2.7 案例分析-----十字路口交通灯控制系统

本章将设计和开发一个十字路口交通灯控制系统，通过案例式教学，向读者直观展示系统的需求分析、设计、编码和测试工作。

2.7.1 交通灯控制系统问题描述

有一个十字路口，该路口有编号为 1~4 的 4 组灯，如图 2-18 所示。每组灯由 1 个左转灯和 1 个直行灯组成（面向十字路口，内侧的为左转灯，外侧为直行灯）。任一时刻每个灯或者为红色或者为绿色。当灯为绿色时，表示可以左转（或直行）。当灯为红色时，表示不能通行。通过这 8 个灯的颜色变化，来控制车辆在该十字路口的左转和直行。

在该系统中，我们假设该路口是封闭的（即汽车始终是在这个路口中来回行驶），有两辆汽车会在该路口中行驶，其中一辆车是由计算机随机控制的，另一辆车是由人在键盘输入指令来控制的。

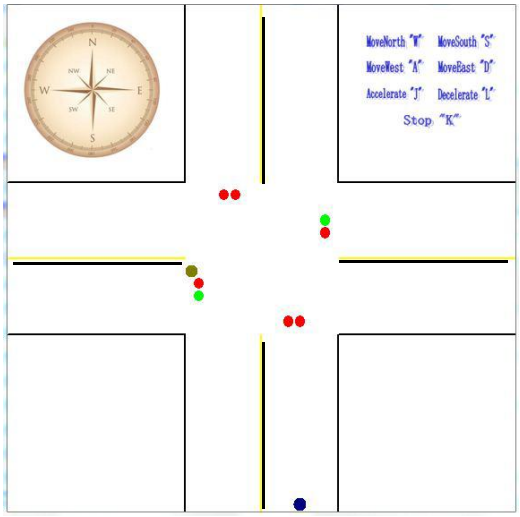


图 2－18 十字路口示意图

现在，要求我们编程实现这个十字路口的交通控制系统：模拟出两辆汽车，并使它们能按照交通规则有序运行在该路口中。

2.7.2 需求分析

开发的第一步是进行需求分析。前面我们已经说过，需求分析需要从系统的数据、功能和行为三方面进行分析。功能方面，主要是模拟出两辆小汽车在交通灯指挥下按交通规则行驶在一个十字路口。我们要分析清楚十字路口道路分布情况以及十字路口交通灯分布情况；行为方面，需要分析 8 个交通灯整体配合的行为（规则）和车辆运行行为（规则）；数据方面，需要分析清楚人控制车辆运行的控制指令。

（一）交通道路配置：

1.道路基本类型：

实际的道路可以有双向道路和单向道路（只能单向行驶）。本系统只考虑双向道路，如图 2－19 所示：

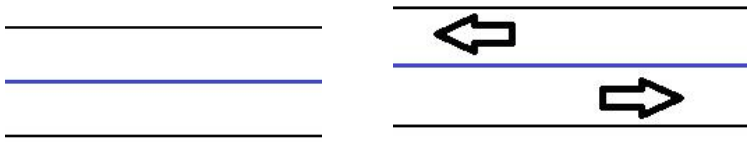


图 2－19　双向路

在此道路中，车辆可以朝两个方向行驶，黄线（即图 2－19 中间那条线）两侧都有车辆，可以含多条车道，黄线两侧的车辆行驶方向不同。黄线为界限，在一侧的车辆不可越过黄线，进入另一侧。车辆在本道路中以自身为参照物的前提下，靠右行驶。如图 2－19 的箭头标明为正确方向。

