**PRACTICAL NO.1**

import array as arr  
  
def acceptData(n):  
 l1=[]  
 while(len(l1)<n):  
 t=(input("Enter telephone nos of client:"))  
 #print(n)  
 if(len(t)==10)and(t.isdigit()):  
 l1.append(t)  
 else:  
 print("\nvaild telephone no:")  
 return(l1)  
  
#example of linear probing  
#n=int(input("Enter nos of clients:"))  
#l1=acceptData(n)  
l1=[1234567890, 1234567891, 1234567892, 1234567885, 1234567887,1234567871,1234567876, 1234567877, 1234567899, 1234567898]  
def linearProbing(l1):  
 n=len(l1)  
 t1=[]  
 for i in range(0,n):  
 t1.append(0)  
 coll=0  
 ht=arr.array('i',t1)  
 for i in l1:  
 t=i%n  
 coll=1  
 if(ht[t]==0):  
 ht[t]=i  
 print('inserted key:',i,' at:',t, "with complexity:","O(",str(coll),")")  
 coll=0  
 else:  
 #print(ht)  
 print("Collision at index: ",t, " for ", i)  
 print("Use probing:")  
 #coll=0  
 cnt=t  
 while(1):  
 cnt=(cnt+1)%n  
 if(cnt==t):  
 print("Collision cannot be resolved or overflow")  
 break  
 else:  
 if(ht[cnt]==0):  
 ht[cnt]=i  
 print('inserted key:',i,' at:',cnt, "with complexity:","O(",str(coll+1),")")  
 coll=0  
 break  
 else:  
 coll+=1  
 return(ht)  
ht=linearProbing(l1)  
# ht is a lookup table  
print(ht)  
  
  
def search(key, ht):  
 print("\_" \* 60)  
 print("Linear Probing:")  
 coll = 1  
 n = len(ht)  
 # print(n)  
 t = key % n  
 if (ht[t] == key):  
 print("\_" \* 60)  
 print("Telephone no found at", t, "Complexity: O(1)", " key:", key)  
  
 else:  
 print("Search Using Linear Probing")  
 coll = 1  
 cnt = t  
 while (1):  
 cnt = (cnt + 1) % n  
 if (cnt == t):  
 print("\_" \* 60)  
 print("Telephone nos not found:", key)  
 print("Telephone not found Complexity:O(", str(coll + 1), ") key:", key)  
 break  
 else:  
 if (ht[cnt] == key):  
 print("\_" \* 60)  
 print("Telephone no found at", cnt, "Complexity:O(", str(coll + 1), ") key:", key)  
 break  
 else:  
 coll += 1  
  
  
search(1234567891, ht)  
search(1234567871, ht)  
search(1234567870, ht)

OUTPUT-

C:\Users\NETRA\PycharmProjects\pythonProject6\venv\Scripts\python.exe C:\Users\NETRA\PycharmProjects\pythonProject6\main.py

inserted key: 1234567890 at: 0 with complexity: O( 1 )

inserted key: 1234567891 at: 1 with complexity: O( 1 )

inserted key: 1234567892 at: 2 with complexity: O( 1 )

inserted key: 1234567885 at: 5 with complexity: O( 1 )

inserted key: 1234567887 at: 7 with complexity: O( 1 )

Collision at index: 1 for 1234567871

Use probing:

inserted key: 1234567871 at: 3 with complexity: O( 3 )

inserted key: 1234567876 at: 6 with complexity: O( 1 )

Collision at index: 7 for 1234567877

Use probing:

inserted key: 1234567877 at: 8 with complexity: O( 2 )

inserted key: 1234567899 at: 9 with complexity: O( 1 )

Collision at index: 8 for 1234567898

Use probing:

inserted key: 1234567898 at: 4 with complexity: O( 7 )

array('i', [1234567890, 1234567891, 1234567892, 1234567871, 1234567898, 1234567885, 1234567876, 1234567887, 1234567877, 1234567899])

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

Linear Probing:

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

Telephone no found at 1 Complexity: O(1) key: 1234567891

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

Linear Probing:

Search Using Linear Probing

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

Telephone no found at 3 Complexity:O( 3 ) key: 1234567871

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

Linear Probing:

Search Using Linear Probing

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

Telephone nos not found: 1234567870

Telephone not found Complexity:O( 11 ) key: 1234567870

Process finished with exit code 0

**PRACTICAL NO.2**

# set operation  
def setCreate():  
 l1 = eval(input("Enter values in set"))  
 return (set(l1))  
  
  
def setUpdate(set1, data): # can be used for union of set  
 s1 = set(data) # single data so use {} else set(list)  
 s1.update(set1)  
 return (s1)  
  
  
def setRemove(set1, data):  
 if (setContains(set1, data)):  
 set1.remove(data)  
 return (set1)  
  
  
def setContains(set1, data):  
 s1 = {data}  
 if (s1.issubset(set1)):  
 return (True)  
 else:  
 return (False)  
  
  
def setSize(s1):  
 return (len(s1))  
  
  
def union(s1, s2):  
 return (s1.union(s2))  
  
  
def diff(s1, s2):  
 s1 -= s2  
 return (s1)  
  
  
def intersect(s1, s2):  
 s1 &= s2  
 return (s1)  
  
  
def symmetric\_diff(s1, s2):  
 s1 ^= s2  
 return (s1)  
  
