Assignment 1

Accept prefix expressions, and construct a binary tree and perform recursive and non-recursive traversals.

```
#include <bits/stdc++.h>
using namespace std;

class node {
public:
      int data;
      node* left;
      node* right;
};


node* newNode(int data)
{
      node* temp = new node();

      temp->data = data;
      temp->left = temp->right = NULL;

      return temp;
}

node* constructTreeUtil(int pre[], int* preIndex, int low, int high,
int size)
{
      if (*preIndex >= size || low > high)
            return NULL;
      node* root = newNode(pre[*preIndex]);
      *preIndex = *preIndex + 1;
      if (low == high)
            return root;
      int i;
      for (i = low; i <= high; ++i)

     {
            if (pre[i] > root->data)
            break;
            }
      root->left = constructTreeUtil(pre, preIndex, *preIndex,i - 1,
size);
      root->right = constructTreeUtil(pre, preIndex, i, high, size);

      return root;
}

node* constructTree(int pre[], int size)
{
```

```
	int preIndex = 0;
	return constructTreeUtil(pre, &preIndex, 0, size - 1,size);
}

void printInorder(node* node)
{
	if (node == NULL)
		return;
	printInorder(node->left);
	cout << node->data << " ";
	printInorder(node->right);
}


int main()
{
	int pre[] = { 10, 5, 1, 7, 40, 50 };
	int size = sizeof(pre) / sizeof(pre[0]);

	node* root = constructTree(pre, size);

	cout << "Inorder traversal of the constructed tree: \n";
	printInorder(root);

	return 0;
}
```

Assignment 2

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide a facility to display whole data sorted in ascending/ Descending order. Also, find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.

```
#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;
}
```

Assignment 3

Create a Binary Search tree and find its mirror image. Print original & new tree level wise. Find height & print leaf nodes.

```
#include<bits/stdc++.h>
using namespace std;
struct Node
{
      int data;
      struct Node* left;
      struct Node* right;
};
struct Node* newNode(int data)
{
      struct Node* node = (struct Node*) malloc(sizeof(struct Node));
      node->data = data;
      node->left = NULL;
      node->right = NULL;

      return(node);
}
void mirror(struct Node* node)
{
      if (node == NULL)
            return;
      else
      {
            struct Node* temp;
            mirror(node->left);
            mirror(node->right);

            temp   = node->left;
            node->left = node->right;
            node->right = temp;
      }
}

void inOrder(struct Node* node)
{
      if (node == NULL)
            return;

      inOrder(node->left);
      cout << node->data << " ";
      inOrder(node->right);
}
int main()
{
      struct Node *root = newNode(1);
      root->left = newNode(2);
```

```
      root->right = newNode(3);
      root->left->left = newNode(4);
      root->left->right = newNode(5);

      cout << "Inorder traversal of the constructed" << " tree is" <<
endl;
      inOrder(root);


      mirror(root);


      cout << "\nInorder traversal of the mirror tree" << " is \n";
      inOrder(root);

      return 0;
}
```

Assignment 4

Create an in-order threaded binary search tree and perform the traversals.

```
#include <iostream>
#define MAX_VALUE 65536
using namespace std;
class N { //node declaration
  public:
    int k;
    N *l, *r;
    bool leftTh, rightTh;
};
class ThreadedBinaryTree {
  private:
  N *root;
  public:
  ThreadedBinaryTree() { //constructor to initialize the variables
    root= new N();
    root->r= root->l= root;
    root->leftTh = true;
    root->k = MAX_VALUE;
  }
  void insert(int key) {
    N *p = root;
    for (;;) {
    if (p->k< key) { //move to right thread
```

```
      if (p->rightTh)
         break;
      p = p->r;
   }
   else if (p->k > key) { // move to left thread
      if (p->leftTh)
         break;
      p = p->l;
   }
   else {
      return;
   }
}
N *temp = new N();
temp->k = key;
temp->rightTh= temp->leftTh= true;
if (p->k < key) {
   temp->r = p->r;
   temp->l= p;
   p->r = temp;
   p->rightTh= false;
}
else {
   temp->r = p;
   temp->l = p->l;
```

```
      p->l = temp;
      p->leftTh = false;
    }
}
void inorder() { //print the tree
   N *temp = root, *p;
   for (;;) {
      p = temp;
      temp = temp->r;
      if (!p->rightTh) {
         while (!temp->leftTh) {
            temp = temp->l;
         }
      }
      if (temp == root)
         break;
         cout<<temp->k<<" ";
      }
      cout<<endl;
   }
};
int main() {
   ThreadedBinaryTree tbt;
   cout<<"Threaded Binary Tree\n";
   tbt.insert(56);
```

```
    tbt.insert(23);
    tbt.insert(89);
    tbt.insert(85);
    tbt.insert(20);
    tbt.insert(30);
    tbt.insert(12);
    tbt.inorder();
    cout<<"\n";
}
```