程序需要考虑双向路的转弯和掉头问题。在图 2－20 中，车辆可在两个方向进行转弯。在图 2－21 中，车辆可以掉头。

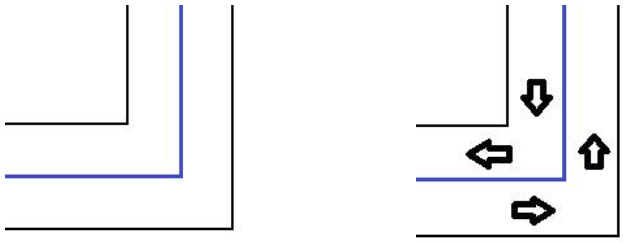


图 2－20 双向路拐弯

图 2－21 双向路掉头

2.交通路口基本类型：

实际的交通路口有十字路口和丁字路口两类，本系统中只考虑十字路口，如下图 2－22 所示。

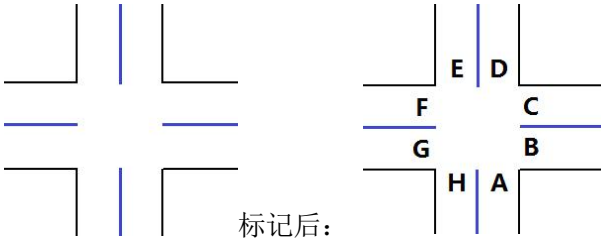


图 2－22 双向道路交叉十字路口

这类路口在交通系统中是最常见的，也是最基本的。当给图 2－22 中路口的各路口标号后，车辆运行方式为：

当 A 路中的车辆到达路口后，可以进行三种选择：直行会进入 D 路，左拐会进入 F 路，右拐会进入 B 路。E 路中的车辆到达路口后，可以进行三种选择：直行会进入 H 路，左拐会进入 B 路，右拐会进入 F 路。其他路口车辆运行方式类似。

（二）交通信号灯配置：

要求十字路口的交通信号灯配置如图 2－23 所示。

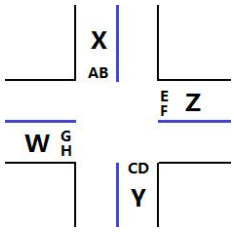


图 2－23 十字路口交通信号灯配置

在双向十字路口中，最普通的情况，需要配置 8 盏交通灯。例如，其中 X 路段中有 AB 两盏信号灯，其中 A 信号灯控制 X 路段的直行车辆，B 信号灯控制 X 路段的左拐车辆。X 路段的右拐车辆不需信号灯控制，随时都可右拐。Y 路段由 CD 两盏共同控制，D 灯控制 Y 路段中的直行车辆，C 灯控制 Y 路段的左拐车辆。从中可以发现，A 灯与 D 灯的显示信息应该相同，B 灯与 C 灯的显示信息也应相同。W 与 Z 路段交通灯配置方式相同。

（三）车辆的行驶规则与要求：

1. 车辆的表示：

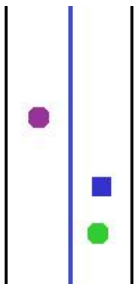


图 2－24 车辆的表示

在道路上行驶的汽车，可以用不同的实心圆形或方形表示，也可以用其他形状或者贴图，对汽车的外观不做过多要求。如图 2－24 所示。

2. 车辆运行轨迹：

当车辆直线行驶时，其运行方向是直线；当车辆需要拐弯时，其拐弯轨迹是直角 90 度，如图 2－25 所示。

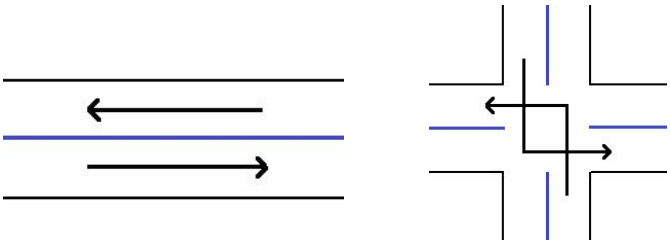


图 2－25 车辆的运行轨迹

3. 车速的设计要求：

在道路上的汽车，全部以恒定相同的速度在公路上运行，当遇到红灯时车速直接变为 0。

4. 车辆与车辆之间的关系：

因为前面已经假设所有的车辆车速均相同且恒定，所以不会有超车的现象发生，程序也不用考虑车辆之间的超车问题。但是如果有车辆因为等待绿灯而暂停，则其他车辆行驶到靠近该车辆时，需要停车，以免发生车辆相撞的事件。

5. 车辆与信号灯之间的关系：

当车辆遇到红色信号灯时，应该在信号灯前指定位置处停止。遇到绿色信号灯时，车辆应该立即启动。左转和直行的车辆如都遇到红灯，应停止在对应的信号灯前。如图 2－26 所示。

