数据结构课程设计 停车场管理系统 大学编程作业（TUST 天津科技大学 2022 年）

一、项目简介

本停车场管理系统，我使用了链栈数据结构来制作，实现了简单的增删查改逻辑，计算了汽车在停车场内停留的时间和应交纳的费用，并且终端界面较为美观易用。通过这次数据结构课程设计的实践，我巩固了数据结构的知识，熟练应用了链表和栈。

这个项目是我大三写的，现在回顾已经非常粗糙，分享出来一方面希望可以帮助初学者，另一方面希望能让同学们可以从目前大学中普遍毫无价值的形式主义作业中解脱出来，更加高效地学习优质计算机知识和主流编程技术，一起发扬开源精神，感受互联网技术的美好愿景。

二、项目要求

1. 问题描述

设停车场内只有一个可停放 n 辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满 n 辆汽车，则后来的汽车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入；当停车场内某辆车要离开时，在它之后开入的车辆必须先退出车场为它让路，待该辆车开出大门外，其它车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。试为停车场编制按上述要求进行管理的模拟程序。

2. 需求分析

（1）功能要求

以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入的输入数据序列进行模拟管理。每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码及到达或离去的时刻，对每一组输入数据进行操作后的输出数据为：若是车辆到达，则输出汽车在停车场内或便道上的停车位置；若是车离去；则输出汽车在停车场内停留的时间和应交纳的费用（在便道上停留的时间不收费）。栈以顺序结构实现，队列以链表实现。

（2）测试数据
选取不同的 n，模拟停车场各种情况。
如 n = 2,输入数据为：（‘A’，1，5），（‘A’，2，10），（‘D’，1，15），（‘A’，3，20），（‘A’，4，25），（‘A’，5，30），（‘D’，2，35），（‘D’，4，40），（‘E’，0，0）。每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码及到达或离去的时刻，其中，‘A’表示到达；‘D’表示离去，‘E’表示输入结束。

3. 概要设计

（1）界面打印功能模块
定义菜单函数，把程序界面输出和数据逻辑处理在代码层面分离，使代码逻辑清晰，做的高内聚、低耦合。

（2）数据结构设计的功能模块
定义 Node 结点类、LinkStack 链栈类、LinkQueue 链队列类来表示栈和列队的数据结构。
Node 结点类的数据成员有 num 变量、time 变量，用来表示汽车牌照号、停车时刻，成员函数有 operator=(const Node& p) 运算符重载函数，用来方便后续对数据的操作。
LinkStack 链栈类的数据成员有 *top 变量、count 变量，用来表示栈顶指针、栈的容量，成员函数有 Push(int n, int t) 入栈函数、Pop() 出栈函数、Top() 栈顶指针函数、StackLength() 返回栈的长度的函数、Exist(int n) 判断栈内是否存在元素 n 的函数，用来进行入栈操作、出栈操作、返回栈顶指针操作、返回栈的长度的操作、判断栈内是否存在元素 n 的操作。
LinkQueue 链队列类的数据成员有 *front 变量、*rear 变量、count 变量，用来表示队头指针、队尾指针、队列的容量，成员函数有 Enter(int n, int t) 入队函数、Delete() 出队函数、Front() 列队队头指针函数、QueueLength() 返回列队的长度的函数、Exist(int n) 判断列队内是否存在元素 n 的函数，用来进行入队操作、出队操作、返回列队队头指针操作、返回列队的长度的操作、判断列队内是否存在元素 n 的操作。

（3）管理函数设计的功能模块
定义 park 对象、park_temp 对象、sidewalk 对象来表示停车场、临时停车场、便道。
通过 while 循环结构、switch 选择结构来实现程序的逻辑判断和功能交互，实现了停车场管理