Output:

```
Threaded Binary Tree
12 20 23 30 56 85 89
```

Assignment 5

Represent a given graph using an adjacency list and perform DFS or BFS.

DFS Traversal

```
#include <iostream>
#include <list>
using namespace std;
//graph class for DFS travesal
class DFSGraph
{
int V;    // No. of vertices
list<int> *adjList;    // adjacency list
void DFS_util(int v, bool visited[]);  // A function used by DFS
public:
   // class Constructor
DFSGraph(int V)
   {
 this->V = V;
 adjList = new list<int>[V];
   }
   // function to add an edge to graph
void addEdge(int v, int w){
adjList[v].push_back(w); // Add w to v's list.
   }
```

```
void DFS();    // DFS traversal function
};
void DFSGraph::DFS_util(int v, bool visited[])
{
   // current node v is visited
visited[v] = true;
cout << v << " ";

   // recursively process all the adjacent vertices of the node
list<int>::iterator i;
for(i = adjList[v].begin(); i != adjList[v].end(); ++i)
if(!visited[*i])
DFS_util(*i, visited);
}

// DFS traversal
void DFSGraph::DFS()
{
   // initially none of the vertices are visited
bool *visited = new bool[V];
for (int i = 0; i < V; i++)
visited[i] = false;

   // explore the vertices one by one by recursively calling  DFS_util
for (int i = 0; i < V; i++)
```

```
if (visited[i] == false)
DFS_util(i, visited);
}

int main()
{
   // Create a graph
DFSGraph gdfs(5);
gdfs.addEdge(0, 1);
gdfs.addEdge(0, 2);
gdfs.addEdge(0, 3);
gdfs.addEdge(1, 2);
gdfs.addEdge(2, 4);
gdfs.addEdge(3, 3);
gdfs.addEdge(4, 4);

cout << "Depth-first traversal for the given graph:"<<endl;
gdfs.DFS();

return 0;
}
```

Output :

```
Depth-first traversal for the given graph:
0 1 2 4 3
```

BFS Traversal

```
// 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 adjecent : adj[s]) {
            if (!visited[adjecent]) {
                visited[adjecent] = true;
                queue.push_back(adjecent);
            }
        }
    }
}

// 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;
}
```

Output:

Following is Breadth First Traversal (starting from vertex 2)

2 0 3 1

Assignment 6

Represent a given graph using an adjacency list or array and find the shortest path using Dijkstra's algorithm.

```
// 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;
}
```

Output:

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

Assignment 7

Represent a given graph using an adjacency list or array and generate a minimum spanning tree using Kruskal's or Prim’s algorithm.