  
# driver  
s1 = setCreate()  
s1 = setUpdate(s1, [1, 2, 6]) # Same as Union  
print(s1)  
s1 = setRemove(s1, 6)  
print(s1)  
  
print(setContains(s1, 5))  
print(setContains(s1, 6))  
  
s1 = setUpdate({1, 2, 3, 4, 5}, {4, 5, 6}) # alternative to union  
print(s1)  
print(union({1, 2, 3, 4, 5}, {4, 5, 6}))  
  
print(diff({1, 2, 3, 4, 5}, {4, 5, 6}))  
  
print(intersect({1, 2, 3, 4, 5}, {4, 5, 6}))  
  
print(symmetric\_diff({1, 2, 3, 4, 5}, {4, 5, 6}))  
  
print(setSize({1, 2, 3, 4, 5}))1

**OUTPUT-**

C:\Users\NETRA\PycharmProjects\pythonProject6\venv\Scripts\python.exe C:\Users\NETRA\PycharmProjects\pythonProject6\main.py

Enter values in set1, 2, 6

{1, 2, 6}

{1, 2}

False

False

{1, 2, 3, 4, 5, 6}

{1, 2, 3, 4, 5, 6}

{1, 2, 3}

{4, 5}

{1, 2, 3, 6}

5

**PRACTICAL N0.3**

#include <iostream>

#include <string.h>

using namespace std;

struct node // Node Declaration

{

string label;

//char label[10];

int ch\_count;

struct node \*child[10];

} \* root;

class GT // Class Declaration

{

public:

void create\_tree();

void display(node \*r1);

GT()

{

root = NULL;

}

};

void GT::create\_tree()

{

int tbooks, tchapters, i, j, k;

root = new node;

cout << "Enter name of book : ";

cin.get();

getline(cin, root->label);

cout << "Enter number of chapters in book : ";

cin >> tchapters;

root->ch\_count = tchapters;

for (i = 0; i < tchapters; i++)

{

root->child[i] = new node;

cout << "Enter the name of Chapter " << i + 1 << " : ";

cin.get();

getline(cin, root->child[i]->label);

cout << "Enter number of sections in Chapter : " << root->child[i]->label << " : ";

cin >> root->child[i]->ch\_count;

for (j = 0; j < root->child[i]->ch\_count; j++)

{

root->child[i]->child[j] = new node;

cout << "Enter Name of Section " << j + 1 << " : ";

cin.get();

getline(cin, root->child[i]->child[j]->label);

}

}

}

void GT::display(node \*r1)

{

int i, j, k, tchapters;

if (r1 != NULL)

{

cout << "\n-----Book Hierarchy---";

cout << "\n Book title : " << r1->label;

tchapters = r1->ch\_count;

for (i = 0; i < tchapters; i++)

{

cout << "\nChapter " << i + 1;

cout << " : " << r1->child[i]->label;

cout << "\nSections : ";

for (j = 0; j < r1->child[i]->ch\_count; j++)

{

cout << "\n"<< r1->child[i]->child[j]->label;

}

}

}

cout << endl;

}

int main()

{

int choice;

GT gt;

while (1)

{

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

cout << "Book Tree Creation" << endl;

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

cout << "1.Create" << endl;

cout << "2.Display" << endl;

cout << "3.Quit" << endl;

cout << "Enter your choice : ";

cin >> choice;

switch (choice)

{

case 1:

gt.create\_tree();

case 2:

gt.display(root);

break;

case 3:

cout << "Thanks for using this program!!!";

exit(1);

default:

cout << "Wrong choice!!!" << endl;

}

}

return 0;

}

**OUTPUT-**

-----------------

Book Tree Creation

-----------------

1.Create

2.Display

3.Quit

Enter your choice : 1

Enter name of book : MOM

Enter number of chapters in book : 2

Enter the name of Chapter 1 : 2

Enter number of sections in Chapter : 2 : 2

Enter Name of Section 1 : ABC

Enter Name of Section 2 : PQR

Enter the name of Chapter 2 : B

Enter number of sections in Chapter : : 1

Enter Name of Section 1 : RPS

-----Book Hierarchy---

Book title : MOM

Chapter 1 : 2

Sections :

ABC

QR

Chapter 2 :

Sections :

RPS

-----------------

Book Tree Creation

-----------------

1.Create

2.Display

3.Quit

Enter your choice : 2

-----Book Hierarchy---

Book title : MOM

Chapter 1 : 2

Sections :

ABC

QR

Chapter 2 :

Sections :

RPS

-----------------

Book Tree Creation

-----------------

1.Create

2.Display

3.Quit

Enter your choice : 3

Thanks for using this program!!!