系统的基本功能，并进行了防御式编程以应对输入错误数据，提高了程序的健壮性。

三、项目源码

程序设计思路：

1. 界面打印
2. 数据结构的设计，定义类表示栈和列队
3. 管理函数的设计，实现停车场管理

```
/*

停车场管理系统 数据结构课程设计

一、问题描述：
设停车场内只有一个可停放n辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，
依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满n辆汽车，
则后来的汽车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入；当停车场内某辆车要离开时，
在它之后开入的车辆必须先退出车场为它让路，待该辆车开出大门外，其它车辆再按原次序进入车场，
每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。试为停车场编制按上述要求进行管理的模拟程序
二、基本要求：
以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入的输入数据序列进行模拟管理。
每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码及到达或离去的时刻，
对每一组输入数据进行操作后的输出数据为：若是车辆到达，则输出汽车在停车场内或便道上的停车位置；若是车离去；
则输出汽车在停车场内停留的时间和应交纳的费用（在便道上停留的时间不收费）。栈以顺序结构实现，队列以链表实现。
三、测试数据：
选取不同的n，模拟停车场各种情况。
如n=2,输入数据为：
A 1 5
A 2 10
D 1 15
A 3 20
A 4 25
A 5 30
D 2 35
D 4 40
E 0 0
每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码及到达或离去的时刻，
其中，‘A’表示到达；‘D’表示离去，‘E’表示输入结束。

*/

/*

    程序设计思路

    1.界面打印
    2.数据结构的设计，定义类表示栈和列队
    3.管理函数的设计，实现停车场管理

*/

#include <iostream>  // 包含 I/O 流库 iostream
```

```
using namespace std;  // 加载命名空间 std

/* 声明全局变量 */
int MAX_SIZE = 0;         // 停车场的容量
int PRICE_HOUR = 0;  // 每小时停车价格
char TEMP_STATUS;   // 停车状态
int TEMP_NUM;           //汽车牌照号
int TEMP_TIME;           // 停车时刻

/* 1.界面打印 */
/* 声明菜单函数 */
void Menu();

/* 2.数据结构的设计，定义类表示栈和列队 */
/* 声明结点类 */
class Node;

/* 声明链栈类 */
class LinkStack;

/* 声明链队列类 */
class LinkQueue;

/*     3.管理函数的设计，实现停车场管理 */
/* 声明停车场管理函数 */
void Manage();

/*主函数*/
int main()
{
    /* 打印菜单 */
    Menu();

    /* 实现功能 */
    Manage();

    return 0;
}

/*     1.界面打印 */
/* 定义菜单函数 */
void Menu()
{
    system("color 0b");//修改字体颜色，0表示背景为黑色，b表示字体为淡浅绿色
```

```
    cout << "欢迎使用停车场管理系统！(^▽^ )" << endl << endl;

    cout << "停车场车辆信息输入提示：" << endl << "\t每行表示一组输入数据，由三项内容构成：\n\t(1) 一个大
}

/* 2.数据结构的设计，定义类表示栈和列队 */
/* 定义结点类 */
class Node
{
private:
    int num;  // 汽车牌照号
    int time;  // 停车时刻
    Node* next;

public:
    /* 声明类构造函数 */
    Node();
    Node(int n, int t);

    /* 声明类析构函数 */
    ~Node();

    /* 声明运算符重载函数 */
    Node operator=(const Node& p);

    /* 声明友元类 */
    /* 友元类有权访问类的所有私有（private）成员和保护（protected）成员 */
    friend class LinkStack;
    friend class LinkQueue;

    /* 声明友元函数 */
    /* 友元函数有权访问类的所有私有（private）成员和保护（protected）成员 */
    friend void Manage();
};

/* 定义类构造函数 */
Node::Node()
{
    num = 0;
    time = 0;
    next = NULL;
}
Node::Node(int n, int t)
```

```
{
    num = n;
    time = t;
    next = NULL;
}

/* 定义类析构函数 */
Node::~Node()
{

}

/* 定义运算符重载函数 */
Node Node::operator=(Node const & p)
{
    this->num = p.num;
    this->time = p.time;
    this->next = p.next;

    return *this;
}

/* 定义链栈类 */
class LinkStack
{
private:
    Node* top;  // 栈顶指针
    int count;  // 栈的容量

public:
    /* 声明类构造函数 */
    LinkStack();

    /* 声明类析构函数 */
    ~LinkStack(void);

    /*  声明入栈函数 */
    void Push(int n, int t);  // 头插法

    /* 声明出栈函数 */
    void Pop();

    /* 声明栈顶指针函数 */
