#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 name for city no." << i+1 <<" : ";

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);

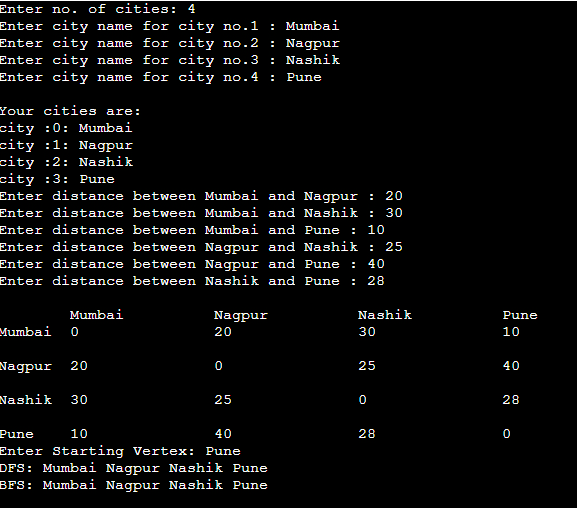
cout << endl;

cout << "BFS: ";

bfs(u, n, cities);

return 0;

}



#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 number of vertices : ";

cin >> n;

cout << "Enter number 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<<endl;

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 <<endl<<"Enter initial vertex : ";

cin >> v;

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

cout << v<<endl;

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++;

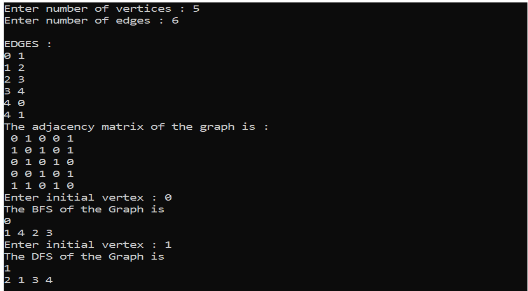
visit1[v] = 0;

visited1[v] = 1;

}

return 0;

}



#include <iostream>

#include <cstring>

using namespace std;

struct hash

{

string word;

string meaning;

int chain;

} obj[10];

void hash\_initialization()

{

for (int i = 0; i < 10; i++)

{

obj[i].word = "-";

obj[i].meaning = "-";

obj[i].chain = -1;

}

}

void display()

{

for (int i = 0; i < 10; i++)

{

cout << obj[i].word << "-->" << obj[i].meaning << "-->" << obj[i].chain << endl;

}

}

int calculate(string word)

{

int key = 0;

for (int i = 0; i < word.length(); i++)

{

key = key + word[i];

}

return key % 10;

}

void collision(int key, string word, string meaning)

{

int i = 1;

while (((key + i) % 10) < 10)

{

if (obj[(key + i) % 10].word == "-")

{

obj[(key + i) % 10].word = word;

obj[(key + i) % 10].meaning = meaning;

obj[(key + i - 1) % 10].chain = (key + i) % 10;

break;

}

else

{

i++;

}

}

}

void insert()

{

string wd, mg;

cout << "Enter the word => ";

cin >> wd;

cout << "Enter the meaning => ";

cin >> mg;

int hash\_key = calculate(wd);

if (obj[hash\_key].word == "-")

{

obj[hash\_key].word = wd;

obj[hash\_key].meaning = mg;

}

else

{

collision(hash\_key, wd, mg);

}

}

void find(string wd)

{

int hash\_key = calculate(wd);

if (obj[hash\_key].word == wd)

{

cout << "found" << endl;

cout << obj[hash\_key].word << "-->" << obj[hash\_key].meaning << endl;

}

else if (obj[hash\_key].chain != -1)

{

int temp = obj[hash\_key].chain;

while (true)

{

if (obj[temp].word == wd)

{

cout << "found" << endl;

cout << obj[temp].word << "-->" << obj[temp].meaning << endl;

break;

}

temp = obj[temp].chain;

}

}

else

{

cout << "Not Found" << endl;

}

}

void Del(string wd)