**PRACTICAL NO.4**

#include<bits/stdc++.h>

using namespace std;

struct Node

{

int data;

Node \*left;

Node \*right;

Node(int data)

{

this->data = data;

left = NULL;

right = NULL;

}

};

void traversal(Node\* root)

{

if (root == NULL)

{

return;

}

else

{

cout << root->data << " ";

traversal(root->left);

traversal(root->right);

}

}

Node\* insert(Node\* &root, int x)

{

if (root == NULL)

{

root = new Node(x);

}

else if (x >= root->data)

{

root->right = insert(root->right, x);

}

else if (x < root->data)

{

root->left = insert(root->left, x);

}

return root;

}

int height(Node\* root)

{

if (root == NULL)

{

return 0;

}

int left\_subtree = 1+height(root->left);

int right\_subtree = 1+height(root->right);

return max(left\_subtree, right\_subtree);

}int minimum(Node\* root)

{

if (root->left == NULL)

{

return root->data;

}

return min(minimum(root->left), root->data);

}

void search(Node\* root, int key, bool &present)

{

if (root == NULL)

{

return;

}

if (root->data == key)

{

present = true;

}

else if (root->data > key)

{

search(root->left, key, present);

}

else if (root->data <= key)

{

search(root->right, key, present);

}

}

void mirror(Node\* &root)

{

if (root == NULL)

{

return;

}

mirror(root->left);

mirror(root->right);

swap(root->left, root->right);

}

int main()

{

Node \*root = NULL;

int n;

cout << "Enter the number of nodes to be inserted in the BST-\n";

cin >> n;

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

{

int element;

cout << "Enter the data value to be inserted-\n";

cin >> element;

insert(root, element);

}

cout << "Displaying the elements of the BST-\n";

traversal(root);

cout << endl;cout << "Height of the tree is " << height(root) << endl;

cout << "Minimum value in the BST is " << minimum(root) << endl;

int key;

cout << "Enter the data value to be searched in the BST-\n";

cin >> key;

bool present = false;

search(root, key, present);

if (present == true)

{

cout << "Key is present in the BST!" << endl;

}

else

{

cout << "Key is not present in the BST!" << endl;

}

return 0;

}

**OUTPUT-**

Enter the number of nodes to be inserted in the BST-

3

Enter the data value to be inserted-

3

Enter the data value to be inserted-

2

Enter the data value to be inserted-

5

Displaying the elements of the BST-

3 2 5

Height of the tree is 2

Minimum value in the BST is 2

Enter the data value to be searched in the BST-

3

Key is present in the BST!

**PRACTICAL NO.5-**

#include <bits/stdc++.h>

using namespace std;

/\* Structure of a node in threaded binary tree \*/

struct Node

{

int key;

Node \*left, \*right;

bool isThreaded;

};

// Helper function to put the Nodes in inorder into queue

Node \*createThreaded(Node \*root)

{

if (root == NULL)

return NULL;

if (root->left == NULL &&

root->right == NULL)

return root;

if (root->left != NULL)

{

Node\* l = createThreaded(root->left);

l->right = root;

l->isThreaded = true;

}

if (root->right ==NULL)

return root;

return createThreaded(root->right);

}

Node \*leftMost(Node \*root)

{

while (root != NULL && root->left != NULL)

root = root->left;

return root;

}

void inorder (Node \*root)

{

if (root ==NULL) return;

Node \*cur =leftMost(root) ;

while (cur !=NULL)

{

cout << cur ->key << " ";

if (cur->isThreaded )

cur = cur->right;

else

cur = leftMost(cur->right);

}

}

Node \*newNode(int key)

{

Node \*temp = new Node;

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

temp->key = key;

return temp;

}

int main()

{

Node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

root->right->left = newNode(6);

root->right->right = newNode(7);

createThreaded(root);

cout << "Inorder traversal of created "

"threaded tree is\n";

inorder(root);

return 0;

}

**OUTPUT-**

Inorder traversal of created threaded tree is

4 2 5 1 6 3 7

**PRACTICAL NO.6-**

#include <iostream>

#include <queue>

using namespace std;

int adj\_mat[50][50] = {0, 0};

int visited[50] = {0};

void dfs(int s, int n, string arr[])

{

visited[s] = 1;

cout << arr[s] << " ";

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

{

if (adj\_mat[s][i] && !visited[i])

dfs(i, n, arr);