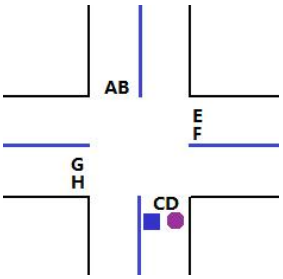


图 2－26 车辆等待信号灯

6.车辆的数量：

实际道路上车辆数量会实时变化，本系统假设就只有 2 辆车会经过该十字路口，一辆是计算机控制，一辆是人通过输入指令来控制。

（四）交通信号灯的运行规则与设计要求：

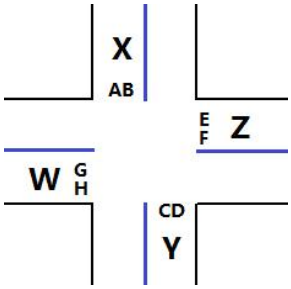


图 2－27 十字路口交通信号灯配置

如图 2－27 所示， A、D 控制 X、Y 路段车辆的直行，其运行周期相同。B、C 控制 X、Y 路段车辆的左拐，其运行周期相同。H、E 控制 W、Z 路段车辆的直行，其周期相同。G、F 控制 W、Z 路段的左拐，其周期相同。假设不考虑交通流量对信号灯持续时间的影响，8 盏灯从亮到灭持续时间均为 2 秒钟，绿灯亮的顺序为图 2－28 所示，顺序依次为(1)-> (2)-> (3)-> (4)，即先是直行灯 A 和 D 亮，2 秒钟之后是左转灯 B 和 C 亮，2 秒钟之后是直行灯 H 和 E

亮，2 秒钟之后是左转灯 G 和 F 亮。可见每盏灯变绿的周期是 6 秒。

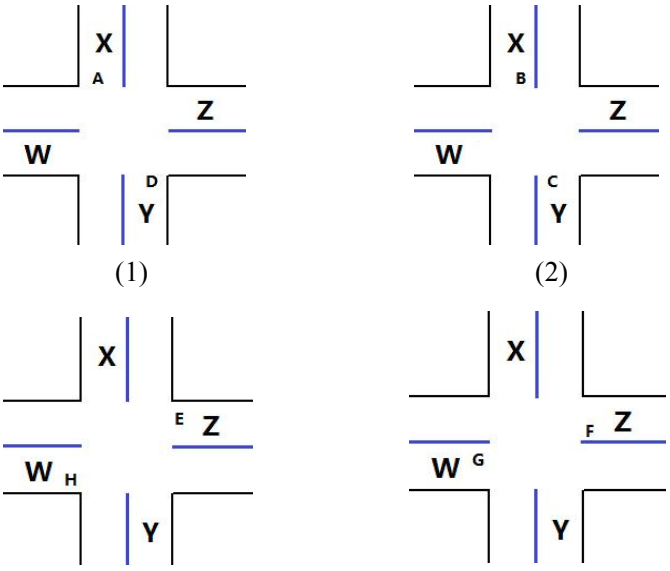


图 2－28 十字路口绿灯亮的顺序

（五）输入输出：

1．输入：

交通系统中的输入数据包括初始化数据和驾驶指令。

初始化数据：可以从键盘输入，也可以从磁盘文件中读取。应初始化的数据包括汽车速度，公路宽度，信号灯周期等等。

驾驶指令：由人通过键盘输入。人充当驾驶员的角色，通过输入指令来控制其中一辆汽车的运行。控制指令如下：‘W’(向北运行)，‘S’(向南运行)、‘A’(向西运行)、‘D’(向东运行)，‘J’(加速)、‘L’(减速)，‘K’(停靠)。