{

int hash\_key = calculate(wd);

if (obj[hash\_key].word == wd)

{

obj[hash\_key].word = "-";

obj[hash\_key].meaning = "-";

obj[hash\_key].chain = -1;

}

else if (obj[hash\_key].chain != -1)

{

int temp = obj[hash\_key].chain;

while (true)

{

if (obj[temp].word == wd)

{

obj[temp].word = "-";

obj[temp].meaning = "-";

obj[temp].chain = -1;

break;

}

temp = obj[temp].chain;

}

}

else

{

cout << "Word Not Found" << endl;

}

}

int main()

{

int choice, n;

string wd\_find, wd\_Del;

hash\_initialization();

do

{

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

cout << "1) Insert" << endl;

cout << "2) Find" << endl;

cout << "3) Delete" << endl;

cout << "4) Print" << endl;

cout << "5) Exit" << endl;

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter how entries you want to make ";

cin >> n;

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

{

insert();

}

break;

case 2:

cout << "Enter the word to found => ";

cin >> wd\_find;

find(wd\_find);

break;

case 3:

cout << "Enter the word to be deleted =>";

cin >> wd\_Del;

Del(wd\_Del);

break;

case 4:

display();

break;

case 5:

break;

default:

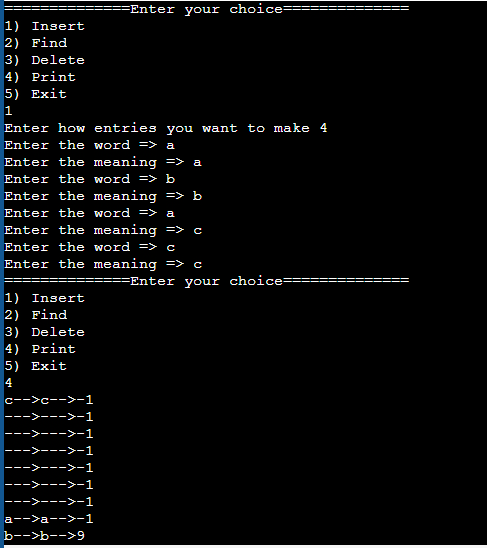
cout << "Invalid choice" << endl;

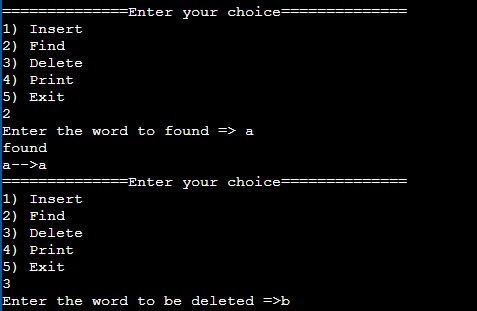
break;

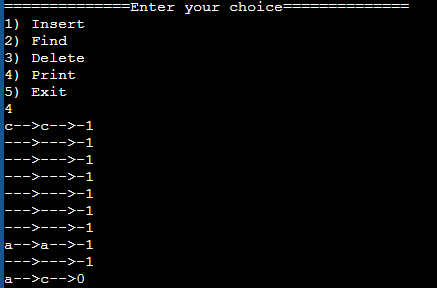
}

} while (choice < 5);

return 0;}}







import java.io.\*;

import java.util.\*;

public class heapsort {

public int[] heap;

public int count;

public void downadjust(int i) {

int j, temp, n;

n = heap[0];

if (2 \* i <= n) {

j = 2 \* i;// j on left child of i

if (j + 1 <= n && heap[j + 1] > heap[j]) // j points to larger of left and right child

j = j + 1;

if (heap[i] < heap[j]) {

temp = heap[i];

heap[i] = heap[j];

heap[j] = temp;

downadjust(j);

}

}

}

public void upadjust(int i) {

int temp;

while (i > 1 && heap[i] > heap[i / 2]) {

temp = heap[i];

heap[i] = heap[i / 2];

heap[i / 2] = temp;

i = i / 2;