}

}

void bfs(int s, int n, string arr[])

{

bool visited[n];

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

visited[i] = false;

int v;

queue<int> bfsq;

if (!visited[s])

{

cout << arr[s] << " ";

bfsq.push(s);

visited[s] = true;

while (!bfsq.empty())

{

v = bfsq.front();

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

{

if (adj\_mat[v][i] && !visited[i])

{

cout << arr[i] << " ";

visited[i] = true;

bfsq.push(i);

}

}

bfsq.pop();

}

}

}

int main()

{

cout << "Enter no. of cities: ";

int n, u;

cin >> n;

string cities[n];

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

{

cout << "Enter city #" << i << " (Airport Code): ";

cin >> cities[i];}

cout << "\nYour cities are: " << endl;

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

cout << "city #" << i << ": " << cities[i] << endl;

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

{

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

{

cout << "Enter distance between " << cities[i] << " and " << cities[j] << " : ";

cin >> adj\_mat[i][j];

adj\_mat[j][i] = adj\_mat[i][j];

}

}

cout << endl;

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

cout << "\t" << cities[i] << "\t";

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

{

cout << "\n"

<< cities[i];

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

cout << "\t" << adj\_mat[i][j] << "\t";

cout << endl;

}

cout << "Enter Starting Vertex: ";

cin >> u;

cout << "DFS: ";

dfs(u, n, cities);

cout << endl;

cout << "BFS: ";

bfs(u, n, cities);

return 0;

}

**OUTPUT-**

Enter no. of cities: 2

Enter city #0 (Airport Code): 01

Enter city #1 (Airport Code): 02

Your cities are:

city #0: 01

city #1: 02

Enter distance between 01 and 02 : 10000

01 02

01 0 10000

02 10000 0

Enter Starting Vertex: a

DFS: 01 02

BFS: 01 02

**PRACTICAL NO.7-**

#include<iostream>

using namespace std;

class tree { int a[20][20],l,u,w,i,j,v,e,visited[20];

public:

void input();

void display();

void minimum();

};

void tree::input()

{

cout<<"Enter the no. of branches: ";

cin>>v;

for(i=0;i<v;i++)

{

visited[i]=0;

for(j=0;j<v;j++)

{

a[i][j]=999;

}

}

cout<<"\nEnter the no. of connections: ";

cin>>e;

for(i=0;i<e;i++)

{

cout<<"Enter the end branches of connections: "<<endl;

cin>>l>>u;

cout<<"Enter the phone company charges for this connection: ";

cin>>w;

a[l-1][u-1]=a[u-1][l-1]=w;

}

}

void tree::display()

{

cout<<"\nAdjacency matrix:";

for(i=0;i<v;i++)

{

cout<<endl;

for(j=0;j<v;j++)

{

cout<<a[i][j]<<" ";

}

cout<<endl;

}

}

void tree::minimum()

{

int p=0,q=0,total=0,min;

visited[0]=1;

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

{

min=999;

for(i=0;i<v;i++)

{

if(visited[i]==1)

{

for(j=0;j<v;j++)

{

if(visited[j]!=1)

{

if(min > a[i][j])

{

min=a[i][j];

p=i;

q=j;

}

}

}

}

}

visited[p]=1;

visited[q]=1;

total=total+min;

cout<<"Minimum cost connection is"<<(p+1)<<" -> "<<(q+1)<<" with charge : "<<min<< endl;

}

cout<<"The minimum total cost of connections of all branches is: "<<total<<endl;

}

int main()

{

int ch;

tree t;

do

{

cout<<"==========PRIM'S ALGORITHM================="<<endl;

cout<<"\n1.INPUT\n \n2.DISPLAY\n \n3.MINIMUM\n"<<endl;

cout<<"Enter your choice :"<<endl;

cin>>ch;

switch(ch)

{

case 1: cout<<"\*\*\*\*\*\*\*INPUT YOUR VALUES\*\*\*\*\*\*\*"<<endl;

t.input();

break;

case 2: cout<<"\*\*\*\*\*\*\*DISPLAY THE CONTENTS\*\*\*\*\*\*\*\*"<<endl;

t.display();

break;

case 3: cout<<"\*\*\*\*\*\*\*\*\*MINIMUM\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

t.minimum();

break;

}

}while(ch!=4);

return 0;

}

**OUTPUT-**

==========PRIM'S ALGORITHM=================

1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :

1

\*\*\*\*\*\*\*INPUT YOUR VALUES\*\*\*\*\*\*\*

Enter the no. of branches: 2

Enter the no. of connections: 1

Enter the end branches of connections:

1

2

Enter the phone company charges for this connection: 10

==========PRIM'S ALGORITHM=================

1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :

2

\*\*\*\*\*\*\*DISPLAY THE CONTENTS\*\*\*\*\*\*\*\*

Adjacency matrix:

999 10

10 999

==========PRIM'S ALGORITHM=================

1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :

3

\*\*\*\*\*\*\*\*\*MINIMUM\*\*\*\*\*\*\*\*\*\*\*\*

Minimum cost connection is1 -> 2 with charge : 10

The minimum total cost of connections of all branches is: 10

==========PRIM'S ALGORITHM=================

1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :

**PRACTICAL NO.8-**

#include <iostream>

using namespace std;

void con\_obst(void);

void print(int, int);

float a[20], b[20], wt[20][20], c[20][20];

int r[20][20], n;

int main()

{

int i;

cout << "\n\*\*\*\*\*\* PROGRAM FOR OBST \*\*\*\*\*\*\n";

cout << "\nEnter the no. of nodes : ";

cin >> n;

cout << "\nEnter the probability for successful search :: ";

cout << "\n————————————————\n";

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

{

cout << "p[" << i << "]";

cin >> a[i];

}

cout << "\nEnter the probability for unsuccessful search :: ";

cout << "\n————————————————\n";

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

{

cout << "q[" << i << "]";

cin >> b[i];

}

con\_obst();

print(0, n);

cout << endl;

}

void con\_obst(void)

{

int i, j, k, l, min;

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

{ // Initialisation

c[i][i] = 0.0;

r[i][i] = 0;

wt[i][i] = b[i];

// for j-i=1 can be j=i+1 wt[i][i + 1] = b[i] + b[i + 1] + a[i + 1];

c[i][i + 1] = b[i] + b[i + 1] + a[i + 1];

r[i][i + 1] = i + 1; }c[n][n] = 0.0;

r[n][n] = 0;

wt[n][n] = b[n];

// for j-i=2,3,4....,n for (i = 2; i <= n; i++)

{

for (j = 0; j <= n - i; j++)

{

wt[j][j + i] = b[j + i] + a[j + i] + wt[j][j + i - 1];

c[j][j + i] = 9999;

for (l = j + 1; l <= j + i; l++)

{

if (c[j][j + i] > (c[j][l - 1] + c[l][j + i]))

{

c[j][j + i] = c[j][l - 1] + c[l][j + i];

r[j][j + i] = l;

}

}

c[j][j + i] += wt[j][j + i];

}

cout << endl;

}

cout << "\n\nOptimal BST is :: ";

cout << "\nw[0][" << n << "] :: " << wt[0][n];

cout << "\nc[0][" << n << "] :: " << c[0][n]; cout << "\nr[0][" << n << "] :: " << r[0][n];

}

void print(int l1, int r1)

{

if (l1 >= r1)

return;

if (r[l1][r[l1][r1] - 1] != 0)

cout << "\n Left child of " << r[l1][r1] << " :: " << r[l1][r[l1][r1] - 1];

if (r[r[l1][r1]][r1] != 0)

cout << "\n Right child of " << r[l1][r1] << " :: " << r[r[l1][r1]][r1];

print(l1, r[l1][r1] - 1);

print(r[l1][r1], r1);

return;

}

**OUTPUT-**

\*\*\*\*\*\* PROGRAM FOR OBST \*\*\*\*\*\*

Enter the no. of nodes : 2

Enter the probability for successful search ::

ùùùùùùùùùùùùùùùù

p[1]2

p[2]3

Enter the probability for unsuccessful search ::

ùùùùùùùùùùùùùùùù

q[0]1

q[1]2

q[2]3

Optimal BST is ::

w[0][2] :: 6

c[0][2] :: 11

r[0][2] :: 2

Left child of 2 :: 1

**PRACTICAL NO.9-**

#include<iostream>

#include<string.h>

using namespace std;

struct node

{

char k[20];

char m[20];

class node \*left;

class node \* right;

};

class dict

{

public:

node \*root;

void create();

void disp(node \*);

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

int search(node \*,char []);

int update(node \*,char []);

node\* del(node \*,char []);

node \* min(node \*);

};

void dict :: create()

{

class node \*temp;

char ch;

do

{

temp = new node;

cout<<"\nEnter Keyword :: ";

cin>>temp->k;

cout<<"Enter Meaning :: ";

cin>>temp->m;

temp->left = NULL;

temp->right = NULL;

if(root == NULL)

{

root = temp;

}

else

{

insert(root, temp);

}

cout<<"\nDo u want to add more (y/n):";

cin>>ch;

}

while(ch =='y' || ch=='Y');

}

void dict :: insert(node \* root,node \*temp)

{

if(strcmp (temp->k, root->k) < 0 )

{

if(root->left == NULL)

root->left = temp;

else

insert(root->left,temp);

}

else

if(root->right == NULL)

root->right = temp;

else

insert(root->right,temp);