2．输出：

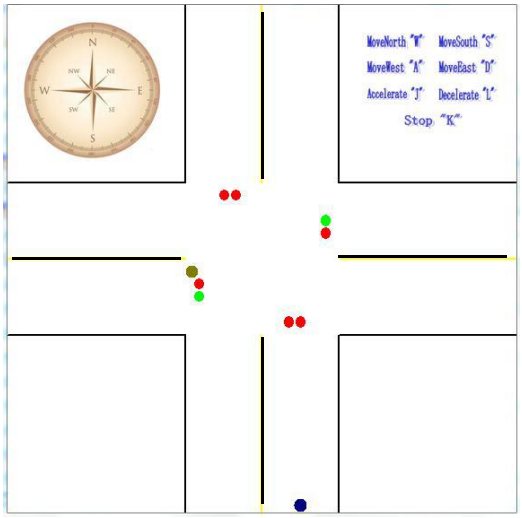
a．汽车运行的动画显示：公路，汽车当前位置，信号灯位置，信号灯的颜色以及持续时间。信号灯的周期变色要明显体现。汽车运行受信号灯控制，应在动画中明显的体现出来。

b．汽车运行情况的记录（结果）文件，每隔 5 秒记录一次，需要记录汽车运行状态和方向，汽车当前位置，每个信号灯的颜色以及持续时间。

2.7.3 概要设计

概要设计主要任务有两个，一是对用户界面进行设计，二是对系统进行模块划分，对全局数据结构和重要的算法进行设计。十字路口交通灯控制系统概要设计书如下：

1.用户界面设计



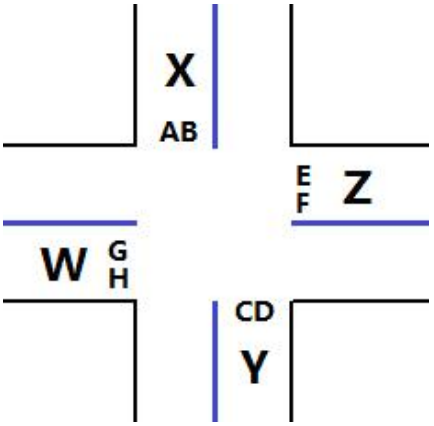


图 2－29 十字路口界面设计　　图 2－30 路口交通灯配置

本程序用户界面设计如图 2－29 所示。

在此道路中，车辆可以朝两个方向行驶，黄线两侧都有车辆，黄线两侧的车辆行驶方向不同。黄线为界限，在一侧的车辆不可越过黄线，进入另一侧。车辆在本道路中以自身为参照物的前提下，靠右行驶。用四种颜色（除红绿，如图中的蓝色实心圆球）实心圆形表示汽车，用红绿两种实心圆形表示红绿交通灯。

在双向十字路口中，最普通的情况，需要配置 8 盏交通灯。如图 2－30 所示。例如，其中 X 路段中有 AB 两盏信号灯，其中 A 信号灯控制 X 路段的直行车辆，B 信号灯控制 X 路段的左拐车辆。X 路段的右拐车辆不需信号灯控制，随时都可右拐。Y 路段由 CD 两盏共同控制，D 灯控制 Y 路段中的直行车辆，C 灯控制 Y 路段的左拐车辆。从中可以发现，A 灯与 D 灯的显示信息应该相同，B 灯与 C 灯的显示信息也应相同。W 与 Z 路段交通灯配置方式相同。