}

}

public void insert(int x) {

heap[++heap[0]] = x;

upadjust(heap[0]);

}

public void create() {

int i, x, n;

heap = new int[30];

heap[0] = 0;

Scanner reader = new Scanner(System.in);

System.out.println("\nEnter No. of elements : ");

n = reader.nextInt();

count = n;

System.out.println("\nEnter heap data : ");

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

x = reader.nextInt();

insert(x);

}

}

public void sort() {

int last, temp;

while (heap[0] > 1) {

last = heap[0];

temp = heap[1];

heap[1] = heap[last];

heap[last] = temp;

heap[0]--;

downadjust(1);

}

}

public void print() {

int n, i;

n = count;

System.out.println("\nsorted data : ");

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

System.out.print(" " + heap[i]);

}

public static void main(String[] args) {

int x;

heapsort myobject = new heapsort();

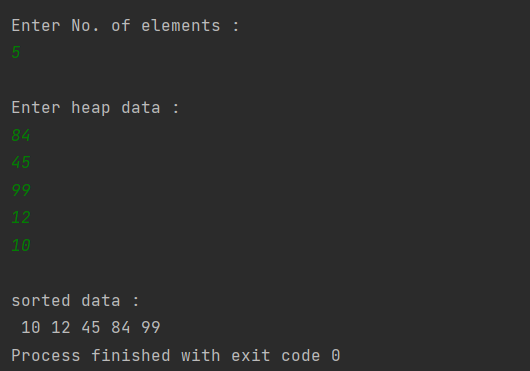
myobject.create();

myobject.sort();

myobject.print();

}

}



#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);

};

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<<"\nHow many Word to insert?:\n";

cin>>n;

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

{

cout<<"\nENter Word: ";

cin>>wordI;

cout<<"\nEnter 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<<"\nWord 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;

}

}

int main() {

string word;

dictionary months;

months.create();

cout<<"Ascending order\n";

months.inorder();

cout<<"\nDescending order:\n";

months.postorder();

cout<<"\nEnter word to search: ";

cin>>word;

int comparisons=months.search(word);

if(comparisons==-1)

{

cout<<"\nNot found word";

}

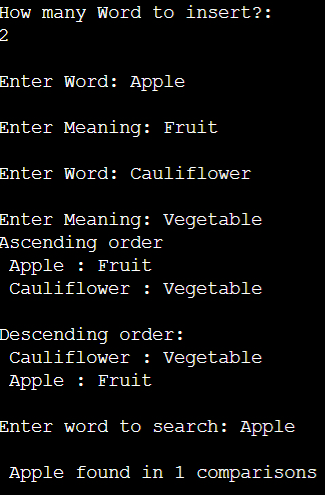
else

{

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

}

return 0;}



#include <iostream>

using namespace std;

#define SIZE 10

class OBST

{

int p[SIZE]; // Probabilities with which we search for an element

int q[SIZE]; // Probabilities that an element is not found

int a[SIZE]; // Elements from which OBST is to be built

int w[SIZE][SIZE]; // Weight ‘w[i][j]’ of a tree having root

//’r[i][j]’

int c[SIZE][SIZE]; // Cost ‘c[i][j] of a tree having root ‘r[i][j]

int r[SIZE][SIZE]; // represents root

int n; // number of nodes

public:

/\* This function accepts the input data \*/

void get\_data()

{

int i;

cout << "\n Optimal Binary Search Tree \n";

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

cin >> n;

cout << "\n Enter the data as…\n";

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

{

cout << "\n a[" << i << "]";

cin >> a[i];

}

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

{

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

cin >> p[i];

}

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

{

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

cin >> q[i];

}

}

/\* This function returns a value in the range ‘r[i][j-1]’ to ‘r[i+1][j]’so

that the cost ‘c[i][k-1]+c[k][j]’is minimum \*/

int Min\_Value(int i, int j)

{

int m, k;

int minimum = 32000;

for (m = r[i][j - 1]; m <= r[i + 1][j]; m++)

