def hashFunction(key, capacity):

return key % capacity

def insertData(key, data):

capacity = len(hashTable)

index = hashFunction(key, capacity)

while hashTable[index]:

index = (index + 1) % capacity

hashTable[index] = [key, data]

def removeData(key):

capacity = len(hashTable)

index = hashFunction(key, capacity)

while hashTable[index][0] != key:

index = (index + 1) % capacity

hashTable[index] = []

insertData(123, "apple")

insertData(432, "mango")

insertData(213, "banana")

insertData(654, "guava")

print(hashTable)

removeData(123)

print(hashTable)

c++

#include <iostream>

using namespace std;

#define SIZE 10

int Hash(int key)

{

return key%SIZE;

}

int probe(int H[],int key)

{

int index=Hash(key);

int i=0;

while(H[(index+i)%SIZE]!=0)

{

i++;

}

return (index+i)%SIZE;

}

void insert(int H[],int key)

{

int index=Hash(key);

if(H[index]!=0)

{

index=probe(H,key);

}

H[index]=key;

}

int search(int H[],int key)

{

int index=Hash(key);

int i=0;

while(H[(index+i)%SIZE]!=key)

{

i++;

}

return (index+i)%SIZE;

}

int main()

{

int HT[10]={0};

insert(HT,12);

insert(HT,25);

insert(HT,35);

insert(HT,26);

cout<<"The Hash table and values are : "<<endl;

for(int i=0;i<SIZE;i++)

{

cout<<"H["<<i<<"] index value is "<<HT[i]<<endl;

}

cout<<"\nKey found at "<<search(HT,35);

return 0;

}

13) lab 11

#include<iostream>

#include<cstring>

using namespace std;

struct node

{

char keyword[15],meaning[30];

struct node \*left,\*right;

int height;

};

class avldictionary //class

{

public:

struct node \*insertkeyword(struct node \*r,char ik[15],char im[15]);

struct node \*searchkeyword(struct node \*trav,char sk[15]);

int balanceFactor(struct node \*r);

int maxheight(struct node \*r);

struct node \*RR(struct node \*r);

struct node \*LL(struct node \*r);

struct node \*LR(struct node \*r);

struct node \*RL(struct node \*r);

void ascending(struct node \*r);

void descending(struct node \*r);

struct node \*del(struct node \*r,char k[15]);

};

int avldictionary::balanceFactor(struct node \*r) //return balance factor of node r

{

int lheight,rheight;

if(r->left==NULL)

lheight=0;

else

lheight=1+r->left->height;

if(r->right==NULL)

rheight=0;

else

rheight=1+r->right->height;

return(lheight-rheight); //return LST's and RST's height difference i.e. BF

}

int avldictionary::maxheight(struct node \*r) //return maxheight (either LST's or RST's)

{

int lheight,rheight;

if(r->left==NULL) //if r's LeftSubTree(LST) is NULL, height of LST is 0

lheight=0;

else

lheight=1+r->left->height;

if(r->right==NULL) //if r's RightSubTree(RST) is NULL, height of RST is 0

rheight=0;

else

rheight=1+r->right->height;

if(lheight > rheight)

return lheight;

else

return rheight;

}

struct node \*avldictionary::insertkeyword(struct node \*r,char ik[15],char im[15])

{

if(r==NULL)

{

r=new struct node;

strcpy(r->keyword,ik); //r's keyword and meaning

strcpy(r->meaning,im); //updated with values given by user

r->left=r->right=NULL; //r's both links are set to NULL

}

else if(strcmp(ik, r->keyword) > 0)

{

r->right=insertkeyword(r->right,ik,im);

if(balanceFactor(r)==-2) //BF is -2 then insertion in RightSubTree

{

if(strcmp(ik, r->right-> keyword) > 0)

r=LL(r); // if insertion in RST's RST then LL

else

r=RL(r); //else in RST's LST(Left Sub Tree) then RL

}

}

else if(strcmp(ik, r->keyword) < 0)

{

r->left=insertkeyword(r->left,ik,im);

if(balanceFactor(r)==2) //BF is 2 then isertion in LeftSubTree

{

if(strcmp(ik, r->left-> keyword) < 0)

r=RR(r); //if insertion in LST's LST then RR

else

r=LR(r); //else in LST's RST then LR

}

}

r->height=maxheight(r); //finds maxheight (either from LST or RST) of r

return r;

}

struct node \*avldictionary::RR(struct node \*parent) //RR rotation

{

struct node \*lchild;

lchild=parent->left;

parent->left=lchild->right;

lchild->right=parent;

parent->height=maxheight(parent);

lchild->height=maxheight(lchild);

return lchild;

}

struct node \*avldictionary::LL(struct node \*parent) //LL rotation

{

struct node \*rchild;

rchild=parent->right;

parent->right=rchild->left;

rchild->left=parent;

parent->height=maxheight(parent);

rchild->height=maxheight(rchild);

return rchild;

}

struct node \*avldictionary::LR(struct node \*parent) //LR double rotation

{

parent->left=LL(parent->left); //call single LL rotation

parent=RR(parent); //call single RR rotation

return parent;

}

struct node \*avldictionary::RL(struct node \*parent) //RL double rotation

{

parent->right=RR(parent->right); //call single RR rotation

parent=LL(parent); //call single LL roation

return parent;

}

void avldictionary::ascending(struct node \*r)