}

void dict:: disp(node \* root)

{

if( root != NULL)

{

disp(root->left);

cout<<"\n"<<root->k<<"\t\t"<<root->m;

disp(root->right);

}

}

int dict :: search(node \* root,char k[20])

{

int c=0;

while(root != NULL)

{

if(strcmp (k,root->k) == 0)

{

cout<<"\nNo of Comparisons ::"<<c;

return 1;

}

if(strcmp (k, root->k) < 0)

root = root->left;

if(strcmp (k, root->k) > 0)

root = root->right;

}

return 0;

}

int dict :: update(node \* root,char k[20])

{

while(root != NULL)

{

if(strcmp (k,root->k) == 0)

{

cout<<"\nEnter New Meaning of Keyword "<<root->k<<" :: ";

cin>>root->m;

return 1;

}

if(strcmp (k, root->k) < 0)

root = root->left;

if(strcmp (k, root->k) > 0)

root = root->right;

}

return 0;

}

node\* dict :: del(node \* root,char k[20])

{

node \*temp;

if(root == NULL)

{

cout<<"\nNo Element Found";

return root;

}

if (strcmp(k,root->k) < 0)

{

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

return root;

}

if (strcmp(k,root->k) > 0)

{

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

return root;

}

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

{

temp = root;

delete temp;

return NULL;

}

if(root->right==NULL)

{

temp = root;

root = root->left;

delete temp;

return root;

}

else if(root->left==NULL)

{

temp = root;

root = root->right;

delete temp;

return root;

}

temp = min(root->right);

strcpy(root->k,temp->k);

root->right = del(root->right, temp->k);

return root;

}

node \* dict :: min(node \*q)

{

while(q->left != NULL)

{

q = q->left;

}

return q;

}

int main()

{

int ch,loop=1;

dict d;

d.root = NULL;

while(loop==1)

{

cout<<"\n-----Menu-----"<<"\n1.Create\n2.Display\n3.Search\n4.Update\n5.Delete\n6.Exit\nEnter Your Choice : ";

cin>>ch;

switch(ch)

{

case 1:

d.create();

break;

case 2:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";

}

else

{

cout<<"Keyword \t Meaning\n";

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

d.disp(d.root);

}

break;

case 3:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";

}

else

{

cout<<"\nEnter Keyword which u want to search :: ";

char k[20];

cin>>k;

int f=d.search(d.root,k);

if( f == 1)

cout<<"\nKeyword Found";

else

cout<<"\nKeyword Not Found";

}

break;

case 4:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";

}

else

{

cout<<"\nEnter Keyword which meaning want to update :: ";

char k[20];

cin>>k;

if(d.update(d.root,k) == 1)

cout<<"\nMeaning Updated";

else

cout<<"\nKeyword Not Found";

}

break;

case 5:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";

}

else

{

cout<<"\nEnter Keyword which u want to delete :: ";

char k[20];

cin>>k;

if(d.root == NULL)

{

cout<<"\nKeyword Not Found";

}

else

{

d.root = d.del(d.root,k);

}

}

break;

case 6:

loop=0;

cout<<"Thank You!";

break;

default:

cout<<"You entered something wrong";

break;

}

}

return 0;

}

**OUTPUT-**

----Menu-----

1.Create

2.Display

3.Search

4.Update

5.Delete

6.Exit

Enter Your Choice : 1

Enter Keyword :: abc

Enter Meaning :: hwllo

Do u want to add more (y/n):y

Enter Keyword :: pqr

Enter Meaning :: bye

Do u want to add more (y/n):n

-----Menu-----

1.Create

2.Display

3.Search

4.Update

5.Delete

6.Exit

Enter Your Choice : 2

Keyword Meaning

----------------------

abc hwllo

pqr bye

-----Menu-----

1.Create

2.Display

3.Search

4.Update

5.Delete

6.Exit

Enter Your Choice : 3

Enter Keyword which u want to search :: pqr

No of Comparisons ::0

Keyword Found

-----Menu-----

1.Create

2.Display

3.Search

4.Update

5.Delete

6.Exit

Enter Your Choice : 4

Enter Keyword which meaning want to update :: pqr

Enter New Meaning of Keyword pqr :: mauu

Meaning Updated

-----Menu-----

1.Create

2.Display

3.Search

4.Update

5.Delete

6.Exit

Enter Your Choice : 5

Enter Keyword which u want to delete :: pqr

-----Menu-----

1.Create

2.Display

3.Search

4.Update

5.Delete

6.Exit

Enter Your Choice : 2

Keyword Meaning

----------------------

abc hwllo

-----Menu-----

1.Create

2.Display

3.Search

4.Update

5.Delete

6.Exit

Enter Your Choice : 6

Thank You!