    Node Top();
```

```
    /*  声明返回栈的长度的函数 */
    int StackLength();

    /* 声明判断栈内是否存在元素 n 的函数 */
    bool Exist(int n);

    /* 声明友元函数 */
    /* 友元函数有权访问类的所有私有 (private) 成员和保护 (protected) 成员 */
    friend void Manage();
};

/* 定义类构造函数 */
LinkStack::LinkStack()
{
    top = new Node;
    count = 0;
}

/* 定义类析构函数 */
LinkStack::~LinkStack(void)
{

}

/*  定义入栈函数 */
void LinkStack::Push(int n, int t)  // 头插法
{
    Node* s = new Node(n, t);

    s->next = top->next;  // 把当前的栈顶元素赋值给新结点的直接后继
    top->next = s;              // 将新的结点 s 赋值给栈顶指针

    count++;
}

/* 定义出栈函数 */
void LinkStack::Pop()
{
    if (count == 0)

        return;

    Node* p = top->next;                    // 将栈顶结点的下一个结点赋值给 p
```

```
    top->next = top->next->next;  // 使得栈顶指针的下一个指针下移一位，指向后一结点
    delete p;                                          // 释放结点 p
    count--;
}

/* 定义栈顶指针函数 */
Node LinkStack::Top()
{
    Node p;

    if (count == 0)

        return p;

    p = *(top->next);

    return p;
}

/* 定义返回栈的长度的函数 */
int LinkStack::StackLength()
{
    return count;
}

/* 定义判断栈内是否存在元素 n 的函数 */
bool LinkStack::Exist(int n)
{
    Node* p = top->next;

    while (p)
    {
        if (p->num == n)

            return true;

        p = p->next;
    }

    return false;
}

/* 定义链队列类 */
class LinkQueue
```

```
{
private:
    Node* front;  // 队头指针
    Node* rear;   // 队尾指针
    int count;  // 队列的容量

public:
    /* 声明类构造函数 */
    LinkQueue();

    /* 声明类析构函数 */
    ~LinkQueue(void);

    /* 声明入队函数 */
    void Enter(int n, int t);

    /* 声明出队函数 */
    void Delete();

    /* 声明列队队头指针函数 */
    Node Front();

    /* 声明返回列队的长度的函数 */
    int QueueLength();

    /* 声明判断列队内是否存在元素 n 的函数 */
    bool Exist(int n);

    /* 声明友元函数 */
    /* 友元函数有权访问类的所有私有 (private) 成员和保护 (protected) 成员 */
    friend void Manage();
};

/* 定义类构造函数 */
LinkQueue::LinkQueue()
{
    rear = front = new Node;
    count = 0;
}

/* 定义类析构函数 */
LinkQueue::~LinkQueue(void)
{
```

```
}

/* 定义入队函数 */
void LinkQueue::Enter(int n, int t)
{
    Node* s = new Node(n, t);

    rear->next = s;  // 把新结点 s 赋值给原队尾结点的后继
    rear = rear->next;

    count++;
}

/* 定义出队函数 */
void LinkQueue::Delete()
{
    if (count == 0)

        return;

    Node* p = front->next;  // 将欲删除的队头结点暂存给 p
    front->next = front->next->next;  // 将原队头结点的后继赋值给头结点后继
    delete p;
    count--;
}

/* 定义列队队头指针函数 */
Node LinkQueue::Front()
{
    Node p;

    if (count == 0)

        return p;

    p = *(front->next);

    return p;
}

/* 定义返回列队的长度的函数 */
int LinkQueue::QueueLength()
{
    return count;
```

```
}

/* 定义判断列队内是否存在元素 n 的函数 */
bool LinkQueue::Exist(int n)
{
    Node* p = front->next;

    while (p)
    {
        if (p->num == n)

            return true;

        p = p->next;
    }

    return false;
}

/*     3.管理函数的设计，实现停车场管理 */
/* 定义停车场管理函数 */
void Manage()
{
    LinkStack park; //停车场
    LinkStack park_temp; //临时停车场
    LinkQueue sidewalk; //便道

    cout << "请输入停车场的容量: " ;