{

if(r!=NULL)

{

ascending(r->left);

cout.width(15);

cout<<r->keyword;

cout<<"|";

cout.width(30);

cout<<r->meaning;

cout<<"|";

cout<<"\n-----------------------------------------------\n";

ascending(r->right);

}

}

void avldictionary::descending(struct node \*r)

{

if(r!=NULL)

{

descending(r->right);

cout.width(15);

cout<<r->keyword;

cout<<"|";

cout.width(30);

cout<<r->meaning;

cout<<"|";

cout<<"\n-----------------------------------------------\n";

descending(r->left);

}

}

struct node \*avldictionary::searchkeyword(struct node \*trav,char sk[15])

{ //func. to search keyword in BST

int count=0;

while(trav!=NULL)

{

count++; //counts no. of comparision needed

if(strcmp(sk,trav->keyword)==0)

{

cout<<"\n\n Keyword FOUND Successfullly...!"; //keyword found

cout<<"\n No. of comparions required are: "<<count;

return trav;

}

else if(strcmp(sk,trav->keyword)>0)

{

trav=trav->right; //traverse to right subtree

}

else

{

trav=trav->left; //traverse to left subtree

}

}

return trav; //return trav=NULL when Keyword not found,

//return trav=BST node matched with given keyword

}

struct node \* avldictionary::del(struct node \*r,char k[15])

{

node \*temp;

if(r==NULL)

return NULL;

else

{

if(strcmp(r->keyword,k)<0)

{

r->right=del(r->right,k);

if(balanceFactor(r)==2)

{

if(balanceFactor(r->left)>=0)

r=LL(r);

else

r=LR(r);

}

}

else if(strcmp(r->keyword,k)>0)

{

r->left=del(r->left,k);

if(balanceFactor(r)==-2)

{

if(balanceFactor(r->right)<=0)

r=RR(r);

else

r=RL(r);

}

}

else//Data to be Deleted is found

{

if(r->right!=NULL)

{

temp=r->right;

while(temp->left!=NULL)

temp=temp->left;

strcpy(r->keyword,temp->keyword);

strcpy(r->meaning,temp->meaning);

r->right=del(r->right,temp->keyword);

if(balanceFactor(r)==2)

{

if(balanceFactor(r->left)>=0)

r=LL(r);

else

r=LR(r);

}

}

else

return(r->left);

}

r->height=maxheight(r);

return r;

}

}

int main()

{

char k[15],m[30];

int choice,n;

struct node \*root=NULL,\*found=NULL; //create root pointer and set to NULL

avldictionary obj; //object of dictionary class created

do{

cout<<endl;

cout<<"1. ENTER NEW KEYWORD."<<endl;

cout<<"2. SEARCH KEYWORD."<<endl;

cout<<"3. PRINT DICTIONARY ASCENDING ORDER."<<endl;

cout<<"4. PRINT DICTIONARY DESCENDING ORDER."<<endl;

cout<<"5. DELETE."<<endl;

cout<<"6. UPDATE THE MEANING OF KEYWORD."<<endl;

cout<<"7. EXIT."<<endl;

cout<<" Enter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\n How many keyword you want to insert: ";

cin>>n;

cin.getline(k,0);

for(int i=0;i<n;i++) //loop to accept n keywords and meaning

{

cout<<"\n Enter keyword: ";

cin.getline(k,15);

cout<<" Enter meaning: ";

cin.getline(m,30);

root=obj.insertkeyword(root,k,m); //inserts keywords to BST

}

cout<<"\n Keyword inserted Successfully....!\n";

break;

case 2:

cout<<"\n Enter keyword to be searched: ";

cin>>k;

found=obj.searchkeyword(root,k); //function call to search

//keyword k in BST

if(found==NULL)

{

cout<<"\n Keyword NOT present...\n";

}

else

{ //if 'found' is not NULL it contains

cout<<endl<<endl<<" "; //BST node searched in BST

cout<<found->keyword<<"==>"; //print information of 'found'

cout<<found->meaning;

cout<<endl;

}

break;

case 3:

cout<<"\n Keywords in Ascending Order\n";

cout<<"\n \n";

cout.width(15);

cout<<"Dict. Keyword"; cout<<"|";

cout.width(30);

cout<<"Keyword's Meaning";

cout<<"|\n";

cout<<"===============================================\n";

obj.ascending(root); //prints dictionary in ascending order

break;

case 4:

cout<<"\n Descending Order\n";

cout<<"\n \n";

cout.width(15);

cout<<"Dict. Keyword";

cout<<"|";

cout.width(30);

cout<<"Keyword's Meaning";

cout<<"|\n";

cout<<"===============================================\n";

obj.descending(root);

cout<<"\n Dictonary Printed Successfully....!\n";

break;

case 5:

cout<<"Enter rhe Keyword to be Deleted";

cin.getline(k,0);

cin.getline(k,15);

root=obj.del(root,k);

break;

case 6:

cout<<"Enter the Keyword Whose meaning needs to be Updated";

cin.getline(k,0);

cin.getline(k,15);

found=obj.searchkeyword(root,k);

if(found==NULL)

cout<<"No such Keyword present to update meaning";

else

{

cout<<"Enter the new Meaning";

cin.getline(m,30);

strcpy(found->meaning,m);

cout<<"Keyword's Meaning is updated successfully";

}

}//switch ends...

}while(choice!=7);

return 0;

}