界面的左上方是一个指南针，指向为上北下南，左西右东。

界面右上方是车辆行驶的输入方式，当用户控制汽车运行时，指令由键盘输入，共 7 条指令，如下：

（1）输入‘W’表示车辆要向北行驶。

（2）输入‘S’表示车辆要向南行驶。

（3）输入‘A’表示车辆要向西行驶。

（4）输入‘D’表示车辆要向东行驶。

（5）输入‘J’表示车辆要加速。

（6）输入‘L’表示车辆要减速。

（7）输入‘K’表示车辆要停靠。

若由计算机控制汽车运行，汽车会随机自动运行，无需输入。

2 自动机模型（状态转换图）

在本题目中，我们需要分析清楚汽车以及交通灯这两者的行为，在此我们使用状态图来进行描述。

A. 汽车状态图：

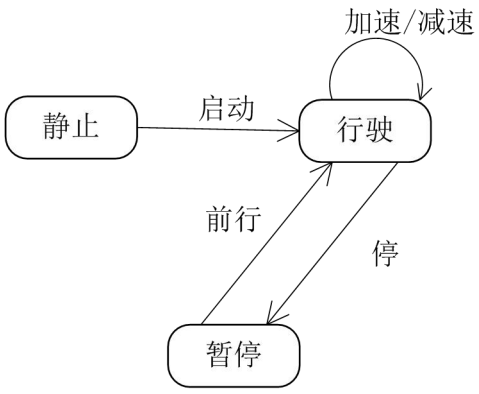


图 2－31 汽车的状态图

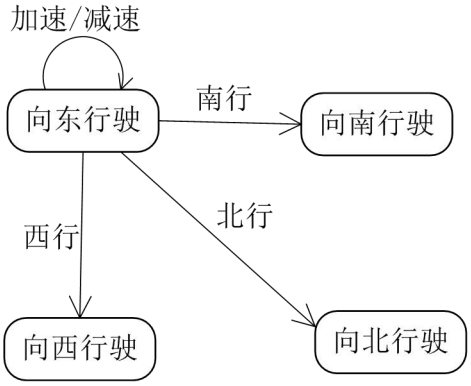


图 2－32 “行驶”状态的细化（部分）

不管是计算机控制的汽车还是人控制的汽车，都是 3 个状态：静止、行驶和暂停，如图 2－31 所示。在静止状态下，若接收到启动命令，则进入行驶状态；在行驶状态下，若接到停的命令则进入暂停状态；在暂停状态下若接收到前行命令则进入行驶状态。本系统中不考虑汽车熄火的问题，所以从暂停状态不会迁移到静止状态。

对于行驶状态，我们可以进一步细化成向东行驶、向南行驶、向西行驶和向北行驶四个状态。图 2－32 给出了在向东行驶状态下接收到各个命令时的状态图。在其他三个状态下接收到各命令时的状态转换类似，为保持图的简洁，此处就不再给出。

对于计算机控制的汽车，它接收到的命令是由系统产生的随机数随机决定的；对于人控制的汽车，它接收到的命令是由人给出的。

B.交通灯状态图：

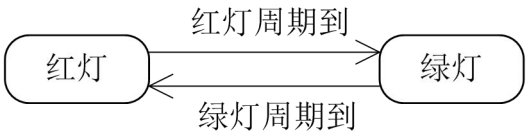


图 2－33 交通灯的状态图

因为本系统不考虑交通灯从灭到亮的过程，所以每一盏交通灯只有两个状态，红灯和绿灯，如图 2－33 所示。若在红灯状态下，红灯亮的时间到达了红灯周期，则状态转换为绿灯；若在绿灯状态下，绿灯亮的时间到达了绿灯周期，则状态转换为绿灯。

本系统中的十字路口会有 8 盏灯，每两盏灯的状态转换是一致的，所以实际编程中应计算好红绿灯的周期（红灯周期是绿灯周期的 3 倍），才能保证 8 盏灯的有序变色。

3 高层数据结构设计

3.1 常量定义

```
//定义接近 0 的常量
#define EXP 0.000001//
```

```
//汽车运行方向常量设置
#define NORTH 0
#define WEST 1
#define SOUTH 2
#define EAST 3
#define STOP 4
//交通灯当前颜色的状态常量设置
#define RedS 0//
#define GreenS 1//
```

3.2 全局变量定义

```
struct Car Auto;//计算机自动控制的汽车
struct Car Customer;//用户控制的汽车
double CustomSpeed;//汽车的速度
double lightlocX[8]={11.5,8.5,11.05,8.95,12.5,7.5,12.5,7.5};// 交通灯 X 坐标
double lightlocY[8]={7.5,12.5,7.5,12.5,11.5,8.5,11,9};// 交通灯 Y 坐标
```

3.3 数据结构的定义

(1) 信号灯的结构：

```
struct light
{
    int state,greenCycle,redCycle;
    time_t stime;
};
```

信号灯的状态（state），绿色灯周期（greenCycle）,红色灯周期（redCycle）
信号灯变灯时间（stime）

(2) 汽车的结构：

```
struct Car
{
    int state;
    double x,y;
};
```

包括汽车的状态(state),与汽车的坐标<x,y>

4. 系统模块划分

一. 软件结构图

本系统程序部分划分为 main.c 、streetDraw.c，carDraw.c，carControl.c，lightDraw.c，lightControl.c，input.c 七个模块。各模块功能如下：

