1. **Group A**

**Assignment 1**

**Aim: Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client’s telephone number. Make use of two collision handling techniques & compare them using number of comparisons required to find a set of telephone numbers.**

**Code:**

# Create HashTable

HashTable = [[] for \_ in range(10)]

# Display records

def display\_hash(hashTable):

for i in range(len(hashTable)):

print(i,end="")

for j in hashTable[i]:

if j !={}:

print("-->", end = " ")

print(j, end = " ")

print()

# Hash function

def Hashing(keyvalue):

return keyvalue % len(HashTable)

# Insert Function to add

# values to the hash table

#Chaining

def insert(Hashtable, keyvalue, value):

hash\_key = Hashing(keyvalue)

Hashtable[hash\_key].append({value:keyvalue})

# Search record

def find(Hashtable, value):

f=0

for i in range( 1, 10):

for j in Hashtable[i]:

if value in j:

print (j)

f=1

return j

break

if f==0:

print("Not Found")

return 0

# update record

def update(Hashtable, keyvalue, value):

pos={}

hash\_key = Hashing(keyvalue)

pos=find(Hashtable,value)

if pos !={}:

del(pos[value])

Hashtable[hash\_key].append({value:keyvalue})

# delete record

def delete(Hashtable, value):

pos={}

pos=find(Hashtable,value)

if pos !={}:

del(pos[value])

# Insert Function to add

# values to the hash table

#linear probing

def insert\_l(Hashtable, keyvalue, value):

hash\_key = Hashing(keyvalue)

key=hash\_key

while Hashtable[hash\_key] !=[]:

hash\_key = Hashing(hash\_key+1)

if Hashtable[hash\_key] ==[]:

Hashtable[hash\_key].append({value:keyvalue})

# Driver Code

print ("...................Insert Telephone Record........................")

insert\_l(HashTable, 9112335640, 'anil')

insert\_l(HashTable, 9321232499, 'Meera')

insert\_l(HashTable, 9567432912, 'Maya')

insert\_l(HashTable, 9999822343, 'Anoop')

insert\_l(HashTable, 9878965433, 'Pooja')

insert\_l(HashTable, 9223343562, 'Noida')

print ("...................Display Telephone Record........................")

display\_hash (HashTable)

print ("...................Search Telephone Record........................")

find(HashTable,"Meera")

print ("...................Update Telephone Record........................")

update(HashTable, 9563245187, 'Maya')

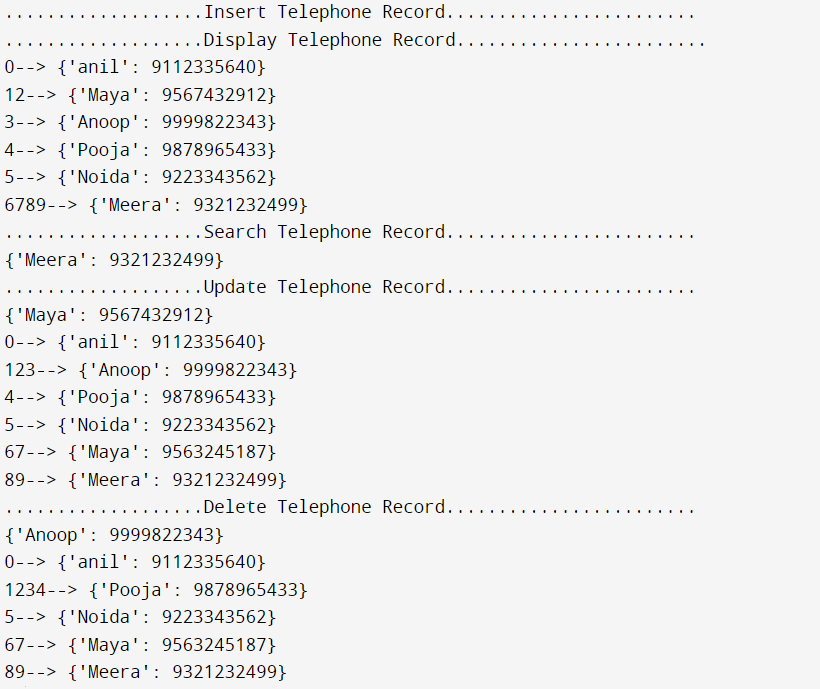
display\_hash (HashTable)

print ("...................Delete Telephone Record........................")

delete(HashTable,'Anoop')

display\_hash(HashTable)

**Output:**



1. **Group A**

**Assignment 2**

**Aim: Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with/without replacement. Data: set of (key, value) pairs, keys are mapped to values, keys must be comparable, keys must be unique standard operations: insert (key, value), find (key), delete (key).**

**Code:**

class Set:

def \_\_init\_\_(self, data=None):

self.data = {}

if data != None:

if len(data) != len(set(data)):

data = set(data)

for d in data:

self.data[d] = d

def insert(self, i):

if i in self.data.keys():

return 'Already in set'

self.data[i] = i

def remove(self, i):

if i not in self.data.keys():

return 'Not in set'

self.data.pop(i)

def size(self):

return len(self.data.keys())

def contains(self, i):

if i in self.data.keys():

return True

return False

def union(self, otherSet):

setData = list(set(self.data.keys()) | set(otherSet.data.keys()))

unionSet = Set(setData)

return unionSet

def intersection(self, otherSet):

setData = list(set(self.data.keys()) & set(otherSet.data.keys()))

intersectionSet = Set(setData)

return intersectionSet

def difference(self, otherSet):

setData = list(set(self.data.keys()) ^ set(otherSet.data.keys()))

differenceSet = Set(setData)

return differenceSet

def subset(self, otherSet):

if set(self.data.keys()) < set(otherSet.data.keys()):

return True

return False

def \_\_repr\_\_(self):

return '['+', '.join(str(x) for x in self.data.keys())+']'

def main():

set1 = Set([])

set2 = Set([])

n1 = int(input("Enter the number of elements in set1: "))

for i in range(n1):

data\_name = input("Enter the elements : ")

#insert

set1.insert(data\_name)

n2 = int(input("Enter the number of elements in set2: "))

for i in range(n2):

data\_name = input("Enter the elements : ")

#insert

set2.insert(data\_name)

#users choice

while(True):

choice = int(input("Enter 1:Union 2: Intersection 3: size 4.subset 5.difference "))

if (choice==1):

print("Set :",set1.union(set2))

elif (choice==2):

print("Set :",set1.intersection(set2))

elif (choice==3):

print("Size of set 1 :",set1.size())

print("Size of set 2 :",set2.size())

elif (choice==4):

print("Subset :",set1.subset(set2))

elif (choice==5):

print("Set :",print(set1.difference(set2)))

else:

print("Invalid Choice")

c=input("Do you want to continue:Y/N")

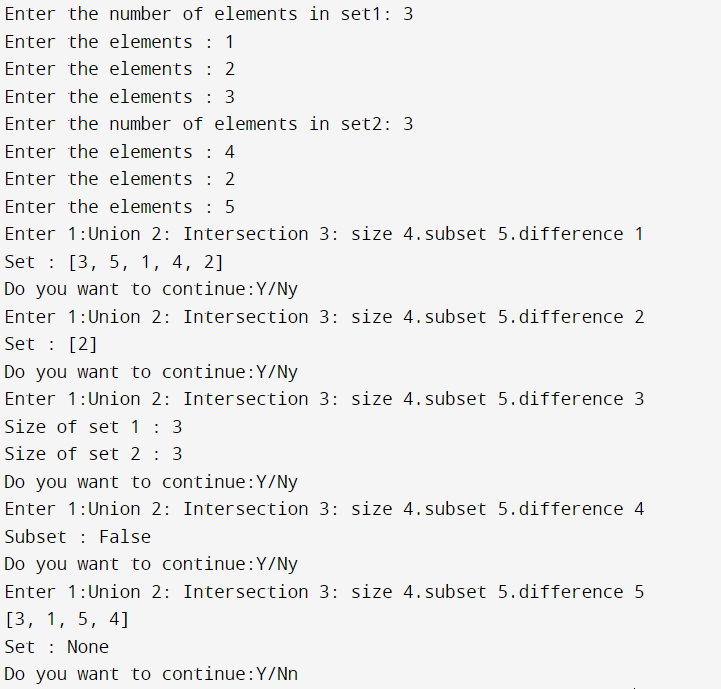
if(c=='N'):