{

if ((c[i][m - 1] + c[m][j]) < minimum)

{

minimum = c[i][m - 1] + c[m][j];

k = m;

}

}

return k;

}

/\* This function builds the table from all the given probabilities It

basically computes C,r,W values \*/

void build\_OBST()

{

int i, j, k, l, m;

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

{

// initialize

w[i][i] = q[i];

r[i][i] = c[i][i] = 0;

// Optimal trees with one node

w[i][i + 1] = q[i] + q[i + 1] + p[i + 1];

r[i][i + 1] = i + 1;

c[i][i + 1] = q[i] + q[i + 1] + p[i + 1];

}

w[n][n] = q[n];

r[n][n] = c[n][n] = 0;

// Find optimal trees with ‘m’ nodes

for (m = 2; m <= n; m++)

{

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

{

j = i + m;

w[i][j] = w[i][j - 1] + p[j] + q[j];

k = Min\_Value(i, j);

c[i][j] = w[i][j] + c[i][k - 1] + c[k][j];

r[i][j] = k;

}

}

}

/\* This function builds the tree from the tables made by the OBST function \*/

void build\_tree()

{

int i, j, k;

int queue[20], front = -1, rear = -1;

cout << "The Optimal Binary Search Tree For the Given Node Is…\n";

cout << "\n The Root of this OBST is ::" << r[0][n];

cout << "\nThe Cost of this OBST is::" << c[0][n];

cout << "\n\n\t NODE \t LEFT CHILD \t RIGHT CHILD ";

cout << "\n";

queue[++rear] = 0;

queue[++rear] = n;

while (front != rear)

{

i = queue[++front];

j = queue[++front];

k = r[i][j];

cout << "\n\t" << k;

if (r[i][k - 1] != 0)

{

cout << "\t\t" << r[i][k - 1];

queue[++rear] = i;

queue[++rear] = k - 1;

}

else

cout << "\t\t";

if (r[k][j] != 0)

{

cout << "\t" << r[k][j];

queue[++rear] = k;

queue[++rear] = j;

}

else

cout < "\t";

} // end of while

cout << "\n";

}

}; // end of the class

/\*This is the main function \*/

int main()

{

OBST obj;

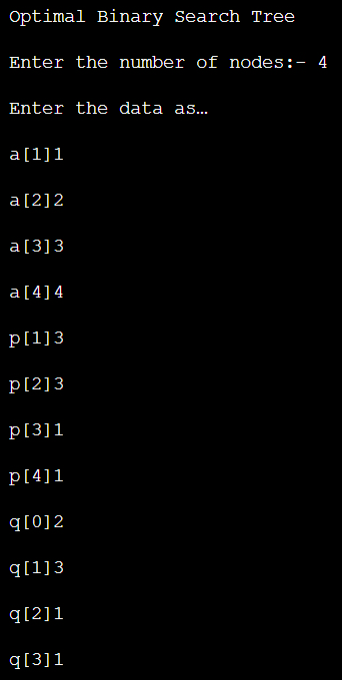
obj.get\_data();

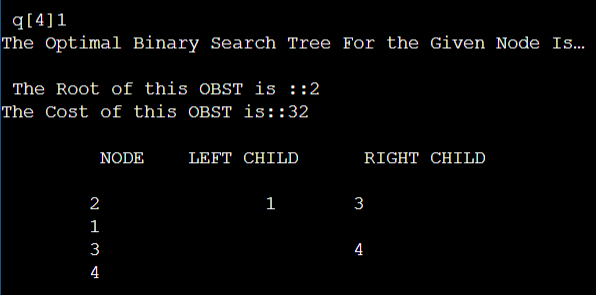
obj.build\_OBST();

obj.build\_tree();

return 0;

}





#include<iostream>

#include<fstream>

#include<stdio.h>

using namespace std;

class Employee{

private:

int code;

char name[20];

float salary;

public:

void read();

void display();

int getEmpCode() { return code;}

int getSalary() { return salary;}

void updateSalary(float s) { salary=s;}

};

void Employee::read(){

cout<<"Enter employee code: ";

cin>>code;

cout<<"Enter name: ";

cin.ignore(1);

cin.getline(name,20);

cout<<"Enter salary: ";

cin>>salary;