1. 模块名称：main.c

模块功能简要描述：主函数，主要是运行各个界面绘画程序，运行计算机控制的汽车的运行情况和用户控制的汽车运行情况（接收人工输入）。

2. 模块名称：streetDraw.c

模块功能简要描述：绘制十字路口界面。

3．模块名称 carDraw.c

模块功能简要描述：在十字路口界面上绘制汽车，并控制汽车向东南西北四个方向移动。

4．模块名称 carControl.c

模块功能简要描述：

（1）根据北、南、东、西四个方向检测用户控制的汽车,使用户控制的汽车不能超过公路边界。

（2）根据北、南、东、西四个方向与停止状态检测用计算机控制的汽车,

（3）完成对计算机控制汽车的线程和用户控制汽车的线程的实现。

5．模块名称 lightDraw.c

模块功能简要描述：绘制信号灯,并给信号灯设置红灯周期与绿灯周期，设置信号灯颜色,并设置红绿色灯启动的开始时间。

6． 模块名称 lightControl.c

模块功能简要描述：当信号灯的变灯时间到达时,改变信号灯的颜色,使信号灯的颜色相反。完成控制交通灯的线程的实现,首先设置交通灯的起始时间,并根据当前时间与交通灯周期改变交通灯的状态。

7．模块名称 input.c

模块功能简要描述：完成对用户输入的线程的实现,根据输入改变汽车的状态。

软件结构图如图 2－34 所示。

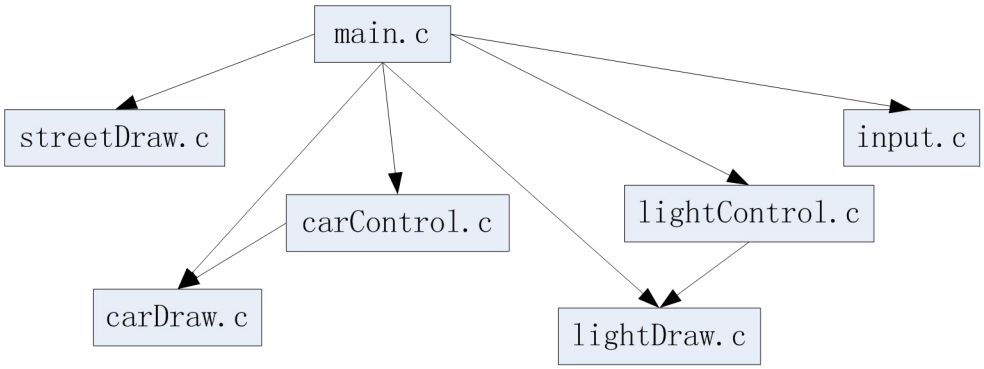


图 2－34 软件结构图

二．文件及函数组成

表 2－2 文件及函数功能描述

源文件	源文件说明	函数名	功能
carControl.c	本文件中的代码用于控制两辆汽车的运行	int checkCustomNORTH()	检测用户控制的汽车,使其运行不能超过公路边界
		Int checkCustomSOUTH()	检测用户控制的汽车,使其运行不能超过公路边界
		int checkCustomEAST()	检测用户控制的汽车,使其运行不能超过公路边界
		int checkCustomWEST()	检测用户控制的汽车,使其运行不能超过公路边界
		Void checkAutoSOUTH()	检测南向运行的汽车下一个时刻向哪个方向运行
		Void checkAutoNORTH()	检测北向运行的汽车下一

