《程序设计课程设计》实验报告

实验名称　　《外卖系统程序设计》概要设计<第一版>

班　　级　　**2018211319**

组　　号

姓　　名　　张翾、王天乐、孙心华

1．用户界面设计

1.1 用户界面

图 1-1-1 外卖派送区域示例

图 1-1-2 A 餐馆所在位置示例

图 1-1-3 B 餐客所在位置示例

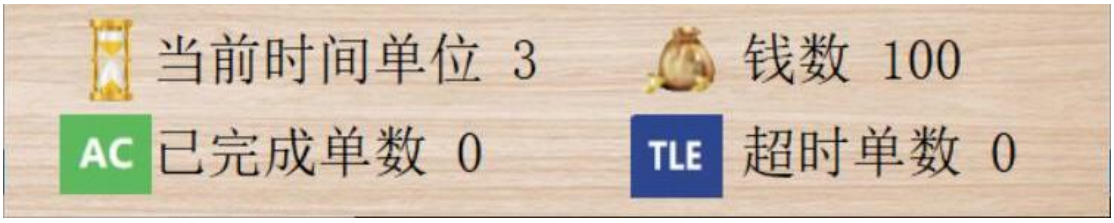

图 1-1-4 动画输出框示例

在此道路中，如图 1-1-1 所示，骑手的取、送餐区域必须在 16*16 范围内，房屋既可以是接单的餐馆如图 1-1-2，也可以是下订单的食客家如图 1-1-3。房屋之间的 8*8 条街道是骑手唯一可走的道路；骑手停在房屋的上下左右街道，即算抵达。骑手走过每个房屋的距离一样，速度也一样，拐弯不花时间，经过路口不花时间。

动画输出：在图形窗口中绘制房间、道路，动态显示骑手位置变化，停靠餐馆或食客时要有明显变化；并在窗口中实时刷新显示钱数、接单数、完成数、超时数如图 1-1-4。

1.2 操作元素和操作效果

当骑手接到派单时，餐馆所在位置的房屋会变成图 1-1-2 所示，食客家所在位置会变成图 1-1-3 所示，输出动画中区域如图 1-2-1。每当按时完成一单派送任务时，动画输出框中已完成单数会增加 1，钱数会增加 10，如图 1-2-2；当某单超过 30 个单位时间但未超过 60 个单位时间送达时，动画输出框中超时单数会增加 1，钱数会减少 50；当某单超过 60 个单位时间未送达时，直接破产，如图 1-2-2。

。

图 1-2-1　出现订单示例

图 1-2-2　完成订单示例

图 1-2-2　破产示例

3 高层数据结构设计

3.1 全局常量/变量定义

```
#define INITIAL_AMOUNT 1000 //初始运营资本
#define PRICE 300 //招聘一位骑手的费用
#define SALARY 10 //完成一单的收入
#define FINE 50 //超过 30 个时间单位的罚款
#define TIMEOUT 30 //超时：30 个时间单位
#define BANKRUPTCY 60 //破产：60 个时间单位
#define XLIMIT 16 //横坐标
#define YLIMIT 16 //纵坐标
```

```
#define HOUSE_X 56 //房子坐标的长
#define HOUSE_Y 30 //房子坐标的宽
```

3.2 各模块常量与变量定义

1. 主函数.c

```
int count//循环变量
struct Rider *Rider[10]//骑手数组
int num//骑手数量
struct SplitTask *SplitTask = NULL, *temp//拆分任务链表
struct Task *Task = NULL//读入任务链表
```

2.模块名称 读入数据.c

```
int locationx1, locationy1
// locationx1 餐馆横坐标,locationy1 餐馆纵坐标
int locationx2, locationy2
// locationx2 食客横坐标,locationy2 食客纵坐标
```

3.模块名称 任务拆分.c

```
SPLITTASK *current2 = NULL//链表当前指针
SPLITTASK *last2 = head2//上一个位置的指针
TASK *current1 = head1//传入的链表的头指针
```

4.模块名称 派单.c

```
SPLITTASK  *current1 = head//链表头指针地址
SPLITTASK  *current1//骑手 1 待派送链表指针
SPLITTASK  *current2//骑手 2 待派送链表指针
SPLITTASK  *current3//骑手 3 待派送链表指针
```

5.模块名称 骑手路径.c

```
int count 循环变量
struct SplitTask *temp,*temp2//拆分任务的指针
struct Order *order1=statement//已完成订单链表
struct *order2=ticket//罚单链表
```