**PRACTICAL NO.10-**

#include <iostream>

#include <queue>

#include <string>

using namespace std;

// Structure to represent a patient

struct Patient {

string name;

int priority; // Priority of the patient, lower value means higher priority

// Constructor

Patient(const string& name, int priority) : name(name), priority(priority) {}

// Overloading the < operator to compare patients based on priority

bool operator<(const Patient& other) const {

// Higher priority patients should come before lower priority ones

return priority > other.priority;

}

};

// Function to display the menu options

void displayMenu() {

cout << "Hospital Management System\n";

cout << "1. Add Patient\n";

cout << "2. Serve Patient\n";

cout << "3. Display Patients\n";

cout << "4. Exit\n";

}

int main() {

priority\_queue<Patient> patients; // Priority queue to store patients

int choice;

do {

displayMenu();

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

string name;

int priority;

cout << "Enter patient name: ";

cin >> name;

cout << "Enter patient priority: ";

cin >> priority;

patients.push(Patient(name, priority));

cout << "Patient added successfully.\n";

break;

}

case 2: {

if (patients.empty()) {

cout << "No patients to serve.\n";

} else {

cout << "Serving patient: " << patients.top().name << endl;

patients.pop();}

break;

}

case 3: {

if (patients.empty()) {

cout << "No patients in the queue.\n";

} else {

cout << "Patients in the queue:\n";

priority\_queue<Patient> temp = patients; // Create a copy of the priority

while (!temp.empty()) {

cout << "Name: " << temp.top().name << ", Priority: " <<

temp.top().priority << endl;

temp.pop();

}

}

break;

}

case 4: {

cout << "Exiting program.\n";

break;

}

default: {

cout << "Invalid choice. Please try again.\n";

break;

}

}

} while (choice != 4);

return 0;

}

**OUTPUT-**

Hospital Management System

1. Add Patient

2. Serve Patient

3. Display Patients

4. Exit

Enter your choice: 1

Enter patient name: abc

Enter patient priority: 2

Patient added successfully.

Hospital Management System

1. Add Patient

2. Serve Patient

3. Display Patients

4. Exit

Enter your choice: 2

Serving patient: abc

Hospital Management System

1. Add Patient

2. Serve Patient

3. Display Patients

4. Exit

Enter your choice: 3

No patients in the queue.

Hospital Management System

1. Add Patient

2. Serve Patient

3. Display Patients

4. Exit

Enter your choice: 1

Enter patient name: sk

Enter patient priority: 4

Patient added successfully.

Hospital Management System

1. Add Patient

2. Serve Patient

3. Display Patients

4. Exit

Enter your choice: 3

Patients in the queue:

Name: sk, Priority: 4

Hospital Management System

1. Add Patient

2. Serve Patient

3. Display Patients

4. Exit

Enter your choice:

**PRACTICAL NO.11-**

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

struct Student {

int rollNumber;

char name[50];

char division;

char address[100];

};

void addStudent() {

ofstream outFile("students.dat", ios::binary | ios::app);

Student student;

cout << "Enter Roll Number: ";

cin >> student.rollNumber;

cin.ignore();

cout << "Enter Name: ";

cin.getline(student.name, 50);

cout << "Enter Division: ";

cin >> student.division;

cin.ignore();

cout << "Enter Address: ";

cin.getline(student.address, 100);

outFile.write(reinterpret\_cast<char\*>(&student), sizeof(Student));

outFile.close();

}

void deleteStudent(int rollNumber) {

ifstream inFile("students.dat", ios::binary);

ofstream outFile("temp.dat", ios::binary);

Student student;

bool found = false;

while (inFile.read(reinterpret\_cast<char\*>(&student), sizeof(Student))) {

if (student.rollNumber != rollNumber) {

outFile.write(reinterpret\_cast<char\*>(&student), sizeof(Student));

}

else {

found = true;

}

}

inFile.close();

outFile.close();

if (found) {

remove("students.dat");

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

cout << "Student record deleted successfully." << endl;

}

else {

remove("temp.dat");cout << "Student record not found." << endl;

}

}

void displayStudent(int rollNumber) {

ifstream inFile("students.dat", ios::binary);

Student student;

bool found = false;

while (inFile.read(reinterpret\_cast<char\*>(&student), sizeof(Student))) {

if (student.rollNumber == rollNumber) {

cout << "Roll Number: " << student.rollNumber << endl;

cout << "Name: " << student.name << endl;

cout << "Division: " << student.division << endl;

cout << "Address: " << student.address << endl;

found = true;

break;

}

}

inFile.close();

if (!found) {

cout << "Student record not found." << endl;

