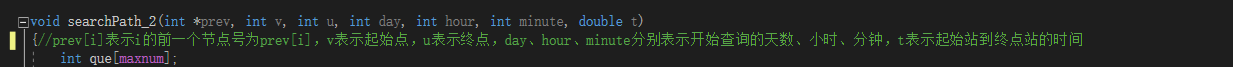


n表示矩阵的行列数，v表示起始点，dist[i]表示起始点到i的距离，prev[i]表示i的前一个节点号为prev[i]，c[][]是矩阵



prev[i]表示i的前一个节点号为prev[i]，v表示起始点，u表示终点，day、hour、minute分别表示开始查询的天数、小时、分钟

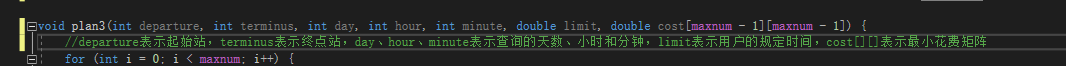


prev[i]表示i的前一个节点号为prev[i]，v表示起始点，u表示终点，day、hour、minute分别表示开始查询的天数、小时、分钟，t表示起始站到终点站的时间

v表示起始点，u表示终点，hour、minute分别表示开始查询的小时、分钟



departure表示起始站，terminus表示终点站，hour、minute表示查询的小时和分钟，limit表示用户的规定时间，cost[][]表示最小花费矩阵



departure表示起始站，terminus表示终点站，day、hour、minute表示查询的天数、小时和分钟，limit表示用户的规定时间，cost[][]表示最小花费矩阵

算法一：

先建立一个15×15的矩阵，每个点表示从城市i出发到城市j的最少费用，不可达的两个城市间的花费设置为maxint，即9999，再调用Dijkstra求出每个城市到起始城市的最少花费。

算法二：

同样先建立一个15×15的矩阵，每个点表示从城市i出发到城市j的最少花费时间，由于我们每天的线路是一样的，所以在求取最少花费时间的时候会判断当前车次结束时间与下一车次开始时间的前后判断是否需要加24来得出最少花费时间，同样，不可达的两个城市间的花费设置为maxint，即9999，再调用Dijkstra求出每个城市到起始城市的最少花费时间。

算法三：

先建立一个15×15的矩阵，每个点表示从城市i出发到城市j的最少费用，不可达的两个城市间的花费设置为maxint，即9999，再调用Dijkstra求出每个城市到起始城市的最少花费。得出一条线路，计算所花费的时间，判断是否在规定的时间内，如果在规定时间内，则所求线路是在规定时间内的最少费用，如果不是，则应该是线路中的某一段或者某几段影响了花费时间，则依次将每段两城市间的花费置为9999，再递归调用算法三，等到判断时间是一个不可能的值时会返回，判断得到最优结果值时会返回，每次递归调用前（除第一次）会将之前的改为9999的两城市之间的花费复位。

数据结构：

class Time\_cost {

public:

double begin\_time;//该车次的开始时间

double time\_cost;// 该车次的花费时间

};//用于算法二的矩阵

class vehicle {

public:

string Vec[1000];//交通工具名

string Departure\_Station[1000];//起始站名

string Terminus[1000];//终点站名

string vehicle\_number[1000];//交通工具的编号

string time\_begin[1000];//车次的开始时间

double time\_cost[1000];//车次的花费时间

double money\_cost[1000];//车次的费用

friend class city;

friend class Passenger;

vehicle (){}

~vehicle(){}

};

class route {

public:

string Vec;

string Departure\_Station;

string Terminus;

string vehicle\_number;

string time\_begin;

double time\_cost;

double money\_cost;

route(){}

~route(){}

friend class city;

friend class Passenger;

};

class Passenger {

public:

string name;

string current\_city;

string Departure\_place;

string Terminus;

route Stortest\_time;

route Min\_cost;

route time\_min\_cost;

vehicle current\_vheicle;

};

class city {

public:

string City\_Name;

vehicle train;

vehicle car;

vehicle plane;

route min\_cost[15]; //算法一路径车次信息的记录

route min\_time[15]; //算法二路径车次信息的记录

route time\_and\_cost[15]; //算法三路径车次信息的记录

city (){}

~city(){}

void init\_city(string file\_1, string file\_2, string file\_3, string city\_name); //file\_1,file\_2,file\_3分别表示火车、汽车、飞机的数据文件名，city\_name表示出发城市的名字

};