}

void Employee::display()

{

cout<<code<<" "<<name<<"\t"<<salary<<endl;

}

fstream file;

void deleteExistingFile(){

remove("EMPLOYEE.DAT");

}

void appendToFille(){

Employee x;

x.read();

file.open("EMPLOYEE.DAT",ios::binary|ios::app);

if(!file){

cout<<"ERROR IN CREATING FILE\n";

return;

}

file.write((char\*)&x,sizeof(x));

file.close();

cout<<"Record added sucessfully.\n";

}

void displayAll(){

Employee x;

file.open("EMPLOYEE.DAT",ios::binary|ios::in);

if(!file){

cout<<"ERROR IN OPENING FILE \n";

return;

}

while(file){

if(file.read((char\*)&x,sizeof(x)))

if(x.getSalary()>=10000 && x.getSalary()<=20000)

x.display();

}

file.close();

}

void searchForRecord(){

//read employee id

Employee x;

int c;

int isFound=0;

cout<<"Enter employee code: ";

cin>>c;

file.open("EMPLOYEE.DAT",ios::binary|ios::in);

if(!file){

cout<<"ERROR IN OPENING FILE \n";

return;

}

while(file){

if(file.read((char\*)&x,sizeof(x))){

if(x.getEmpCode()==c){

cout<<"RECORD FOUND\n";

x.display();

isFound=1;

break;

}

}

}

if(isFound==0){

cout<<"Record not found!!!\n";

}

file.close();

}

void increaseSalary(){

//read employee id

Employee x;

int c;

int isFound=0;

float sal;

cout<<"enter employee code \n";

cin>>c;

file.open("EMPLOYEE.DAT",ios::binary|ios::in);

if(!file){

cout<<"ERROR IN OPENING FILE \n";

return;

}

while(file){

if(file.read((char\*)&x,sizeof(x))){

if(x.getEmpCode()==c){

cout<<"Salary hike? ";

cin>>sal;

x.updateSalary(x.getSalary()+sal);

isFound=1;

break;

}

}

}

if(isFound==0){

cout<<"Record not found!!!\n";

}

file.close();

cout<<"Salary updated successfully."<<endl;

}

void insertRecord(){

//read employee record

Employee x;

Employee newEmp;

newEmp.read();

fstream fin;

file.open("EMPLOYEE.DAT",ios::binary|ios::in);

//open file in write mode

fin.open("TEMP.DAT",ios::binary|ios::out);

if(!file){

cout<<"Error in opening EMPLOYEE.DAT file!!!\n";

return;

}

if(!fin){

cout<<"Error in opening TEMP.DAT file!!!\n";

return;

}

while(file){

if(file.read((char\*)&x,sizeof(x))){

if(x.getEmpCode()>newEmp.getEmpCode()){

fin.write((char\*)&newEmp, sizeof(newEmp));

}

//no need to use else

fin.write((char\*)&x, sizeof(x));

}

}

fin.close();

file.close();

rename("TEMP.DAT","EMPLOYEE.DAT");

remove("TEMP.DAT");

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

}

int main()

{

char ch;

//if required then only remove the file

deleteExistingFile();

do{

int n;

cout<<"ENTER CHOICE\n"<<"1.ADD AN EMPLOYEE\n"<<"2.DISPLAY\n"<<"3.SEARCH\n"<<"4.INCREASE SALARY\n"<<"5.INSERT RECORD\n";

cout<<"Make a choice: ";

cin>>n;

switch(n){

case 1:

appendToFille();

break;

case 2 :

displayAll();

break;

case 3:

searchForRecord();

break;

case 4:

increaseSalary();

break;

case 5:

insertRecord();

break;

default :

cout<<"Invalid Choice\n";

}

cout<<"Do you want to continue ? : ";

cin>>ch;

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

return 0;}