Prim’s algorithm.

```
#include <iostream>
#define V 7      // number of vertices in graph.
using namespace std;
 int main ()
 {
  int G[V][V] =
  {
     {0,28,0,0,0,10,0},
     {28,0,16,0,0,0,14},
     {0,16,0,12,0,0,0},
     {0,0,12,22,0,18},
     {0,0,0,22,0,25,24},
     {10,0,0,0,25,0,0},
     {0,14,0,18,24,0,0}
  };
   int edge;
   int visit[V];
   for(int i=0;i<V;i++)
    {
    visit[i]=false;
    }
   edge = 0;
   visit[0] = true;
   int x;            //  row number
   int y;            //  col number

   cout<<"\n Result of Minimum Spanning Tree using Prim's
Algorithm\n\n";
   cout << " Edge" << "        : " << "Weight\n ====================";
   cout << endl;
   while (edge < V - 1) {

       int min = INT_MAX;
       x = 0;
       y = 0;

       for (int i = 0; i < V; i++)
        {
         if (visit[i])
         {
             for (int j = 0; j < V; j++)
             {
               if (!visit[j] && G[i][j])
               { // not in selected and there is an edge
```

```
                    if (min > G[i][j])
                    {
                        min = G[i][j];
                        x = i;
                        y = j;
                    }
                }
            }
          }
        }
        cout << " "<<x <<  " ---> " << " "<<y << "   :  " << "
"<<G[x][y];
        cout << endl;

        visit[y] = true;
        edge++;
      }
      cout<<"======================\n";
   return 0;
 }
```

Kruskal's algorithm

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAX = 100;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];
void init()
{
    for(int i = 0;i < MAX;++i)
        id[i] = i;
}
int root(int x)
{
    while(id[x] != x)
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x;
}
void union1(int x, int y)
{
    int p = root(x);
    int q = root(y);
    id[p] = id[q];
}
long long kruskal(pair<long long, pair<int, int> > p[])
{
```

```
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0;i < edges;++i)
    {
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        if(root(x) != root(y))
        {
            minimumCost = minimumCost+cost;
            union1(x, y);
        }
    }
    return minimumCost;
}
int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    init();
    cout <<"Enter Nodes and edges: ";
    cin >> nodes >> edges;
    for(int i = 0;i < edges;++i)
    {
        cout<<"Enter the value of X, Y and edges: ";
        cin >> x >> y >> weight;
        p[i] = make_pair(weight, make_pair(x, y));
    }
    sort(p, p + edges);
    minimumCost = kruskal(p);
    cout <<"Minimum cost is:- "<< minimumCost << endl;
    return 0;
}
```

Assignment 8

Create a hash table and handle the collisions using linear probing with or without replacement.

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
const int T_S = 5;
class HashTable {
  public:
    int k;
    int v;
    HashTable(int k, int v) {
      this->k = k;
      this->v = v;
    }
};
class DelNode:public HashTable {
  private:
    static DelNode *en;
    DelNode():HashTable(-1, -1) {}
  public:
    static DelNode *getNode() {
      if (en == NULL)
        en = new DelNode();
```

```
        return en;
    }
};
DelNode *DelNode::en = NULL;
class HashMapTable {
  private:
    HashTable **ht;
  public:
    HashMapTable() {
      ht = new HashTable* [T_S];
      for (int i = 0; i < T_S; i++) {
        ht[i] = NULL;
      }
    }
    int HashFunc(int k) {
      return k % T_S;
    }
    void Insert(int k, int v) {
      int hash_val = HashFunc(k);
      int init = -1;
      int delindex = -1;
      while (hash_val != init && (ht[hash_val]  == DelNode::getNode() || ht[hash_val] != NULL
&& ht[hash_val]->k != k)) {
        if (init == -1)
          init = hash_val;
```

```
        if (ht[hash_val] == DelNode::getNode())
            delindex = hash_val;
            hash_val = HashFunc(hash_val + 1);
        }
        if (ht[hash_val] == NULL || hash_val == init) {
            if(delindex != -1)
                ht[delindex] = new HashTable(k, v);
            else
                ht[hash_val] = new HashTable(k, v);
        }
        if(init != hash_val) {
            if (ht[hash_val] != DelNode::getNode()) {
                if (ht[hash_val] != NULL) {
                    if (ht[hash_val]->k== k)
                        ht[hash_val]->v = v;
                }
            } else
            ht[hash_val] = new HashTable(k, v);
        }
    }
    int SearchKey(int k) {
        int hash_val = HashFunc(k);
        int init = -1;
        while (hash_val != init && (ht[hash_val] == DelNode::getNode() || ht[hash_val] != NULL
&& ht[hash_val]->k!= k)) {
```

```
        if (init == -1)
            init = hash_val;
            hash_val = HashFunc(hash_val + 1);
        }
        if (ht[hash_val] == NULL || hash_val == init)
            return -1;
        else
            return ht[hash_val]->v;
    }
    void Remove(int k) {
        int hash_val = HashFunc(k);
        int init = -1;
        while (hash_val != init && (ht[hash_val] == DelNode::getNode() || ht[hash_val] != NULL
&& ht[hash_val]->k!= k)) {
            if (init == -1)
                init = hash_val;
                hash_val = HashFunc(hash_val + 1);
        }
        if (hash_val != init && ht[hash_val] != NULL) {
            delete ht[hash_val];
            ht[hash_val] = DelNode::getNode();
        }
    }
    ~HashMapTable() {
        delete[] ht;
```

```
    }
};
int main() {
  HashMapTable hash;
  int k, v;
  int c;
  while(1) {
    cout<<"1.Insert element into the table"<<endl;
    cout<<"2.Search element from the key"<<endl;
    cout<<"3.Delete element at a key"<<endl;
    cout<<"4.Exit"<<endl;
    cout<<"Enter your choice: ";
    cin>>c;
    switch(c) {
      case 1:
        cout<<"Enter element to be inserted: ";
        cin>>v;
        cout<<"Enter key at which element to be inserted: ";
        cin>>k;
        hash.Insert(k, v);
      break;
      case 2:
        cout<<"Enter key of the element to be searched: ";
        cin>>k;
        if(hash.SearchKey(k) == -1) {
```

```
            cout<<"No element found at key "<<k<<endl;
            continue;
          } else {
            cout<<"Element at key "<<k<<" : ";
            cout<<hash.SearchKey(k)<<endl;
          }
        break;
        case 3:
          cout<<"Enter key of the element to be deleted: ";
          cin>>k;
          hash.Remove(k);
        break;
        case 4:
          exit(1);
        default:
          cout<<"\nEnter correct option\n";
      }
   }
   return 0;
}
```