			个时刻向哪个方向运行
		void checkAutoEAST()	检测东向运行的汽车下一个时刻向哪个方向运行
		void checkAutoWEST()	检测西向运行的汽车下一个时刻向哪个方向运行
		void checkAutoSTOP()	检测处于停靠状态的汽车下一个时刻向哪个方向运行
		DWORD WINAPI AutoCar(LPVOID lpParameter)	计算机控制汽车向北、南、东、西四个方向运行或停止
		DWORD WINAPI CustomCar(LPVOID lpParameter)	用户控制汽车向北、南、东、西四个方向运行或停止
carDraw.c	在十字路口界面上绘画汽车模型，并控制汽车向东南西北四个方向移动。	void CreateCar(struct Car *ptr,int _state);	根据方向创建汽车
		void MoveNorth(struct Car *ptr,double l)	控制向北移动汽车
		void MoveSouth(struct Car *ptr,double l)	控制向南移动汽车
		void MoveEast(struct Car *ptr,double l)	控制向东移动汽车
		void MoveWest(struct Car *ptr,double l)	控制向西移动汽车
		void setCarActive (struct Car *ptr)	设置汽车为活动汽车
lightControl.c	改变信号灯的颜色,完成控制交通灯的线程的实现。	void ChangeLight (struct light *ptr,time_t now)	当信号灯的变灯时间到达时,改变信号灯的颜色,使信号灯的颜色相反
		DWORD WINAPI Light (LPVOID lpParameter)	控制交通灯的线程
lightDraw.c	绘制信号灯,并给信号灯设置红灯周期与绿灯周期，设置信号灯颜色,并设置红绿色灯启动的开始时间。	void CreateLight(int _greenCycle,int _redCycle)	绘制信号灯,并给信号灯设置红灯周期与绿灯周期
		void SetLightRed (struct light *ptr,time_t _stime)	将信号灯颜色变为红色,并设置红色灯启动的开始时间_stime
		void SetLightGreen (struct light *ptr,time_t _stime)	将信号灯颜色变为绿色,并设置绿色灯启动的开始时间_stime

streetDraw.c	绘制十字路口	void DrawSys()	绘制十字路口
input.c	完成对用户输入的线程的实现,根据输入改变汽车的状态。	DWORD WINAPI InputOrder(LPVOID lpParameter)	用户输入的线程

三. 函数说明

表 2—3 函数说明

序号	函数原型	功能	参数	返回值
1	int checkCustomNORTH()	检测用户控制的汽车,使其运行不能超过公路边界	void	根据公路坐标判断北向行驶的汽车是否超过公路边界，超界返回 1，否则返回 0
2	int checkCustomSOUTH()	检测用户控制的汽车,使其运行不能超过公路边界	void	根据公路坐标判断南向行驶的汽车是否超过公路边界，超界返回 1，否则返回 0
3	int checkCustomEAST()	检测用户控制的汽车,使其运行不能超过公路边界	void	根据公路坐标判断东向行驶的汽车是否超过公路边界，超界返回 1，否则返回 0
4	int checkCustomWEST()	检测用户控制的汽车,使其运行不能超过公路边界	void	根据公路坐标判断西向行驶的汽车是否超过公路边界，超界返回 1，否则返回 0
5	void checkAutoSOUTH()	检测南向运行的汽车下一个时刻向哪个方向运行	void	void
6	Void checkAutoNORTH()	检测北向运行的汽车下一个时刻向哪个方向运行	void	void
7	void checkAutoEAST()	检测东向运行的汽车下一个时刻向哪个方向运行	void	void
8	void checkAutoWEST()	检测西向运行的汽车下一个时刻向哪个方向运行	void	void
9	void checkAutoSTOP()	检测停靠状态的汽车下一个时刻向哪个方向运行	void	void
	DWORD WINAPI AutoCar	计算机控制汽车向北、南、东、西四	LPVOID 是一个没有类型的指针，可以将任	DWORD WINAPI 返回 DWORD(32 位数据)的 API 函数

10	(LPVOID lpParameter)	个方向运行或停止	意类型的指针赋值给LPVOID类型的变量	
11	DWORD WINAPI CustomCar (LPVOID lpParameter)	用户控制汽车向北、南、东、西四个方向运行或停止	LPVOID	DWORD WINAPI 返回 DWORD(32位数据)的 API 函数
12	void CreateCar(struct Car *ptr,int _state)	根据方向创建汽车	ptr是指向汽车的一个指针，_state是汽车的初始运行方向	void
13	void MoveNorth(struct Car *ptr,double l)	控制向北移动汽车	ptr是指向汽车的一个指针,l是车每次的移动距离	void
14	void MoveSouth(struct Car *ptr,double l)	控制向南移动汽车	ptr是指向汽车的一个指针,l是车每次的移动距离	void
15	void MoveEast(struct Car *ptr,double l)	控制向东移动汽车	ptr是指向汽车的一个指针,l是车每次的移动距离	void
16	void MoveWest(struct Car *ptr,double l)	控制向西移动汽车	ptr是指向汽车的一个指针，l是车每次的移动距离	void
17	void setCarActive (struct Car *ptr)	设置汽车为活动汽车	ptr是指向汽车的一个指针	void
18	void ChangeLight (struct light*ptr,time_t now)	改变交通灯颜色	ptr是指向信号灯的一个指针,now是当前时间	void
19	DWORD WINAPI Light(LPVOID lpParameter)	控制交通灯的线程	LPVOID是一个没有类型的指针，可以将任意类型的指针赋值给LPVOID类型的变量	DWORD WINAPI 返回 DWORD(32位数据)的 API 函数
20	void CreateLight(int _greenCycle,int _redCycle)	绘制信号灯,并给信号灯设置红灯周期与绿灯周期	greenCycle、_redCycle 分别为红绿灯周期	void