6.模块名称 打印.c

```
int i//循环控制变量
int flag
//flag 为 0 输出餐馆；flag 为 1 输出食客；flag 为 2 输出餐客
SPLITTASK *temp = NULL//拆分订单链表当前指针
```

7.模块名称 动画输出.c

```
int i, j//循环变量
char s[50]//存储需要输出的字符
```

PIMAGE img[6]//保存图像

8.模块名称 释放链表.c
TASK*p = head//任务链表指针
SPLITTASK*p = head//拆分任务链表指针

3.3 数据结构的定义

```
typedef struct Order//存储结单，罚单序号
{
    int num;//订单数量
    struct Order *next;//下一结点指针
}ORDER;

typedef struct Task//读入数据
{
    int num;//订单数量
    int time;//下单时间
    int locationx1, locationy1, locationx2, locationy2;
     //餐馆位置、食客位置
    struct Task *next;//下一结点指针
}TASK;

typedef struct SplitTask
{
    int num;//订单序号
    int time;//下单时间
    int locationx, locationy;//需要到达位置的横纵坐标
    int reachable;//是否可达（0否1是）
    char type;//取件还是寄件（A取B寄）
    struct SplitTask *next;//下一结点指针
}SPLITTASK;

typedef struct Rider
{
    int locationx, locationy;//当前所在位置横纵坐标
    int area;//骑手管辖区域（0跨区1左2右）
    int task;//目前任务数
    int time;//完成已接订单所需最短时间
    int direction;//骑手方向
    SPLITTASK *next;//下一结点指针
    SPLITTASK *former;//上一结点指针}RIDER;

typedef struct PrintResult//打印结果
```

```
{
    int time;//当前时间
    int money;//当前金额
    int receiptCnt;//接单数
    int compeleteCnt;//完成数
    int timeoutCnt;//超时数
}PRINTRESULT;
```

4 系统模块划分

4.1 系统模块结构图

图 4-1 模块结构图示例

1.模块名称 主函数.c

独立线程：鼠标键入、音乐背景

模块功能简要描述：按要求进行时间单位记录，初始化骑手各项数据，调用多线程，调用各函数完成输出

2.模块名称 读入数据.c

模块功能简要描述：完成对文件或鼠标输入的订单数据获取，保存到订单链表中。

3.模块名称 任务拆分.c

模块功能简要描述:对已读入数据进行拆分,分为取餐任务 A 和送餐任务 B 两个链表。

4.模块名称 派单.c

模块功能简要描述：对于订单链表，根据骑手数量和骑手区域划分进行派单，将待处理订单根据时间派给骑手，加入待派送队列。

5. 模块名称 骑手路径.c
模块功能简要描述：对于每个骑手的代送订单，规划出最优路径，在最短时间内完成最多订单派送。

6. 模块名称 打印.c
模块功能简要描述：对于完成或超时的订单，在文件中输出骑手及派送订单详细信息。

7. 模块名称 动画输出.c
模块功能简要描述：对于完成或超时的订单，在动画中输出骑手及派送订单位置信息。

8. 模块名称 释放链表.c
模块功能简要描述：对于已完成的所有订单，释放其链表。

4.2 各模块函数说明

源文件	序号	函数原型	功能	参数	返回值
读入数据	1	TASK*scanfData(PRINTRESULT *result, int *max)	读取 sales 文件并创建订单链表	PRINTRESULT *result //记录需要输出变量指针 int *max //最后一单下单时间	链表头指针
任务拆分	1	Struct SplitTask *splitLinkList(TASK *head1)	拆分订单任务，分为餐馆位置和食客位置信息	TASK *head1 //读入任务链表的指针	链表头指针
派单	1	int riderJudge(SPLITTASK *head)	区域判断：（骑手数量>=3）左侧区域分给骑手 2，右侧区域份购骑手 3，跨区域骑手 1	SPLITTASK *head //拆分任务链表的指针	骑手类型 123

	2	SPLITTASK*copyNode(SPLITTASK *currentPtr) *currentPtr)	拷贝结点信息	SPLITTASK *currentPtr) *currentPtr //结点	无
	3	void distribute(TASK *headPtr, int moment, struct Rider **rider, struct PrintResult *result, int num)	将任务拆分，加入待派送队列；待任务处理完成	TASK *headPtr //任务链表指针 int moment //当前时刻 struct Rider **Rider //骑手结构体指针数组 struct PrintResult *result //打印链表指针 int num //骑手数量	无
	4	FreeSplitTask(head)	释放拆分链表	head //指针	无
骑手路径	1	int success(struct Rider **Rider,int num,int time,struct PrintResult *PrintResult,struct Order *statement,struct Order *ticket)	优化路线，并判断是不是不要动就能到一个餐馆或者食客位置	struct Rider **Rider //骑手结构体指针数组 int num //骑手数量 int time //当前时刻 struct PrintResult *PrintResult //打印结果指针 struct Order *statement //结单链表指针 struct Order *ticket //罚单链表指针	如果订单时间超过60个时间单位return-1
	2	void moveto(struct Rider *Rider,int x,int y)	让骑手移动并判断移动之后是不是到了一个餐馆或者食客位置	struct Rider *Rider //骑手链表指针 int x //当前横坐标 int y //当前纵坐标	无