Output:

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 1

Enter element to be inserted: 10

Enter key at which element to be inserted: 2

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 1

Enter element to be inserted: 7

Enter key at which element to be inserted: 6

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 1

Enter element to be inserted: 4

Enter key at which element to be inserted: 5

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 1

Enter element to be inserted: 12

Enter key at which element to be inserted: 3

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 15

Enter correct option

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 1

Enter element to be inserted: 15

Enter key at which element to be inserted: 8

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 2

Enter key of the element to be searched: 6

Element at key 6 : 7

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 3

Enter key of the element to be deleted: 2

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 2

Enter key of the element to be searched: 2

No element found at key 2

1.Insert element into the table

2.Search element from the key

3.Delete element at a key

4.Exit

Enter your choice: 4

Assignment 9

Implementation of simple index file.

```
# Python program for Indexed
# Sequential Search


def indexedSequentialSearch(arr, n, k):

    elements = [0] * 20
    indices = [0] * 20
    j, ind, start, end = 0, 0, 0, 0
    set_flag = 0

    for i in range(0, n, 3):

        # Storing element
        elements[ind] = arr[i]

        # Storing the index
        indices[ind] = i
        ind += 1

    if k < elements[0]:
        print("Not found")
        exit(0)

    else:

        for i in range(1, ind + 1):
            if k <= elements[i]:
                start = indices[i - 1]
                end = indices[i]
                set_flag = 1
                break
    if set_flag == 0:
        start = indices[i-1]
        end = n
    for i in range(start, end + 1):
        if k == arr[i]:
            j = 1
            break

    if j == 1:
        print("Found at index", i)
    else:
        print("Not found")


# Driver code
if __name__ == "__main__":
```

```
arr = [6, 7, 8, 9, 10]
n = len(arr)

# Element to search
k = 8

# Function call
indexedSequentialSearch(arr, n, k)
```