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**



1. **Group B**

**Assignment 1**

**Aim: Beginning with an empty binary search tree, construct binary search tree by inserting the values in the order given. After constructing a binary tree – 1. Insert new node 2. Find number of nodes in longest path from root. 3. Minimum data value found in the tree 4. Change a tree so that the roles of the left and right pointers are swapped at every node. 5. Search a value.**

**Code:**

#include<iostream>

#include<math.h>

using namespace std;

struct Bstnode

{

int data;

Bstnode \*left;

Bstnode \*right;

};

class Btree

{

int n;

int x;

int flag;

public:

Bstnode \* root;

Btree()

{

root = NULL;

}

Bstnode \*GetNewNode(int in\_data)

{

Bstnode \* ptr = new Bstnode();

ptr->data = in\_data;

ptr->left = NULL;

ptr->right = NULL;

return ptr;

}

Bstnode \*insert( Bstnode \*temp , int in\_data)

{

if( temp == NULL )

{

temp = GetNewNode(in\_data);

}

else if( temp->data > in\_data)

{

temp->left = insert(temp->left , in\_data);

}

else

{

temp->right = insert( temp->right , in\_data);

}

return temp;

}

void input()

{

cout<<"ENTER NUMBER OF ELEMENTS IN THE BST : ";

cin>>n;

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

{

cout<<"Element = ";

cin>>x;

root = insert(root , x);

}

}

void search(Bstnode\* root, int key, Bstnode\* parent)

{

// if the key is not present in the key

if (root == NULL)

{

cout << "Key not found";

return;

}

// if the key is found

if (root->data == key)

{

if (parent == NULL) {

cout << "The node with key " << key << " is root node"<<endl;

}

else if (key < parent->data)

{

cout << "The given key is the left node of the node with key"<< parent->data<<endl;

}

else {

cout << "The given key is the right node of the node with key "

<< parent->data<<endl;

}

return;

}

if (key < root->data) {

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

} else {

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

}

}

void minvalue(Bstnode \*temp)

{

while(temp->left != NULL)

{

temp = temp->left;

}

cout<<"MINIMUM VALUE = "<<temp->data<<endl;

}

void printTree(Bstnode\* node) {

if (node == NULL) {

return;

}

printTree(node->left);

cout << node->data << " ";

printTree(node->right);

}

void convertTreeToItsMirror(Bstnode\* node) {

if (node == NULL) {

return;

}

else {

Bstnode\* temp;

convertTreeToItsMirror(node->left);

convertTreeToItsMirror(node->right);

temp = node->left;

node->left = node->right;

node->right = temp;

}

}

void display()

{

cout<<endl<<"--- INORDER TRAVERSAL ---"<<endl;

inorder(root);

cout<<endl;

cout<<endl<<"--- POSTORDER TRAVERSAL ---"<<endl;

postorder(root);

cout<<endl;

cout<<endl<<"--- PREORDER TRAVERSAL ---"<<endl;

preorder(root);

cout<<endl;

}

void inorder(Bstnode \*temp)

{

if(temp != NULL)

{

inorder(temp->left);

cout<<temp->data<<" ";

inorder(temp->right);

}

}

void postorder(Bstnode \*temp)

{

if(temp != NULL)

{

postorder(temp->left);

postorder(temp->right);

cout<<temp->data<<" ";

}

}

void preorder(Bstnode \*temp)

{

if(temp != NULL)

{

cout<<temp->data<<" ";

preorder(temp->left);

preorder(temp->right);

}

}

int depth(Bstnode \*temp)

{

if(temp == NULL)

return 0;

return (max((depth(temp->left)),(depth(temp->right))) +1);

}

};

int main()

{

Btree obj;

int ch;

char ans;

do

{

cout<<"1) Insert new node"

"2)Search a value "

"3)Minimum data value found in the tree "

"4)Change a tree so that the roles of the left and right pointers are swapped at every node"

"5)Find number of nodes in longest path from root"<<endl;

cin>>ch;

switch(ch){

case 1:obj.input();obj.display();break;

case 2:obj.search(obj.root, 5, NULL);break;

case 3:obj.minvalue(obj.root);break;

case 4:cout<<"Mirror of the tree"<<endl;

obj.convertTreeToItsMirror(obj.root);

obj.printTree(obj.root);break;

case 5:cout<<"\n"<<"Depth of the tree"<<endl;

cout<<obj.depth(obj.root)<<endl;break;

}

cout<<"\nDo you want to continue : (Y/N)"<<endl;

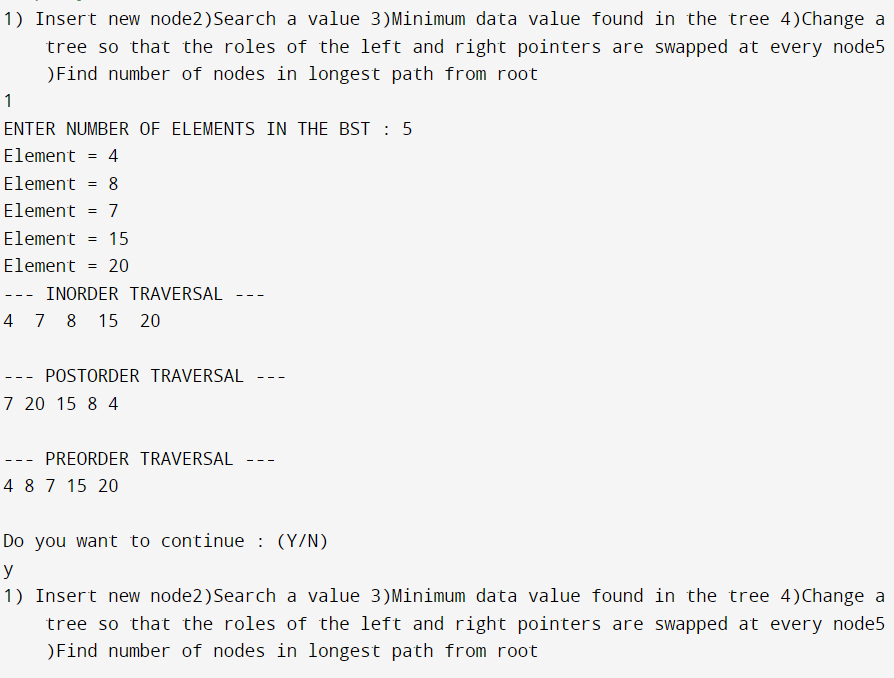
cin>>ans;

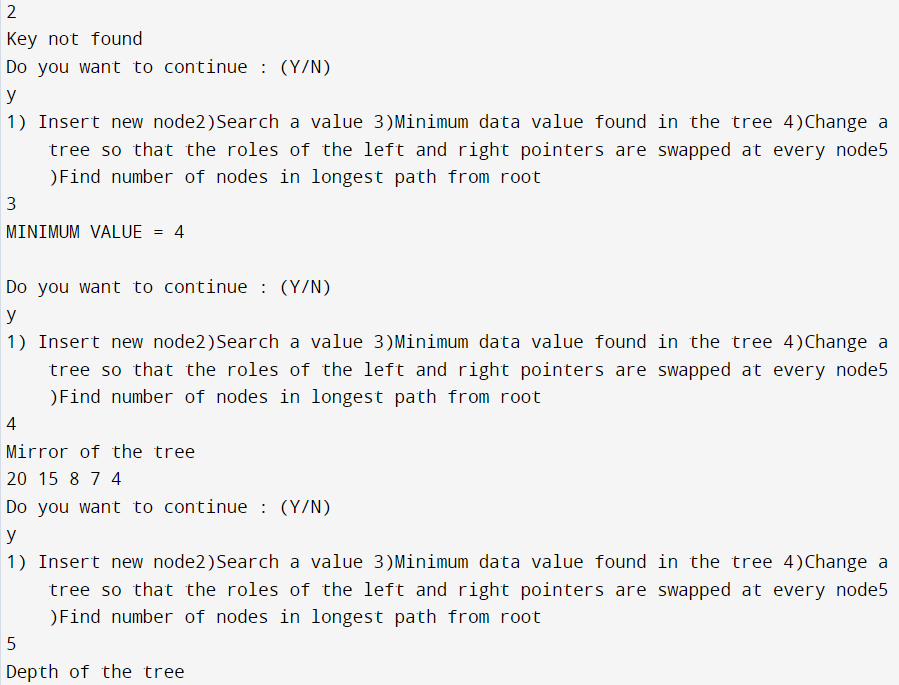
}while (ans=='Y'||ans=='y');

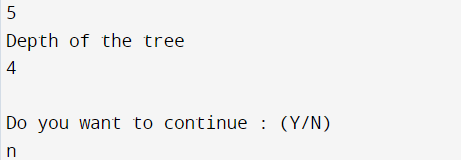
return 0;

}

**Output:**







Assignment No.2 A book consists of chapters ,chapters consist of section & section consist of subsection .Construct a tree& print the nodes.Find the time & space requirements of your method .