	3	int path(struct Rider **Rider,int num,int time,struct PrintResult *PrintResult,struct Order *statement,struct Order *ticket)	让骑手向指定位置（餐馆或者食客）移动	struct Rider **Rider
				//骑手结构体指针数组
				int num
				//骑手数量
				int time
				//当前时刻
				struct PrintResult *PrintResult
				//打印结果指针
				struct Order *statement
				//结单链表指针
				struct Order *ticket
				//罚单链表指针
					如果订单时间超过60个时间单位return-1

打印	1	void printResult(PRINTRESULT *printresult, int num, RIDER **rider, ORDER *head1, ORDER *head2, FILE *fp)	在命令行中输出时间、钱、接单数、完成数、超时数、骑手位置及停靠时餐馆、食客信息	struct PrintResult *PrintResult
				//打印结果指针
				int num
				//骑手数量
				struct Rider **Rider
				//骑手结构体指针数组
				ORDER *head1
				//订单指针
				ORDER *head2
				//订单指针
				FILE *fp
				//指向文件的指针
					无

	2	Void printResultInFile(PRINTRESULT *printresult, int num, RIDER **rider, ORDER *head1, ORDER *head2)	向文件中输出时间、钱、接单数、完成数、超时数、骑手位置及停靠时餐馆、食客信息	struct PrintResult *PrintResult
				//打印结果指针
				int num
				//骑手数量
				struct Rider **Rider
				//骑手结构体指针数组
				ORDER *head1
				//订单指针
				ORDER *head2
				//订单指针
					无

				FILE *fp //指向文件的指针	

动画输出	1	void runRider(RIDER **rider, int num,PRINTRESULT*PrintResult)	输出动画窗口，实现骑手路线可视化	struct Rider **Rider
				//骑手结构体指针数组
				int num
				//骑手数量
				int time
				//当前时刻
				struct PrintResult *PrintResult
				//打印结果指针
					无

释放链表	1	void FreeTask(TASK*head)	释放 Task 链表	TASK*head //任务链表指针	无
	2	void FreeSplitTask(SPLITTASK*head)	释放 SplitTask 链表	SPLITTASK*head //拆分任务链表指针	无

TAKEAWAY.h	1	typedef struct Order ORDER	储存结单、罚单序号	int num
				//订单数量
				struct Order *next
				//下一结点指针
					无

	2	typedef struct Task TASK	储存读入订单数据的链表	int num
				//订单数量
				int time
				//下单时间
				int locationx1, locationy1, locationx2, locationy2
				//餐馆位置、食客位置
				struct Task *next
				//下一结点指针
					无

	3	typedef struct SplitTask SPLITTASK	储存拆分后的链表	int num
				//订单序号
				int time
				//下单时间
				int locationx, locationy
				//需要到达位置的横纵坐标
				int reachable
					无

				//是否可达（0 否 1 是）
				char type
				//取件还是寄件（A 取 B 寄）
				struct SplitTask *next
				//下一结点指针	

	4	typedef struct Rider RIDER	储存骑手信息	int locationx, locationy
				//当前所在位置横纵坐标
				int area
				//骑手管辖区域（0 跨区 1 左 2 右）
				int task
				//目前任务数
				int time
				//完成已接订单所需最短时间
				int direction
				//骑手方向
				SPLITTASK *next
				//下一结点指针
				SPLITTASK *former
				//上一结点指针
					无

	5	Typedef struct PrintResult PRINTRESULT	储存打印结果信息	int time
				//当前时间
				int money
				//当前金额
				int receiptCnt
				//接单数
				int compeleteCnt
				//完成数
				int timeoutCnt
				//超时数
					无