Output:

Found at index 2

Assignment 10

Company maintains employee information such as employee ID, name, designation and salary. Allow users to add, delete information about employees. Display information of a particular employee. If an employee does not exist, an appropriate message is displayed. If it is, then the system displays the employee details. Use a sequential file to maintain the data.

```
#include<iostream>
#include<fstream>
#include<cstring>
using namespace std;
class tel
 {

 public:
     int rollNo,roll1;
     char name[10];
     char div;
     char address[20];
     void accept()
     {
          cout<<"\n\tEnter Roll Number : ";
          cin>>rollNo;
          cout<<"\n\tEnter the Name : ";
          cin>>name;
          cout<<"\n\tEnter the Division:";
          cin>>div;
          cout<<"\n\tEnter the Address:";
          cin>>address;
     }
        void accept2()
        {
               cout<<"\n\tEnter the Roll No. to modify : ";
               cin>>rollNo;
        }
        void accept3()
        {
              cout<<"\n\tEnter the name to modify : ";
              cin>>name;
        }
        int getRollNo()
        {
           return rollNo;
        }
       void show()
       {


     cout<<"\n\t"<<rollNo<<"\t\t"<<name<<"\t\t"<<div<<"\t\t"<<address;
```

```
      }
};
int main()
{
     int
i,n,ch,ch1,rec,start,count,add,n1,add2,start2,n2,y,a,b,on,oname,add3,s
tart3,n3,y1,add4,start4,n4;
     char name[20],name2[20];
     tel t1;
     count=0;
     fstream g,f;
     do
     {
          cout<<"\n>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<";
          cout<<"\n1.Insert and overwrite\n2.Show\n3.Search &
Edit(number)\n4.Search & Edit(name)\n5.Search &
Edit(onlynumber)\n6.Search & edit(only name)\n 7.Delete a Student
Record\n 8.Exit\n\tEnter the Choice\t:";
          cin>>ch;
          switch(ch)
          {
          case 1:
               f.open("StuRecord.txt",ios::out);
               x:t1.accept();
               f.write((char*) &t1,(sizeof(t1)));
               cout<<"\nDo you want to enter more
records?\n1.Yes\n2.No";
               cin>>ch1;
                    if(ch1==1)
                         goto x;
                    else
                    {
                         f.close();
                         break;
                    }

          case 2:
               f.open("StuRecord.txt",ios::in);
               f.read((char*) &t1,(sizeof(t1)));
               //cout<<"\n\tRoll No.\t\tName \t\t Division \t\t
Address";
               while(f)
               {
                    t1.show();
                    f.read((char*) &t1,(sizeof(t1)));
               }
               f.close();
               break;
          case 3:
               cout<<"\nEnter the roll number you want to find";
               cin>>rec;
               f.open("StuRecord.txt",ios::in|ios::out);
```

```
      f.read((char*)&t1,(sizeof(t1)));
      while(f)
      {
            if(rec==t1.rollNo)
            {
                  cout<<"\nRecord found";
                  add=f.tellg();
                  f.seekg(0,ios::beg);
                    start=f.tellg();
                  n1=(add-start)/(sizeof(t1));
                  f.seekp((n1-1)*sizeof(t1),ios::beg);
                  t1.accept();
                  f.write((char*) &t1,(sizeof(t1)));
                  f.close();
                  count++;
                  break;
            }
            f.read((char*)&t1,(sizeof(t1)));
      }
      if(count==0)
      cout<<"\nRecord not found";
      f.close();
      break;

case 4:
       cout<<"\nEnter the name you want to find and edit";
       cin>>name;
      f.open("StuRecord.txt",ios::in|ios::out);
      f.read((char*)&t1,(sizeof(t1)));
      while(f)
      {
       y=(strcmp(name,t1.name));
       if(y==0)
       {
             cout<<"\nName found";
             add2=f.tellg();
             f.seekg(0,ios::beg);
             start2=f.tellg();
             n2=(add2-start2)/(sizeof(t1));
             f.seekp((n2-1)*sizeof(t1),ios::beg);
             t1.accept();
             f.write((char*) &t1,(sizeof(t1)));
             f.close();
             break;
       }
      f.read((char*)&t1,(sizeof(t1)));
      }
      break;
case 5:
      cout<<"\n\tEnter the roll number you want to modify";
      cin>>on;