Program:-

#include<iostream>

using namespace std;

struct node {

string data;

struct node \*left;

struct node \*right;

};

struct node\* insert(struct node \*\*q,string data) {

struct node\* temp = new node();

temp = \*q;

struct node\* newnode = new node();

if(temp->left == NULL)

{

newnode->data = data;

newnode->left = NULL;

newnode->right = NULL;

temp->left = newnode; }

else

{

newnode->data = data;

newnode->left = NULL;

newnode->right = NULL;

temp->right = newnode; }

return newnode;

}

void display(struct node \*root)

{

if(root == NULL)

return;

cout<<root->data<<endl;

display(root->left);

display(root->right);

}

int main()

{

struct node \*book = new node();

struct node \*chap = new node();

struct node \*sec = new node();

struct node \*sub\_sec = new node();

string book\_name, chapter,section,sub\_section;

cout<<"Enter the book name : ";

getline(cin,book\_name);

book->data = book\_name;

book->left = NULL;

book->right = NULL;

for(int i = 0;i<2;i++)

{

cout<<"Enter the chapter name : ";

getline(cin,chapter);

chap = insert(&book,chapter);

for(int j = 0;j<2;j++)

{

cout<<"Enter the "<<chapter<<"'s section name : ";

getline(cin,section);

sec = insert(&chap,section);

for(int k = 0;k<2;k++) {

cout<<"Enter the "<<section<<"'s subsection name : ";

getline(cin,sub\_section);

sub\_sec = insert(&sec,sub\_section); } } }

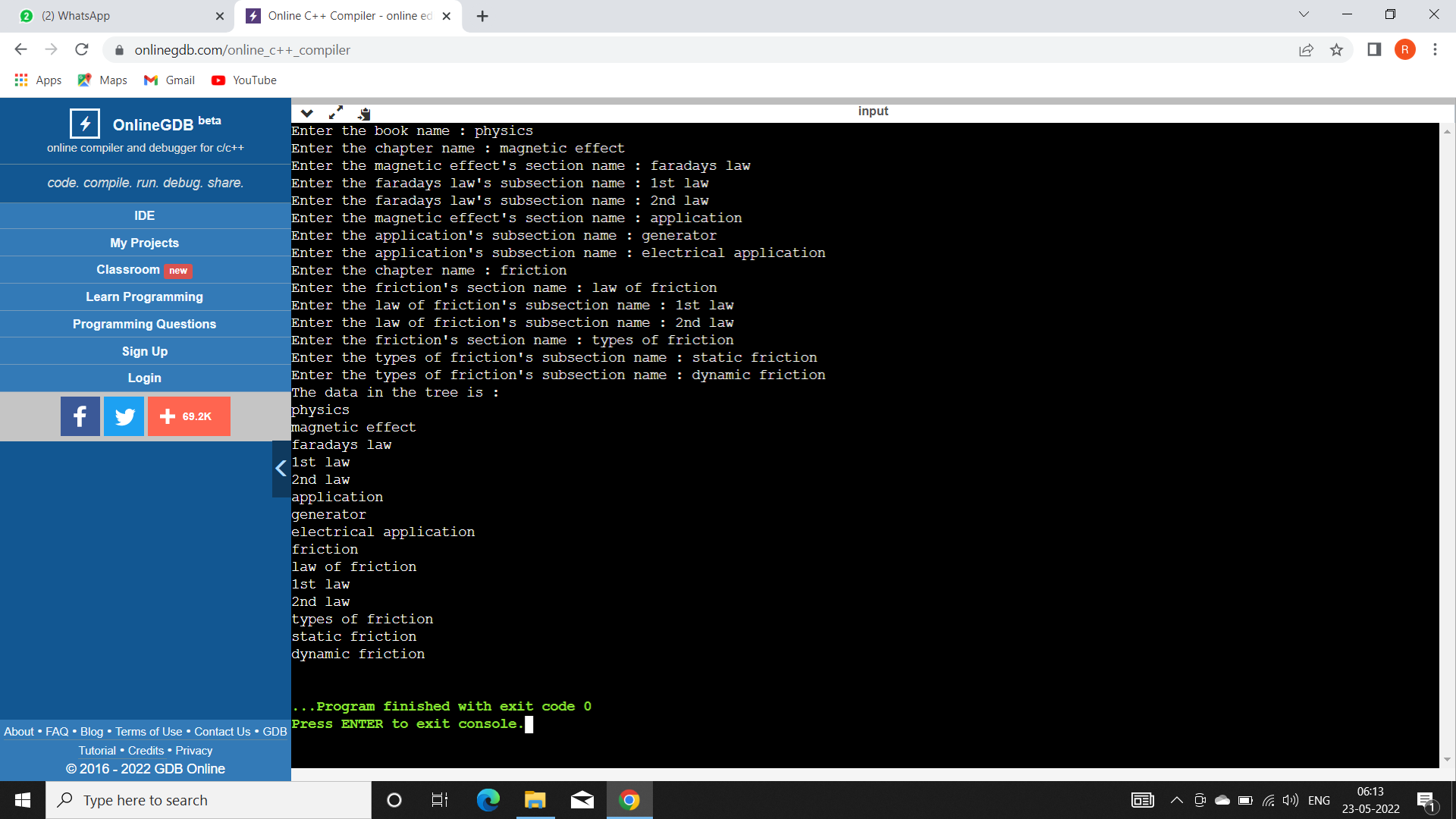
cout<<"The data in the tree is : "<<endl;

display(book);

return 0;

}

Output:-



1. **Group B**

**Assignment 2**

**Aim: Construct an expression tree from the given prefix expression. Ex. + - - a\*bc/def and traverse it using post order traversal and then delete the entire tree.**

**Code:**

#include <iostream>

#include <string.h>

using namespace std;

struct node

{

char data;

node \*left;

node \*right;

};

class tree

{

char prefix[20];

public:

node \*top;

void expression(char[]);

void display(node \*);

void non\_rec\_postorder(node \*);

void del(node \*);

};

class stack1

{

node \*data[30];

int top;

public:

stack1()

{

top = -1;

}

int empty()

{

if (top == -1)

return 1;

return 0;

}

void push(node \*p)

{

data[++top] = p;

}

node \*pop()

{

return (data[top--]);

}

};

void tree::expression(char prefix[])

{

char c;

stack1 s;

node \*t1, \*t2;

int len, i;

len = strlen(prefix);

for (i = len - 1; i >= 0; i--)

{

top = new node;

top->left = NULL;

top->right = NULL;

if (isalpha(prefix[i]))

{

top->data = prefix[i];

s.push(top);

}

else if (prefix[i] == '+' || prefix[i] == '\*' || prefix[i] == '-' || prefix[i] == '/')

{

t2 = s.pop();

t1 = s.pop();

top->data = prefix[i];

top->left = t2;

top->right = t1;

s.push(top);

}

}

top = s.pop();

}

void tree::display(node \*root)

{

if (root != NULL)

{

cout << root->data;

display(root->left);

display(root->right);

}

}

void tree::non\_rec\_postorder(node \*top)

{

stack1 s1, s2; /\*stack s1 is being used for flag . A NULL data implies that the right subtree has not been visited \*/

node \*T = top;

cout << "\n";

s1.push(T);

while (!s1.empty())

{

T = s1.pop();

s2.push(T);

if (T->left != NULL)

s1.push(T->left);

if (T->right != NULL)

s1.push(T->right);

}

while (!s2.empty())

{

top = s2.pop();

cout << top->data;

}

}

void tree::del(node \*node)

{

if (node == NULL)

return;

/\* first delete both subtrees \*/

del(node->left);

del(node->right);

/\* then delete the node \*/

cout <<endl<<"Deleting node : " << node->data<<endl;

free(node);

}

int main()

{

char expr[20];

tree t;

cout <<"Enter prefix Expression : ";

cin >> expr;

cout << expr;

t.expression(expr);

//t.display(t.top);

//cout<<endl;

t.non\_rec\_postorder(t.top);

t.del(t.top);

// t.display(t.top);

}

**Output:**



1. **Group B**

**Assignment 3**

**Aim: Convert a given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm.**

**Code:**

#include <iostream>

using namespace std;

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

struct Node

{

int key;

Node \*left, \*right;

// Used to indicate whether the right pointer

// is a normal right pointer or a pointer

// to inorder successor.

bool isThreaded;

};

// Converts tree with given root to threaded

// binary tree.