21	void SetLightRed(struct light *ptr,time_t _stime)	将信号灯颜色变为红色,并设置红色灯启动的开始时间_stime	ptr 是指向信号灯的一个指针，_stime 是红色灯启动的开始时间	void
22	void SetLightGreen(struct light *ptr,time_t _stime)	将信号灯颜色变为绿色,并设置绿色灯启动的开始时间_stime	ptr 是指向信号灯的一个指针，_stime 是绿色灯启动的开始时间	void
23	void DrawSys()	绘制十字路口	void	void
24	DWORD WINAPI InputOrder(LPVOID lpParameter)	用户输入的线程	LPVOID 是一个没有类型的指针，可以将任意类型的指针赋值给 LPVOID 类型的变量	DWORD WINAPI 返回 DWORD(32 位数据)的 API 函数

5. 高层算法设计

下面给出该工程中几个高层算法的伪代码：

信号灯的变灯函数：

```
void ChangeLight(struct light *ptr,time_t now)
{
  if 信号灯为红色
     if 信号灯的时间到达红色交通灯的周期
        信号灯变成绿色
  else if 信号灯为绿色
     if 信号灯的时间到达绿色交通灯的周期
        信号灯变成红色
}
```

►

用户控制汽车运行的线程：

```
DWORD WINAPI CustomCar(LPVOID lpParameter)
{
  while(1)
  {
      /*检测用户控制运行的汽车的运行状态，做出相应的处理：修改状态为 STOP 或者
        继续前行；汽车运行方向的变化是在 InputOrder 线程中根据用户输入的指令进
        行修改的*/
      switch 汽车状态
      {
        case 向北行驶：
            if 检测到北行的汽车超过公路边界
              汽车的运行状态变为 STOP;
            else
              汽车向北移动一段位移;
            break;
        case 向南行驶：
            if 检测到南行的汽车超过公路边界
              汽车的运行状态变为 STOP;
            else
              汽车向南移动一段位移;
            break;
        case 向西行驶：
            if 检测到西行的汽车超过公路边界
              汽车的运行状态变为 STOP;
            else
              汽车向西移动一段位移;
            break;
        case 向东行驶：
            if 检测到东行的汽车超过公路边界
              汽车的运行状态变为 STOP;
            else
              汽车向东移动一段位移;
            break;
    }
  }
  return 0;
}
```

计算机控制汽车运行的线程：

```
DWORD WINAPI AutoCar(LPVOID lpParameter)
{
    while(1)
    {
        //检测自动运行的汽车的运行状态，做出相应方向上的处理
        switch 汽车状态
        {
          case 向北行驶：
                汽车向北移动一段位移
                if 汽车到达需要改变方向的位置
                      改变汽车运行的方向
                break;
          case 向南行驶：
                汽车向南移动一段位移
                if 汽车到达需要改变方向的位置
                      改变汽车运行的方向
                break;
          case 向西行驶：
                汽车向西移动一段位移
                if 汽车到达需要改变方向的位置
                      改变汽车运行的方向
                 break;
          case 向东行驶：
                汽车向东移动一段位移
                if 汽车到达需要改变方向的位置
                     改变汽车运行的方向
                break;
          case 暂停：
                if 汽车到达需要改变方向的位置
                      改变汽车运行的方向
                break;
        }
    }
    return 0;
}
```