}

}

int main() {

int choice;

int rollNumber;

do {

cout << "----- Student Information System -----" << endl;

cout << "1. Add Student" << endl;

cout << "2. Delete Student" << endl;

cout << "3. Display Student" << endl;

cout << "4. Quit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

addStudent();

break;

case 2:

cout << "Enter Roll Number of student to delete: ";

cin >> rollNumber;

deleteStudent(rollNumber);

break;

case 3:

cout << "Enter Roll Number of student to display: ";

cin >> rollNumber;

displayStudent(rollNumber);

break;

case 4:

break;

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

break;

}

cout << endl;

} while (choice != 4);

return 0;

}

**OUTPUT-**

----- Student Information System -----

1. Add Student

2. Delete Student

3. Display Student

4. Quit

Enter your choice: 1

Enter Roll Number: 55

Enter Name: abc

Enter Division: c

Enter Address: pune

----- Student Information System -----

1. Add Student

2. Delete Student

3. Display Student

4. Quit

Enter your choice: 3

Enter Roll Number of student to display: 55

Roll Number: 55

Name: abc

Division: c

Address: pune

----- Student Information System -----

1. Add Student

2. Delete Student

3. Display Student

4. Quit

Enter your choice:

**PRACTICAL NO.12-**

#include <bits/stdc++.h>

#define max 20

using namespace std;

struct employee {

string name;

long int code;

string designation;

int exp;

int age;

};

int num;

void showMenu();

employee emp[max], tempemp[max],

sortemp[max], sortemp1[max];

// void build()

// {

// cout << "Maximum Entries can be "<< max << "\n";

// cout << "Enter the number of Entries required: ";

// cin >> num;

// if (num > 20) {

// cout << "Maximum number of Entries are 20.\n";

// num = 20;

// }

// cout << "Enter the following data:\n";4

// for (int i = 0; i < num; i++) {

// cout << "Name: ";

// cin >> emp[i].name;

// cout << "Employee ID: ";

// cin >> emp[i].code;

// cout << "Designation: ";

// cin >> emp[i].designation;

// cout << "Experience: ";

// cin >> emp[i].exp;

// cout << "Age: ";

// cin >> emp[i].age;

// }

// showMenu();

// }

void insert()

{

if (num < max) {

int i = num;

num++;

cout << "Enter the information of the Employees.\n";

cout << "Name: ";

cin >> emp[i].name;

cout << "Employee ID: ";

cin >> emp[i].code;

cout << "Designation: ";

cin >> emp[i].designation;

cout << "Experience: ";

cin >> emp[i].exp;

cout << "Age: ";

cin >> emp[i].age;

}

else {

cout << "Employee Size Full\n";

}

showMenu();

}

// Function to delete record at index i

void deleteIndex(int i)

{

for (int j = i; j < num - 1; j++) {

emp[j].name = emp[j + 1].name;

emp[j].code = emp[j + 1].code;

emp[j].designation = emp[j + 1].designation;

emp[j].exp = emp[j + 1].exp;

emp[j].age = emp[j + 1].age;

}

return;

}

// Function to delete record

void deleteRecord()

{

cout << "Enter the Employee ID to Delete Record: ";

int code;

cin >> code;

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

if (emp[i].code == code) {

deleteIndex(i);

num--;

break;

}

}

showMenu();

}

void searchRecord()

{

cout << "Enter the Employee ID to Search Record: ";

int code;

cin >> code;

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

// If the data is found

if (emp[i].code == code) {

cout << "Name: "

<< emp[i].name << "\n";

cout << "Employee ID: "

<< emp[i].code << "\n";

cout << "Designation: "

<< emp[i].designation << "\n";

cout << "Experience: "

<< emp[i].exp << "\n";

cout << "Age: "

<< emp[i].age << "\n";

break;

}

}

showMenu();

}

void showMenu()

{

cout << "------------------------- Employee Management System -------------------------\n\n";

cout << "Available Options:\n\n";

cout << "Insert New Entry (1)\n";

cout << "Delete Entry (2)\n";

cout << "Search a Record (3)\n";

cout << "Exit (4)\n";

int option;

cin >> option;

if (option == 1) {

insert();

}

else if (option == 2) {

deleteRecord();

}

else if (option == 3) {

searchRecord();

}

else if (option == 4) {

return;

}

else {

cout << "Expected Options are 1 to 4";

showMenu();

}

}

int main()

{

showMenu();

return 0;

}

**OUTPUT-**

------------------------- Employee Management System -------------------------

Available Options:

Insert New Entry (1)

Delete Entry (2)

Search a Record (3)

Exit (4)

1

Enter the information of the Employees.

Name: pari

Employee ID: 37

Designation: employee

Experience: 4

Age: 34

------------------------- Employee Management System -------------------------

Available Options:

Insert New Entry (1)

Delete Entry (2)

Search a Record (3)

Exit (4)

3

Enter the Employee ID to Search Record: 37

Name: pari

Employee ID: 37

Designation: employee

Experience: 4

Age: 34