// This function returns rightmost child of

// root.

Node \*createThreaded(Node \*root)

{

// Base cases : Tree is empty or has single node

if (root == NULL)

return NULL;

if (root->left == NULL &&

root->right == NULL)

return root;

// Find predecessor if it exists

if (root->left != NULL)

{

// Find predecessor of root (Rightmost child in left subtree)

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

// Link a thread from predecessor to root.

l->right = root;

l->isThreaded = true;

}

// If current node is rightmost child

if (root->right == NULL)

return root;

// Recur for right subtree.

return createThreaded(root->right);

}

// A utility function to find leftmost node in a binary tree rooted with 'root'.

// This function is used in inOrder()

Node \*leftMost(Node \*root)

{

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

root = root->left;

return root;

}

// Function to do inorder traversal of a threaded binary tree

void inOrder(Node \*root)

{

if (root == NULL) return;

// Find the leftmost node in Binary Tree

Node \*cur = leftMost(root);

while (cur != NULL)

{

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

// If this Node is a thread Node, then go to inorder successor

if (cur->isThreaded)

cur = cur->right;

else // Else go to the leftmost child in right subtree

cur = leftMost(cur->right);

}

}

// A utility function to create a new node

Node \*newNode(int key)

{

Node \*temp = new Node;

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

temp->key = key;

return temp;

}

// Driver program to test above functions

int main()

{

/\* 1

/ \

2 3

/ \ / \

4 5 6 7 \*/

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);

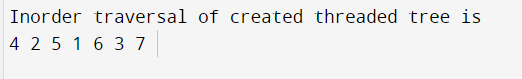
cout << "Inorder traversal of created threaded tree is\n";

inOrder(root);

return 0;

}

**Output:**



1. **Group C**

**Assignment 1**

**Aim: Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS.**

**Code:**

#include<iostream>

#include<stdlib.h>

using namespace std;

int cost[10][10],i,j,k,n,qu[10],front,rear,v,visit[10],visited[10];

int stk[10],top,visit1[10],visited1[10];

int main()

{

int m;

cout <<"enter no of vertices";

cin >> n;

cout <<"enter no of edges";

cin >> m;

cout <<"\nEDGES \n";

for(k=1;k<=m;k++)

{

cin >>i>>j;

cost[i][j]=1;

cost[j][i]=1;

}

//display function

cout<<"The adjacency matrix of the graph is:"<<endl;

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

{

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

{

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

}

cout<<endl;

}

cout <<"Enter initial vertex";

cin >>v;

cout <<"The BFS of the Graph is\n";

cout << v;

visited[v]=1;

k=1;

while(k<n)

{

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

if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)

{

visit[j]=1;

qu[rear++]=j;

}

v=qu[front++];

cout<<v << " ";

k++;

visit[v]=0;

visited[v]=1;

}

cout <<"Enter initial vertex";

cin >>v;

cout <<"The DFS of the Graph is\n";

cout << v;

visited[v]=1;

k=1;

while(k<n)

{

for(j=n;j>=1;j--)

if(cost[v][j]!=0 && visited1[j]!=1 && visit1[j]!=1)

{

visit1[j]=1;

stk[top]=j;

top++;

}

v=stk[--top];

cout<<v << " ";

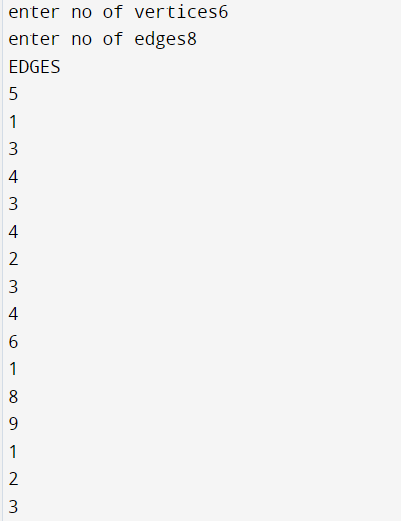
k++;

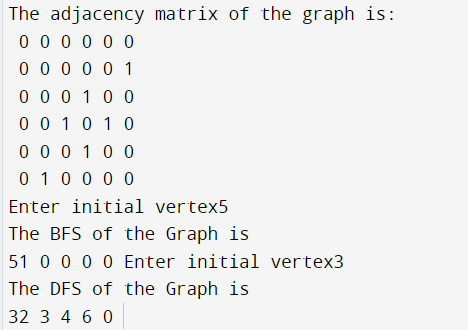
visit1[v]=0; visited1[v]=1;

}

}

**Output:**





1. **Group C**

**Assignment 2**

**Aim: You have a business with several offices; you want to lease phone lines to connect them up with each other and the phone company charge different amount of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structure.**

**Code:**

#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;

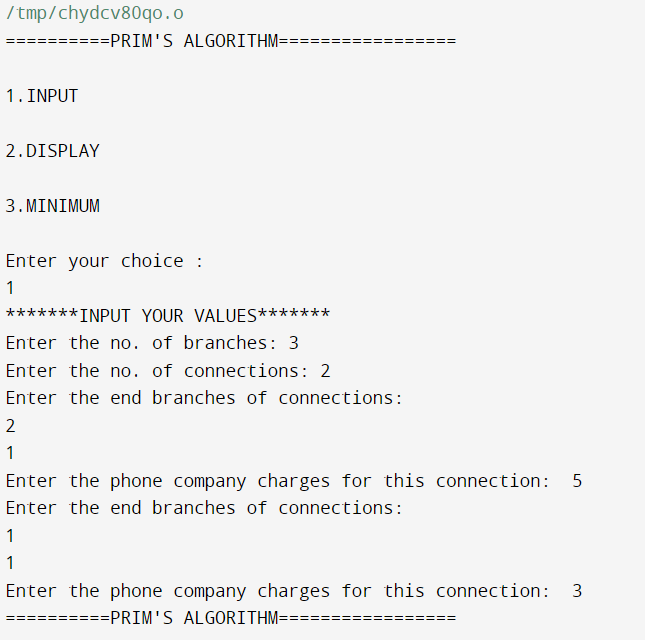
}

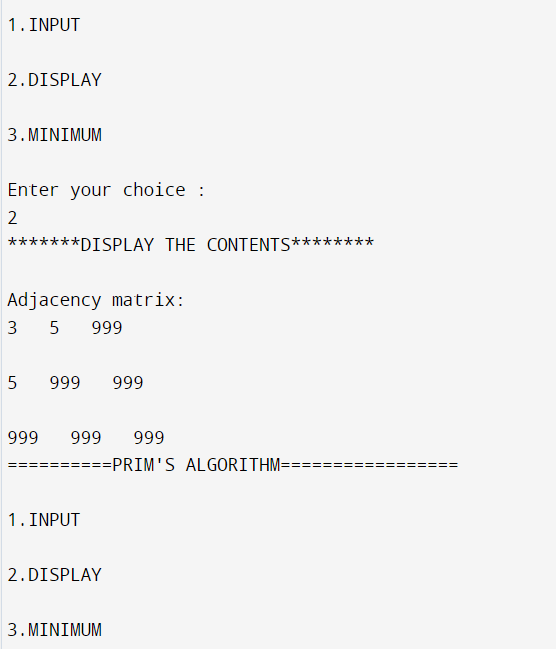
}while(ch!=4);