      f.open("StuRecord.txt",ios::in|ios::out);
```

```
                f.read((char*) &t1,(sizeof(t1)));
                while(f)
                {
                  if(on==t1.rollNo)
                  {
                    cout<<"\n\tNumber found";
                    add3=f.tellg();
                    f.seekg(0,ios::beg);
                    start3=f.tellg();
                    n3=(add3-start3)/(sizeof(t1));
                    f.seekp((n3-1)*(sizeof(t1)),ios::beg);
                    t1.accept2();
                    f.write((char*)&t1,(sizeof(t1)));
                    f.close();
                    break;
                  }
                  f.read((char*)&t1,(sizeof(t1)));
                }
               break;
          case 6:
                cout<<"\nEnter the name you want to find and edit";
                 cin>>name2;
                f.open("StuRecord.txt",ios::in|ios::out);
                f.read((char*)&t1,(sizeof(t1)));
                while(f)
                {
                 y1=(strcmp(name2,t1.name));
                 if(y1==0)
                 {
                       cout<<"\nName found";
                       add4=f.tellg();
                       f.seekg(0,ios::beg);
                       start4=f.tellg();
                       n4=(add4-start4)/(sizeof(t1));
                       f.seekp((n4-1)*sizeof(t1),ios::beg);
                       t1.accept3();
                       f.write((char*) &t1,(sizeof(t1)));
                       f.close();
                       break;
                 }
                f.read((char*)&t1,(sizeof(t1)));
                }
                break;
        case 7:
                int roll;
                cout<<"Please Enter the Roll No. of Student Whose Info
You Want to Delete: ";
                cin>>roll;
                f.open("StuRecord.txt",ios::in);
                g.open("temp.txt",ios::out);
                f.read((char *)&t1,sizeof(t1));
                while(!f.eof())
```

```
                {
                      if (t1.getRollNo() != roll)
                            g.write((char *)&t1,sizeof(t1));
                      f.read((char *)&t1,sizeof(t1));
                }
                cout << "The record with the roll no. " << roll << "
has been deleted " << endl;
                f.close();
                g.close();
                remove("StuRecord.txt");
                rename("temp.txt","StuRecord.txt");
                break;
         case 8:
            cout<<"\n\tThank you";
            break;


         }
  }while(ch!=8);
}
/*
>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
      Enter the Choice :1

      Enter Roll Number : 49

      Enter the Name : abhishek

      Enter the Division:b

      Enter the Address:delhi

Do you want to enter more records?
1.Yes
2.No1

      Enter Roll Number : 39

      Enter the Name : ajinkya

      Enter the Division:b

      Enter the Address:pune
```

```
Do you want to enter more records?
1.Yes
2.No2

>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
     Enter the Choice :2

     49         abhishek         b          delhi
     39         ajinkya          b          pune
>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
     Enter the Choice :3

Enter the roll number you want to find39

Record found
     Enter Roll Number : 67

     Enter the Name : venky

     Enter the Division:b

     Enter the Address:pune

>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
     Enter the Choice :2

     49         abhishek         b          delhi
     67         venky      b          pune
```

```
>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
     Enter the Choice :4

Enter the name you want to find and editabhishek

Name found
     Enter Roll Number : 47

     Enter the Name : safaya

     Enter the Division:b

     Enter the Address:pune

>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
     Enter the Choice :7
Please Enter the Roll No. of Student Whose Info You Want to Delete: 67
The record with the roll no. 67 has been deleted

>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
     Enter the Choice :2

     47         safaya           b          pune
>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
```

```
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
     Enter the Choice :8
     */
```