    cin >> MAX_SIZE;
    cout << endl;
    cout << "请输入每小时停车价格: ";
    cin >> PRICE_HOUR;
    cout << endl;
    cout << "请输入停车场车辆信息: " << endl;
    cout << endl;

    while (cin >> TEMP_STATUS && TEMP_STATUS != 'E')
    {
        int time = 0;
        int flag = 0;

        cin >> TEMP_NUM >> TEMP_TIME;

        switch (TEMP_STATUS)
```

```
        {
        case 'A':  // 车辆进入
            if (park.StackLength() < MAX_SIZE)  // 进入停车场
            {
                park.Push(TEMP_NUM, TEMP_TIME);
                cout << "汽车" << TEMP_NUM << "停靠在停车场" << park.StackLength() << "号位置。" <<
            }

            else //进入便道
            {
                sidewalk.Enter(TEMP_NUM, TEMP_TIME);
                cout << "汽车" << TEMP_NUM << "停靠在便道的" << sidewalk.QueueLength() << "号位置。"
            }

            break;

        case 'D':  // 车辆离开
            if (!park.Exist(TEMP_NUM))  // 该车不在停车场中
            {
                cout << "汽车" << TEMP_NUM << "不在停车场。" << endl << endl;

                while (sidewalk.Front().num != TEMP_NUM && sidewalk.Exist(TEMP_NUM))
                {
                    sidewalk.Enter(sidewalk.Front().num, sidewalk.Front().time);
                    sidewalk.Delete();
                }

                if (sidewalk.Front().num == TEMP_NUM)
                {
                    sidewalk.Enter(sidewalk.Front().num, sidewalk.Front().time);
                    sidewalk.Delete();
                }
            }

            else //该车在停车场中
            {
                while (park.Exist(TEMP_NUM) && park.StackLength() != 0)
                {
                    if (park.Top().num != TEMP_NUM)
                    {
                        park_temp.Push(park.Top().num, park.Top().time);  // 出 park 栈, 入 park_te
                        park.Pop();
                    }
```

```
                else
                {
                    time = TEMP_TIME - park.Top().time;
                    cout << "汽车" << park.Top().num << "停车" << time << "小时，需缴纳停车费" <
                    park.Pop();
                    flag = 1;

                    break;
                }
            }

            while (park_temp.StackLength() != 0)    //压回 park 栈
            {
                park.Push(park_temp.Top().num, park_temp.Top().time);
                park_temp.Pop();
            }

            if (sidewalk.QueueLength() != 0 && flag == 1)  // 便道汽车进停车场
            {
                park.Push(sidewalk.Front().num, TEMP_TIME);  // 进入停车场
                sidewalk.Delete();                                                    // 便道上离开
                cout << "汽车" << park.Top().num << "停靠在停车场" << park.StackLength() << "号
            }
        }

        break;

    default:
        cout << endl;
        cout << "输入格式错误，请重新输入：" << endl << endl;

        break;
    }
}

cout << "程序已结束！" << endl;
}
```

四、交流学习

互联网开源精神需要大家一起互相交流学习，互相支持奉献。欢迎大家与我友好交流。

加我 QQ 好友获取所有项目源码和项目文档，感谢大家的支持！