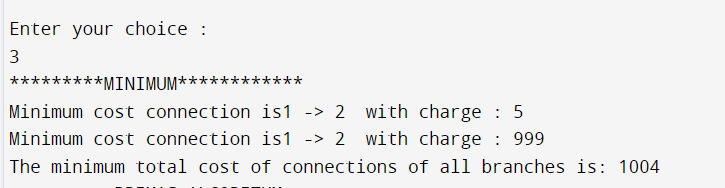
return 0;

}

**Output:**







1. **Group D**

**Assignment 1**

**Aim: Given sequence k=k1<…..<kn of n sorted keys, with a search probability pi for each key ki. Build the binary search tree that has the least search cost given the access probability for each key?**

**Code:**

// A naive recursive implementation of

// optimal binary search tree problem

#include <iostream>

# define INT\_MAX 50

using namespace std;

// A utility function to get sum of

// array elements freq[i] to freq[j]

int sum(int freq[], int i, int j);

// A recursive function to calculate

// cost of optimal binary search tree

int optCost(int freq[], int i, int j)

{

// Base cases

if (j < i) // no elements in this subarray

return 0;

if (j == i) // one element in this subarray

return freq[i];

// Get sum of freq[i], freq[i+1], ... freq[j]

int fsum = sum(freq, i, j);

// Initialize minimum value

int min = INT\_MAX;

// One by one consider all elements

// as root and recursively find cost

// of the BST, compare the cost with

// min and update min if needed

for (int r = i; r <= j; ++r)

{

int cost = optCost(freq, i, r - 1) +

optCost(freq, r + 1, j);

if (cost < min)

min = cost;

}

// Return minimum value

return min + fsum;

}

// The main function that calculates

// minimum cost of a Binary Search Tree.

// It mainly uses optCost() to find

// the optimal cost.

int optimalSearchTree(int keys[],

int freq[], int n)

{

// Here array keys[] is assumed to be

// sorted in increasing order. If keys[]

// is not sorted, then add code to sort

// keys, and rearrange freq[] accordingly.

return optCost(freq, 0, n - 1);

}

// A utility function to get sum of

// array elements freq[i] to freq[j]

int sum(int freq[], int i, int j)

{

int s = 0;

for (int k = i; k <= j; k++)

s += freq[k];

return s;

}

// Driver Code

int main()

{

int keys[] = {10, 12, 20};

int freq[] = {34, 8, 50};

int n = sizeof(keys) / sizeof(keys[0]);

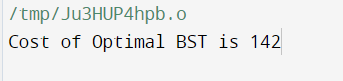
cout << "Cost of Optimal BST is "

<< optimalSearchTree(keys, freq, n);

return 0;

}

**Output:**



1. **Group D**

**Assignment 2**

**Aim: A dictionary stores keyword and its meaning. Provide facility for adding new keyword, updating values of any entry. Provide facility to display whole data sorted in ascending /descending order also find how many maximum comparisons may require for finding any keyword. Use height balance tree and find the complexity for finding a keyword.**

**Code:**

#include<iostream>

#include<string.h>

using namespace std;

class dict{

dict \*root,\*node,\*left,\*right,\*tree1;

string s1,s2;

int flag,flag1,flag2,flag3,cmp; public:

dict(){

flag=0,flag1=0,flag2=0,flag3=0,cmp=0;

root=NULL;

}

void input();

void create\_root(dict\*,dict\*);

void check\_same(dict\*,dict\*);

void input\_display();

void display(dict\*);

void input\_remove();

dict\* remove(dict\*,string);

dict\* findmin(dict\*);

void input\_find();

dict\* find(dict\*,string);

void input\_update();

dict\* update(dict\*,string);

};

void dict::input(){

node=new dict;

cout<<"\nEnter the keyword:\n";

cin>>node->s1;

cout<<"Enter the meaning of the keyword:\n";

cin.ignore();

getline(cin,node->s2);

create\_root(root,node);}

void dict::create\_root(dict \*tree,dict \*node1){

int i=0,result;

char a[20],b[20];

if(root==NULL){

root=new dict;

root=node1;

root->left=NULL;

root->right=NULL;

cout<<"\nRoot node created successfully"<<endl;

return; }

for(i=0;node1->s1[i]!='\0';i++){

a[i]=node1->s1[i];

}

for(i=0;tree->s1[i]!='\0';i++){

b[i]=tree->s1[i];}

result=strcmp(b,a);

check\_same(tree,node1);

if(flag==1){

cout<<"The word you entered already exists.\n";

flag=0;

}

else{

if(result>0){

if(tree->left!=NULL){

create\_root(tree->left,node1);}

else{

tree->left=node1;

(tree->left)->left=NULL;

(tree->left)->right=NULL;

cout<<"Node added to left of "<<tree->s1<<"\n";

return; }

}

else if(result<0){

if(tree->right!=NULL){

create\_root(tree->right,node1);

}

else{

tree->right=node1;

(tree->right)->left=NULL;

(tree->right)->right=NULL;

cout<<"Node added to right of "<<tree->s1<<"\n";

return;}

}}}

void dict::check\_same(dict \*tree,dict \*node1){

if(tree->s1==node1->s1){

flag=1;

return;

}

else if(tree->s1>node1->s1){

if(tree->left!=NULL){

check\_same(tree->left,node1);

}

}

else if(tree->s1<node1->s1){

if(tree->right!=NULL){

check\_same(tree->right,node1);

}}

}

void dict::input\_display(){

if(root!=NULL){

cout<<"The words entered in the dictionary are:\n\n";

display(root);

}

else{

cout<<"\nThere are no words in the dictionary.\n";

}

}

void dict::display(dict \*tree){

if(tree->left==NULL&&tree->right==NULL){

cout<<tree->s1<<" = "<<tree->s2<<"\n\n";}

else{

if(tree->left!=NULL){

display(tree->left);}

cout<<tree->s1<<" = "<<tree->s2<<"\n\n";

if(tree->right!=NULL){

display(tree->right);

}}}

void dict::input\_remove(){

char t;

if(root!=NULL){

cout<<"\nEnter a keyword to be deleted:\n";

cin>>s1;

remove(root,s1);

if(flag1==0){

cout<<"\nThe word '"<<s1<<"' has been deleted.\n";

}

flag1=0;

}

else{

cout<<"\nThere are no words in the dictionary.\n";

}}

dict\* dict::remove(dict \*tree,string s3){

dict \*temp;

if(tree==NULL){

cout<<"\nWord not found.\n";

flag1=1;

return tree;

}

else if(tree->s1>s3){

tree->left=remove(tree->left,s3);

return tree;

}

else if(tree->s1<s3){

tree->right=remove(tree->right,s3);

return tree;

}

else{

if(tree->left==NULL&&tree->right==NULL){

delete tree;

tree=NULL;

}

else if(tree->left==NULL){

temp=tree;

tree=tree->right;

delete temp;

}

else if(tree->right==NULL){

temp=tree;

tree=tree->left;

delete temp;

}

else{

temp=findmin(tree->right);

tree=temp;

tree->right=remove(tree->right,temp->s1);

}}

return tree;

}

dict\* dict::findmin(dict \*tree){

while(tree->left!=NULL){

tree=tree->left;

}

return tree;

}

void dict::input\_find(){

flag2=0,cmp=0;

if(root!=NULL){

cout<<"\nEnter the keyword to be searched:\n";

cin>>s1;

find(root,s1);

if(flag2==0){

cout<<"Number of comparisons needed: "<<cmp<<"\n";

cmp=0;}}

else{

cout<<"\nThere are no words in the dictionary.\n";}}

dict\* dict::find(dict \*tree,string s3){

if(tree==NULL){

cout<<"\nWord not found.\n";

flag2=1;

flag3=1;

cmp=0;}

else{

if(tree->s1==s3){

cmp++;

cout<<"\nWord found.\n";

cout<<tree->s1<<": "<<tree->s2<<"\n";

tree1=tree;

return tree;}

else if(tree->s1>s3){

cmp++;

find(tree->left,s3);}

else if(tree->s1<s3){

cmp++;

find(tree->right,s3);