Assignment 11

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide a facility to display whole data sorted in ascending/ Descending order. Also, find how many maximum comparisons may require for finding any keyword. Use Height balanced tree and find the complexity for finding a keyword.

```
#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;
}
/*
1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
      Enter your choice: 1

 How many keyword you want to insert: 5

 Enter keyword: assumption
 Enter meaning: gfsdj

 Enter keyword: tyweww
 Enter meaning: ffhsd

 Enter keyword: mfhagsd
 Enter meaning: hjsgdhf
```

```
 Enter keyword: hdfsaff
 Enter meaning: ffssdf

 Enter keyword: zjfs
 Enter meaning: fsjdjf

 Keyword inserted Successfully....!

1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 2

 Enter keyword to be searched: tyweww


 Keyword FOUND Successfullly...!
 No. of comparions required are: 2

 tyweww==>ffhsd

1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 3

 Keywords in Ascending Order


  Dict. Keyword|             Keyword's Meaning|
===============================================
     assumption|                         gfsdj|
-----------------------------------------------
        hdfsaff|                       ffssdf|
-----------------------------------------------
        mfhagsd|                      hjsgdhf|
-----------------------------------------------
         tyweww|                         ffhsd|
-----------------------------------------------
           zjfs|                       fsjdjf|
-----------------------------------------------

1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
```

```
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 4

 Descending Order


  Dict. Keyword|             Keyword's Meaning|
==============================================
           zjfs|                          fsjdjf|
----------------------------------------------
         tyweww|                           ffhsd|
----------------------------------------------
        mfhagsd|                        hjsgdhf|
----------------------------------------------
        hdfsaff|                          ffssdf|
----------------------------------------------
     assumption|                           gfsdj|
----------------------------------------------

     Dictonary Printed Successfully....!

1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 5
Enter rhe Keyword to be Deletedmfhagsd

1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 3

 Keywords in Ascending Order


  Dict. Keyword|             Keyword's Meaning|
==============================================
     assumption|                           gfsdj|
----------------------------------------------
        hdfsaff|                          ffssdf|
```

```
-----------------------------------------------
         tyweww|                          ffhsd|
-----------------------------------------------
           zjfs|                         fsjdjf|
-----------------------------------------------

1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 6
Enter the Keyword Whose meaning needs to be Updatedtyweww


 Keyword FOUND Successfullly...!
 No. of comparions required are: 1Enter the new Meaningaaaagfsd
Keyword's Meaning is updated successfully
1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 3

 Keywords in Ascending Order


  Dict. Keyword|             Keyword's Meaning|
===============================================
     assumption|                          gfsdj|
-----------------------------------------------
        hdfsaff|                         ffssdf|
-----------------------------------------------
         tyweww|                       aaaagfsd|
-----------------------------------------------
           zjfs|                         fsjdjf|
-----------------------------------------------

1. ENTER NEW KEYWORD.
2. SEARCH KEYWORD.
3. PRINT   DICTIONARY ASCENDING ORDER.
4. PRINT   DICTIONARY DESCENDING ORDER.
5. DELETE.
6. UPDATE THE MEANING OF KEYWORD.
7. EXIT.
     Enter your choice: 7
*/
```

Assignment 12

Implement Heap sort.

```
// C++ program for implementation of Heap Sort
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root Since we are using 0
based indexing
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
```

```
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver program
int main()
{
    int arr[] = { 60 ,20 ,40 ,70, 30, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
  //heapify algorithm
  // the loop must go reverse you will get after analyzing manually
  // (i=n/2 -1) because other nodes/ ele's are leaf nodes
  // (i=n/2 -1) for 0 based indexing
  // (i=n/2)  for 1 based indexing
     for(int i=n/2 -1;i>=0;i--){
       heapify(arr,n,i);
   }

  cout << "After heapifying array is \n";
    printArray(arr, n);


    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);

  return 0;
}
```

Output:

After heapifying array is

70 60 40 20 30 10

Sorted array is

10 20 30 40 60 70