#include<bits/stdc++.h>

#include <time.h>

using namespace std;

#define INF 99999999

//this function return the distinct zipcode

int\* printDistinct(int arr[], int n)

{

int \*b=new int[5],c=0;

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

{ //outer loop picks the element from left

int j;

for (j=0; j<i; j++) //inner loop check if the element is present on left side of it

if (arr[i] == arr[j]) //if present ,then skip it

break;

if (i == j) //else store it

b[c++]=arr[i];

}

return b;

}

void swap(int\* xp, int\* yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void bubbleSort(int distance[], int distinct\_zipcode[], int n)

{

int i, j;

for (i = 0; i < n - 1; i++)

for (j = 0; j < n - i - 1; j++)

if (distance[j] > distance[j + 1])

{

swap(&distance[j], &distance[j + 1]);

swap(&distinct\_zipcode[j], &distinct\_zipcode[j + 1]);

}

}

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

{

int min=INT\_MAX,min\_index;

for(int i=0;i<5;i++)

if(fin[i]==false && dist[i]<=min)

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

return min\_index;

}

int\* printSolution(int dist[], int \*distinct)

{

int\*arr=new int[5];

cout <<"\tZip Code \tDistance from Source" << endl;

for (int i = 0; i < 5; i++)

{

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

arr[i]=dist[i];

}

return arr;

}

//return the shortest distance between the source node i.e from user input present location to neighbour location

int\* dijkstra(int graph[5][5],int src, int \*distinct)

{

int dist[5]; //Output array. dist[i] hold the shortest dist. from source to i

bool fin[5]; //True, if shortest dist. is finialized

int \*a=new int[5];

for (int i = 0; i < 5; i++) //Initalizing dist as INFINITE and fin as FALSE

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

dist[src]=0; //distance source from itself is 0;

for(int count=0;count<4;count++)

{

int u=minDistance(dist,fin);

fin[u]=true; //pick the min. distance vertex from the set if vertex

for(int i=0;i<5;i++)

if(fin[i]==false && graph[u][i]!=0)

dist[i] = min(dist[i],dist[u] + graph[u][i]);

}

a= printSolution(dist, distinct);

return a;

}

//return the zipcode where the required vehicle service is available

int\* checkAvailability(int vehicleID[],int vehicleType[],int zipCode[],int v, int a[10])

{

cout <<"\n\tVehicle ID \t Zip Code " << endl;

int \*d=new int[10];

int j=0;

for(int i=0;i<18;i++)

{

if(v==vehicleType[i])

{

a[j] = vehicleID[i];

d[j++]=zipCode[i];

cout<<"\t"<<vehicleID[i]<<"\t\t "<<zipCode[i]<<endl;

}

}

return d;

}

void Dispatch(int \*distinct\_zipcode,int distance[], string emergencyVehicle,int available\_vehicle\_ID[10],int available\_zipcode[])

{

bubbleSort(distance,distinct\_zipcode,5); //sort the distinct\_zipcode & distance array acc to distance

bool found=false;

int v\_id,z,dist;

for(int i=0;i<5;i++) //for zipcode

{

for(int j=0;j<10;j++) //for all Available Vehicle

{

if(distinct\_zipcode[i]==available\_zipcode[j])

{

found=true;

v\_id=available\_vehicle\_ID[j];

z=available\_zipcode[j];

dist=distance[i];

break;

}

}

if(found)

break;

}

if(found)

cout<<"\n\t"<<emergencyVehicle<<" with Vehicle Number "<<v\_id<<" is Dispatched from zipcode "

<<z<<" at the distance "<<dist<<endl;

else

cout<<"Vehicle NOT Available";

}

void HomeScreen()

{

int i, j, k = 0;

for (i = 3; i >= 0; i--)

{

cout << "\t\t\t\t\t\t";

for (j = 3; j > k; j--)

{

cout << " ";

}

cout << "\*";

for (j = 1; j < (k \* 2); j++)

cout << " ";

if (i < 3)

cout << "\*";

cout << "\n";

k++;

}

cout << "\n\t\t\t\t\t W E L C O M E\n";

cout << "\t\t\t\t\t\t T O\n";

cout << "\t\t\t E M E R G E N C Y V E H I C L E D I S P A T C H \n\t\t\t\t\t S Y S T E M\n\n";

for (i = 3; i >= 0; i--)

{

cout << "\t\t\t\t\t\t";

for (j = 3; j > i; j--)

{

cout << " ";

}

cout << "\*";

for (j = 1; j < (i \* 2); j++)

cout << " ";

if (i >= 1)

cout << "\*";

cout << "\n";

}

cout << "\n\n";

time\_t now; // Show date and time function

time(&now);

cout << " \n\t\t\t\t\t" << ("%s\n", ctime(&now));

cout << endl<< endl;

cout << endl;

cout << endl;

}

int main()

{

HomeScreen();

int vehicleID[18]={1,2,3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19};

int vehicleType[18]={1,1,1,2,3,3,1,3,3,3,2,2,3,3,3,3,1,3};

int zipCode[18]={64151,64151,64151,64151,64151,64151,64149,64149,64149,64149,64320,

64372,64372,64372,64372,64372,63210,63210};

//64151-0, 64149-1, 64320 -2, 64372-3, 63210-4

int adjacencyMatrix[5][5]={{0,10,INF,30,100},

{10,0,50,INF,120},

{INF,50,0,20,10},

{30,INF,20,0,60},

{100,120,10,60,0}};

int zip1,v;

int \*distance;

bool serviceavailable = false;

cout<<"\n\n";

cout<<"\tZipCodes where our service is currently available\n";

cout<<"\t64151, 64149, 64320, 64372, 63210\n";

cout<<"\n\tPlease Enter Your Area Code : ";

cin>>zip1;

for(int i=0;i<18;i++)

{

if(zip1==zipCode[i])

{

serviceavailable = true;

}

}

if(!serviceavailable)

{

cout<<"\n\t\t\t\t\tSorry!!!SERVICE IS UNAVAILABLE \n\n";

exit(0);

}

system("CLS");

int \*distinct\_zipcode=printDistinct(zipCode,18);

//storing the shortest distance from source node to other zip codes

for(int i=0;i<5;i++)

{

if(zip1==distinct\_zipcode[i])

{

distance=dijkstra(adjacencyMatrix,i, distinct\_zipcode);

break;

}

}

cout<<endl<<endl;

cout<<"\tVehicle Types:\n";

cout<<"\t1 - Ambulance\n\t2 - Fire Truck\n\t3 - Police Car\n\n";

string emergencyVehicle;

while(1)

{

bool input=false;

cout<<"\n\tVehicle Type You Required : ";

cin>>v;

switch(v)

{

case 1: emergencyVehicle="Ambulance";

input=true;

break;

case 2: emergencyVehicle="Fire Truck";

input=true;

break;

case 3: emergencyVehicle="Police Car";

input=true;

break;

default : cout<<"WRONG VEHICLE TYPE ENTERED!! PLEASE ENTER AGAIN"<<endl;

}

if(input)

break;

}

system("CLS");

cout<<"\n\tVehicle Available at the locations : \n\n";

int available\_vehicle\_ID[10];

memset(available\_vehicle\_ID,0,sizeof(available\_vehicle\_ID));

int \*available\_zipcode=checkAvailability(vehicleID,vehicleType,zipCode,v, available\_vehicle\_ID);

cout<<"\n\tDispatching \n";

Dispatch(distinct\_zipcode,distance,emergencyVehicle,available\_vehicle\_ID,available\_zipcode);

return 0;

}