}

}

return tree;

}

void dict::input\_update(){

if(root!=NULL){

cout<<"\nEnter the keyword to be updated:\n";

cin>>s1;

update(root,s1);

}

else{

cout<<"\nThere are no words in the dictionary.\n";

} }

dict\* dict::update(dict \*tree,string s3){

flag3=0;

find(tree,s3);

if(flag3==0){

cout<<"\nEnter the updated meaning of the keyword:\n";

cin.ignore();

getline(cin,tree1->s2);

cout<<"\nThe meaning of '"<<s3<<"' has been updated.\n";

}

return tree;

}

int main(){

int ch;

dict d;

do{

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

"\n\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:\n"

"\nEnter your choice:\n"

"1.Add new keyword.\n"

"2.Display the contents of the Dictionary.\n"

"3.Delete a keyword.\n"

"4.Find a keyword.\n"

"5.Update the meaning of a keyword.\n"

"6.Exit.\n"

"===============================================\n";

cin>>ch;

switch(ch){

case 1:d.input();

break;

case 2:d.input\_display();

break;

case 3:d.input\_remove();

break;

case 4:d.input\_find();

break;

case 5:d.input\_update();

break;

default:cout<<"\nPlease enter a valid option!\n";

break;

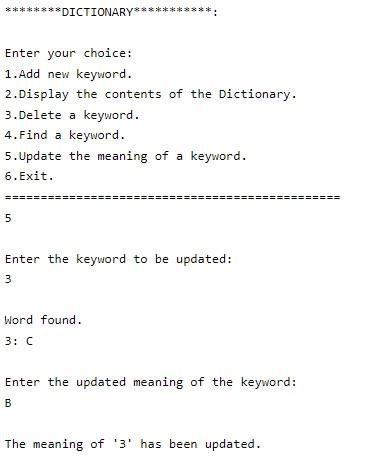
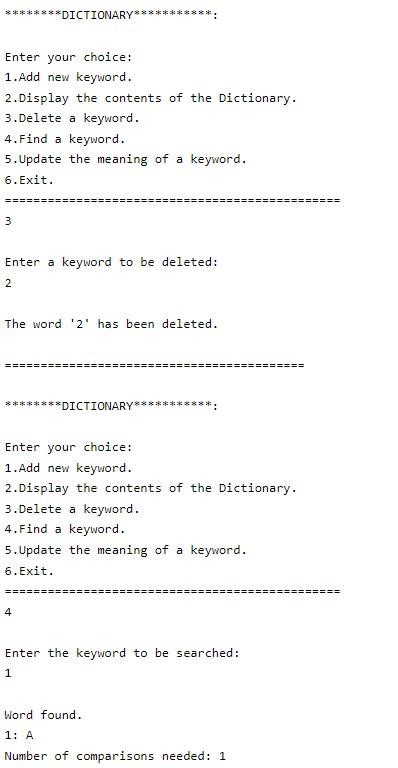
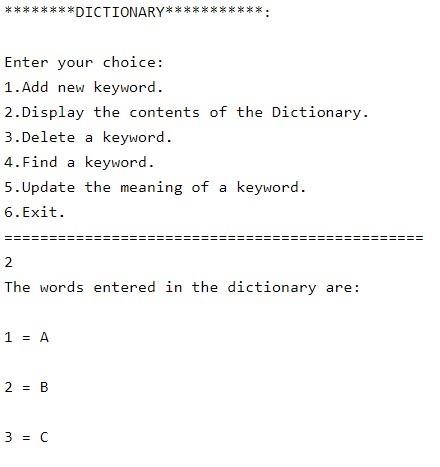
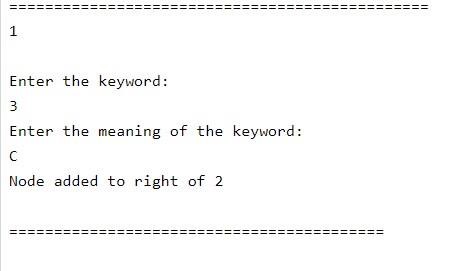
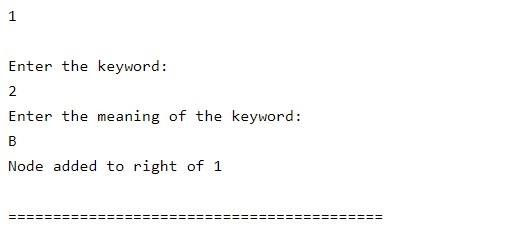
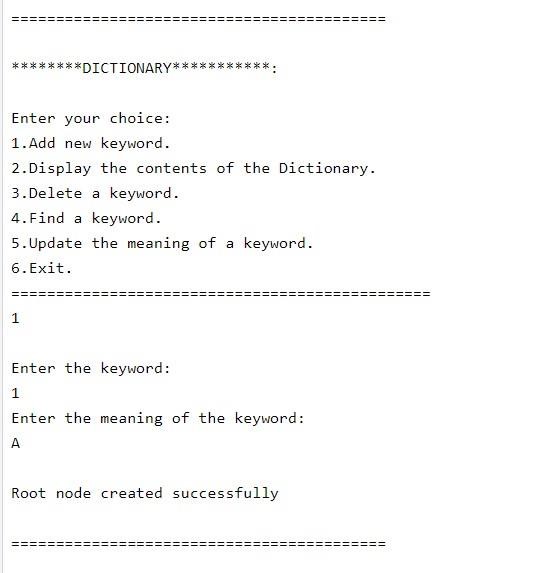
}

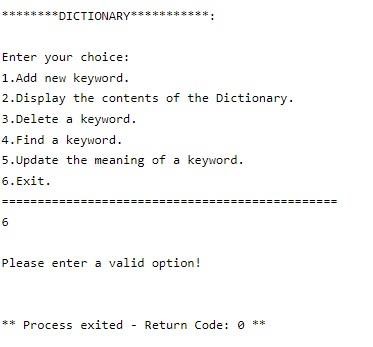
}while(ch!=6);

return 0;

}

**Output:**





1. **Group E**

**Assignment 1**

**Aim: Consider a scenario for hospital to cater service to different kinds of patients as a) serious (top priority) b) non-serious (Medium priority) c) General check up (least priority). Implement the priority queue to cater services to the patients.**

**Code:**

#include<iostream>

#include<stdlib.h>

#include<string.h>

#include<iomanip>

using namespace std;

// Patient Class

class Patient

{

int pr;

char name[20];

Patient\* next;

Patient(char n[], int p)

{

pr=p;

strcpy(name,n);

next=NULL;

}

// Display Patient info

void showPatient()

{

cout<<setw(20)<<name<<setw(20);

if(pr==1)

cout<<"General Chech-Up\n";

else if(pr==2)

cout<<"Half Serious\n";

else

cout<<"Serious\n";

}

friend class DB;

};

// DataBase Class (Patient record)

class DB

{

Patient\* head;

public:

DB():head(NULL){}

// Insert patient according to priority

void insert(char n[], int pat)

{

Patient\* p=new Patient(n, pat);

if(!head)

head=p;

else if(p->pr > head->pr)

{

p->next=head;

head=p;

}

else

{

Patient\* q;

for(q=head ; q->next!=NULL && q->next->pr >= p->pr ; q=q->next);

if(!q->next)

q->next=p;

else

{

p->next=q->next;

q->next=p;

}

}

}

// Provide service to first patient inside DB

void svc()

{

if(!head)

cout<<"Currently no Patients in DB\n";

else

{

cout<<"\nService is Provided to\n";

cout<<setw(20)<<"Name"<<setw(20)<<"Priority\n"; head->showPatient();

head=head->next;

}

}

// Display Database

void showDB()

{

if(!head)

cout<<"\nCurrently no patients in DB\n";

else

{

cout<<"\n\n\t\t--> Patient DataBase <--\n\n"; cout<<setw(20)<<"Name"<<setw(20)<<"Priority\n"; for(Patient\* q=head ; q ; q=q->next)

q->showPatient();

}

}

};

int main()

{

//clrscr();

int ch, pr;

char name[20];

DB db;

do{

cout<<"\n --> MENU <--\n";

cout<<"\n1) Add new patient\n2) Provide Service\n3) Display Patient List\n4) Quit\n";