4.3 函数调用图示及说明

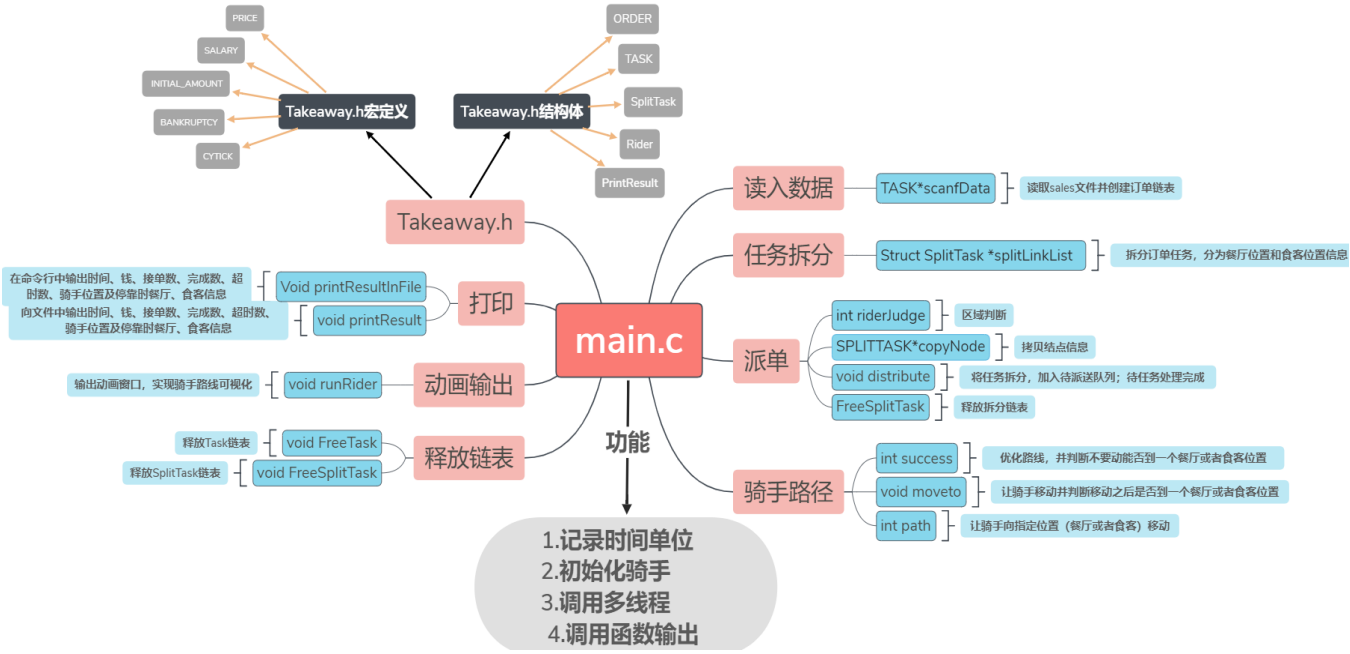

图 4-3 函数调用关系图示例

5 高层算法设计

鼠标键入的线程的伪代码：

```
DWORD WINAPI    mouseInput(SPLITTASK *head)//鼠标线程处理
{
    先将 current 指针移至待处理队列 SplitTasK 的末尾
    while (1)
    {
        等待鼠标左键按下，获取鼠标地址，贴食客和餐客的图，像素坐标转换为位置坐标      如果鼠标在房子范围内，添加任务
        if (m1.is_down())
        {
            等待鼠标左键弹起，获取鼠标地址，鼠标在房子范围内，添加任务
        }
        Sleep(20);
    }
}
```

动画输出的伪代码：

```
void animationOutput(RIDER **rider, int num, PRINTRESULT *PrintResult)
{
    创建图形对象，取图，贴图
    背景
    时间
    金钱
    已完成单数
    超时单数

//点亮餐客  食客
    for (i = 0; i < num; i++)
    {
        if (餐客)
            贴餐客图
        else
            贴食客图
    }
    //打印字符
    当前时间单位 PrintResult->time
    钱数 PrintResult->money
    已完成单数  PrintResult->compeleteCnt
超时单数  PrintResult->timeoutCnt);

for (i = 0; i < num; i++)//骑手动画
    {
        坐标转换
        贴图
    }
    for (i = 0; i < 9; i++)//释放 img 空间
        delimage(img[i]);
}
```

骑手路线规划的伪代码：

骑手链表是一个骑手结构体加一连串排好序的任务

```
while(新的任务加进来)
{
    if(新任务的餐馆位置在骑手和他正在前往的点所围成的矩形中间)
    直接插在骑手后面，优先完成新任务;
    else
    {
        while(遍历整个链表)
        {
            if(在任意两个相邻任务点 A 和 B 形成的矩形中)
            插在那两个任务中间;
            else
            放到待派送链表的末尾;
        }
    }
}
```

教师评语：