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

cin>>ch;

switch(ch)

{

case 1:

cout<<"\nEnter name of Patient :: ";

cin>>name;

cout<<"\nEnter '3' for serious\nEnter '2' for Half-serious\nEnter '1' for General Check-Up\n";

cout<<"\nEnter priority --> ";

cin>>pr;

db.insert(name, pr);

break;

case 2:

db.svc();

break;

case 3:

db.showDB();

break;

case 4:

exit(1);

default:

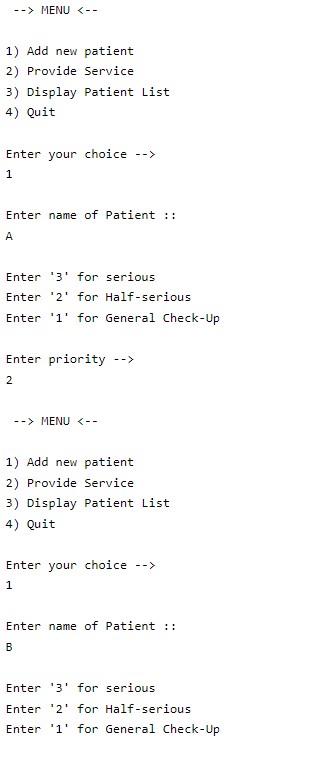
cout<<"\nInvalid Choice ....";

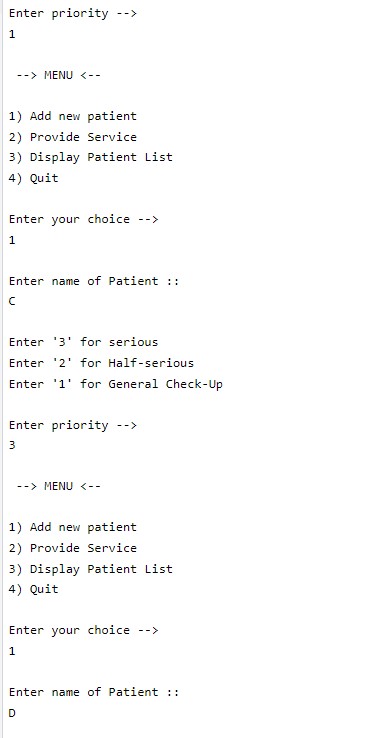
}

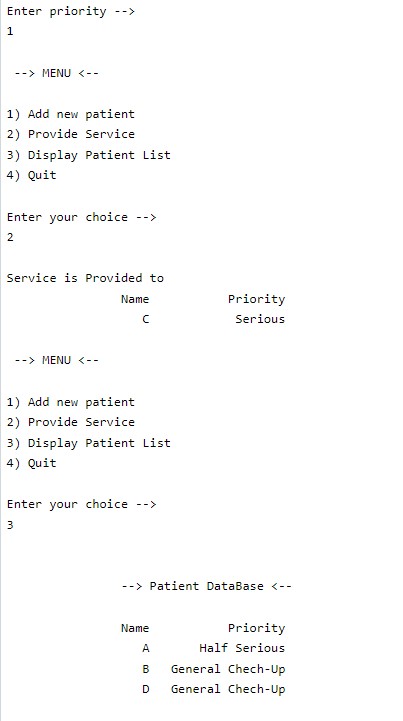
}while(1);

}

**Output:**







1. **Group F**

**Assignment 1**

**Aim: Department maintains a student information. The file contains roll number, name and division and address. All user to add, delete information of student. Display information of particular employee. If record of students does not exists an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to maintain the data.**

**Code:**

**#include<iostream>**

**#include<fstream>**

**#include<cstring>**

**using namespace std;**

**class Student{**

**int marks;**

**char name[100],**

**subject[100],**

**code[100],**

**rollno[100];**

**public:**

**void accept(){**

**cout<<"Enter roll number : ";**

**cin >> rollno;**

**cout<<"Enter name : ";**

**cin.ignore();**

**cin.getline(name,100);**

**cout<<"Enter subject code : ";**

**cin >> code;**

**cout<<"Enter subject name : ";**

**cin.ignore();**

**cin.getline(subject,100);**

**cout<<"Enter marks : ";**

**cin >> marks;**

**}**

**void output(){**

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

**cout<<"Roll Number : "<<rollno<<endl;**

**cout<<"Name : "<<name<<endl;**

**cout<<"Subject Code : "<<code<<endl;**

**cout<<"Subject Name : "<<subject<<endl;**

**cout<<"Marks : "<<marks<<endl;**

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

**}**

**char\* get\_roll(){**

**return rollno;**

**}**

**};**

**int main(){**

**int ch;**

**char roll[100];**

**int c = 0;**

**Student s;**

**do{**

**cout<<"================================================================="<<endl;**

**cout<<"Press 1 to insert a new record : "<<endl;**

**cout<<"Press 2 to display a particular record : "<<endl;**

**cout<<"Press 3 to delete a record : "<<endl;**

**cout<<"Press 0 to Exit : "<<endl;**

**cout<<"================================================================="<<endl;**

**cin >> ch;**

**switch(ch){**

**case 1:{**

**ofstream f;**

**f.open("Student.dat",ios::app);**

**s.accept();**

**f.write((char\*)&s,sizeof(s));**

**c++;**

**f.close();**

**}**

**break;**

**case 2:{**

**cout<<"Enter the Roll Number : ";**

**cin >> roll;**

**bool flag = 0;**

**ifstream f;**

**f.open("Student.dat");**

**for(int i = 0;i<c;i++){**

**f.read((char\*)&s,sizeof(s));**

**if(strcmp(s.get\_roll(),roll) == 0)**

**{**

**cout<<"Record found with following details : "<<endl;**

**s.output();**

**flag = 1;**

**break;**

**}**

**cout<<endl;**

**}**

**if(flag == 0)**

**cout<<"No such record found."<<endl;**

**f.close();**

**break;**

**}**

**case 3:{**

**cout<<"Enter the Roll Number to be deleted : ";**

**cin >> roll;**

**ofstream fout;**

**ifstream fin;**

**bool flag = false;**

**fout.open("temp.dat",ios::app);**

**fin.open("Student.dat");**

**for(int i = 0;i<c;i++){**

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

**if(strcmp(s.get\_roll(),roll) == 0){**

**flag = true;**

**continue;**

**}**

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

**}**

**if(!flag){**

**cout<<"No such record found."<<endl;**

**break;**

**}**

**else**

**cout<<"Record is deleted successfully."<<endl;**

**fin.close();**

**fout.close();**

**remove("Student.dat");**

**rename("temp.dat","Student.dat");**

**break;**

**}**

**case 0:**

**cout<<"Exited"<<endl;**

**break;**

**default:**

**cout<<"Invalid Choice !!! Please try again."<<endl;**

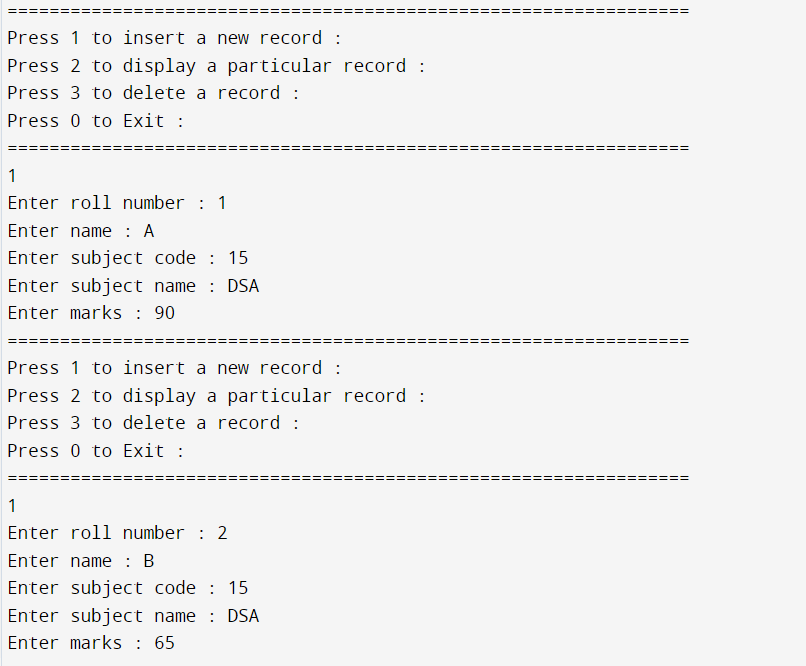
**}**

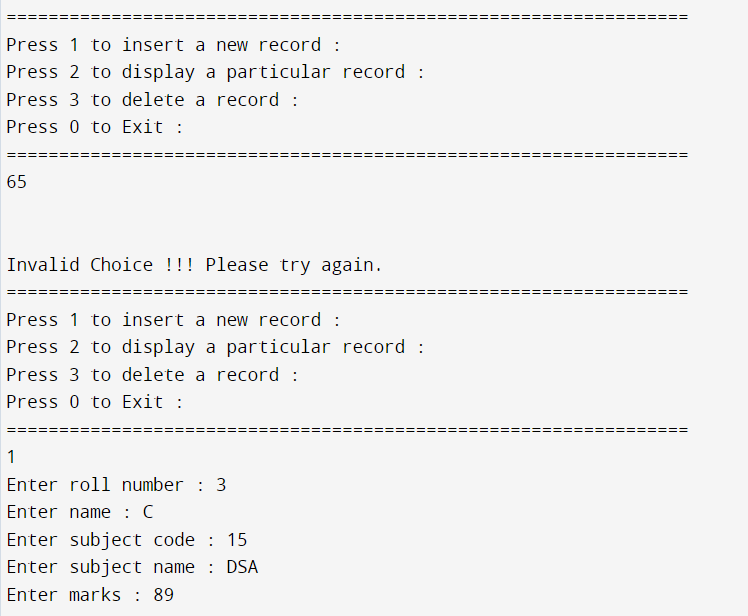
**}while(ch != 0);**

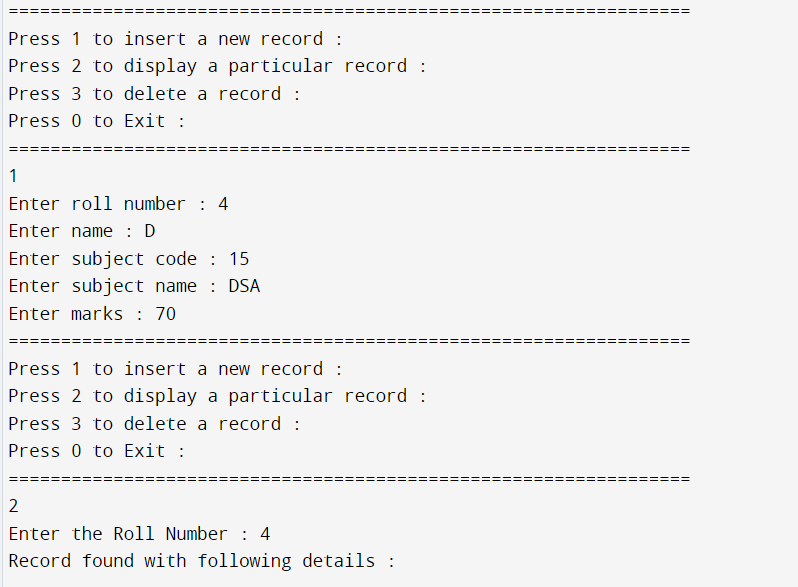
**return 0;**

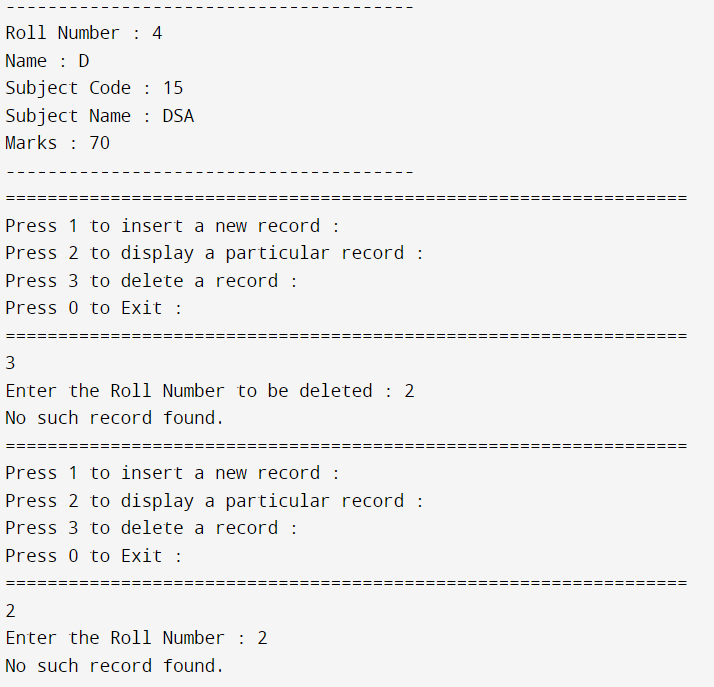
**}**

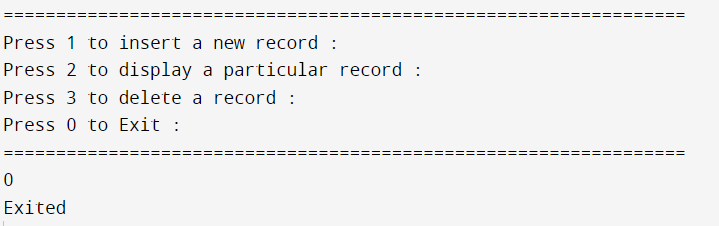
**Output:**

****

****

****

****

****

1. **Group F**

**Assignment 2**

**Aim: Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exists an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.**

**Code:**

#include<iostream>

#include<fstream>

#include<cstring>

using namespace std;

class Employee

{

int salary;

char name[100],desig[100],empid[100];

public:

void accept()

{

cout<<"Enter Employee ID : ";

cin >> empid;

cout<<"Enter name : ";

cin.ignore();

cin.getline(name,100);

cout<<"Enter Designation : ";

cin.getline(desig,100);

cout<<"Enter salary : ";

cin >> salary;

}

void output()

{

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

cout<<"Employee ID : "<<empid<<endl;

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

cout<<"Designation : "<<desig<<endl;

cout<<"Salary : "<<salary<<endl;

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

}

char\* get\_id()

{

return empid;

}

};

int main()

{

int ch;

char id[100];

int c = 0;

Employee s;

do

{

cout<<"================================================================="<<endl;

cout<<"Press 1 to insert a new record : "<<endl;

cout<<"Press 2 to display a particular record : "<<endl;

cout<<"Press 3 to delete a record : "<<endl;

cout<<"Press 0 to Exit : "<<endl;

cout<<"================================================================="<<endl;

cin >> ch;

switch(ch)

{

case 1:

{

ofstream f;

f.open("Employee.dat",ios::app);

s.accept();

f.write((char\*)&s,sizeof(s));

c++;

f.close();

}

break;

case 2:

{

cout<<"Enter the Employee ID : ";

cin >> id;

bool flag = 0;

ifstream f;

f.open("Employee.dat");

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

{

f.read((char\*)&s,sizeof(s));

if(strcmp(s.get\_id(),id) == 0)

{

cout<<"Record found with following details : "<<endl;

s.output();

flag = 1;

break;

}

cout<<endl;

}

if(flag == 0)

cout<<"No such record found."<<endl;

f.close();

break;

}

case 3:

{

cout<<"Enter the Employee ID to be deleted : ";

cin >> id;

ofstream fout;

ifstream fin;

bool flag = false;

fout.open("temp.dat",ios::app);

fin.open("Employee.dat");

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

{

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

if(strcmp(s.get\_id(),id) == 0)

{

flag = true;

continue;

}

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

}

if(!flag)

{

cout<<"No such record found."<<endl;

break;

}

else

cout<<"Record is deleted successfully."<<endl;

fin.close();

fout.close();

remove("Employee.dat");

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

break;

}

case 0:

cout<<"Exited"<<endl;

break;

default:

cout<<"Invalid Choice !!! Please try again."<<endl;

}

}while(ch != 0);

return 0;

}

**Output:**

