# 北京邮电大学软件学院

# 2019-2020学年第1学期实验报告

**课程名称： 算法与数据结构**

**实验名称： 栈、队列与递归算法设计**

**实验完成人：**

**姓名：**\_\_\_刘广田\_\_\_**学号：**\_\_2018212046\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_贾红娓\_\_\_\_\_\_\_\_\_\_李璐路\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2019 年 10 月 20 日**

1. **实验目的**

仅仅认识到栈和队列是两种特殊的线性表是远远不够的，本次实验的目的在于使学生深入了解栈和队列的特征，以便在实际问题背景下灵活运用它们；同时还将巩固这两种结构的构造方法，接触较复杂问题的递归算法设计。

1. **实验内容**
2. 数制转换问题

实现将十进制数转换为八进制数

1. 括号匹配的检验

实现对圆括号、方括号的匹配检验

1. 停车场管理问题

实现模拟一个长度为10的停车场和一个便道，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），若车场内空间已被停满，则后来的汽车只能在门外的便道上等候，一旦有车开走，首先根据车的类型即占地面积判断其能进入停车场，如果类型合适，则排在便道上的第一辆车即可开入，如果类型不合适，则仍需等待；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，其中不包括在便道上的时间。

1. **实验环境**

Visual Studio 2019 Community Version

1. **实验结果**
2. 数制转换问题：

实现了将用户输入的十进制数转换为八进制数

1. 括号匹配的检验

实现了对用户输入的圆括号、方括号串的匹配检验

1. 停车场管理问题

实现了模拟一个长度为10的停车场和一个便道，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），若车场内空间已被停满，则后来的汽车只能在门外的便道上等候，一旦有车开走，首先根据车的类型即占地面积判断其能进入停车场，如果类型合适，则排在便道上的第一辆车即可开入，如果类型不合适，则仍需等待；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，其中不包括在便道上的时间。

1. **附录**

（附上实验文档，如：问题分析、设计方案、算法、设计图、程序、仿真结果、运行结果、调试心得等，具体内容根据实验要求来

问题分析和设计方案：

1. 数制转换问题：

首先读取用户输入的十进制数，使用除8取余法，将余数挨个存储在数组中，输出时将数组逆序输出即可得到对应的八进制数

1. 括号匹配的检验：

将括号的匹配问题转换为等差数列等差中项问题，定义函数使四种括号转化为对应的数字。

首先读取用户输入的圆括号、方括号表达式串，判断表达式串中括号的个数，若括号个数为奇数则直接输出不匹配，若个数为偶数则继续判断，从左到右依次读取括号，如果括号为“(”或“[”则直接压栈，如果括号为“)”或“]”则判断栈顶元素经函数转化后的值与当前括号经函数转化后的值和是否满足等差中项条件，如果满足则继续扫描下一个括号，如果不满足则直接返回不匹配。如果扫描能完整进行，则返回匹配。

1. 停车场管理问题：

首先将停车场实现为有限空间的顺序栈结构，将便道实现为链式队列结构，将汽车的信息封装在Car类实例中，在main函数中实现基本的管理逻辑，在各个类中实现对应操作以及输出信息。具体为：

Stack类：

实现可扩容（在本例中只需在main函数中控制即不会触发扩容）的模板类顺序栈结构，栈中定义栈中所存放元素的数据项data，栈当前大小的数据项size，以及反映本例中不同车型大小的数据项pointer，以及栈应有的pop，push等操作。

Parkinglot类：（停车场类）

使用Stack类的实例指针作为公有的数据成员，并实现当有车进入停车场时，车位置信息的输出（位置指的是车头位置），当有车离开停车场时，车的停放时间以及需缴纳费用的输出。

Queue类：（便道类）

实现了便道上等待车辆的存储，以及队列应有的enQueue，deQueue操作

main：

实现停车场管理的基本逻辑

算法：

1. 数制转换问题：

除8取余法得到的余数依次存储在数组中，输出时将数组元素逆序输出。

1. 括号匹配的检验：

主要思想是将括号的匹配转化为等差中项条件的检验

先检查括号串中括号的个数，若是奇数则直接返回不匹配，是偶数则进行下面的判断，遍历括号串，遇到左圆括号、左方括号则直接入栈，遇到右圆括号、右方括号则检验栈顶元素和当前元素是否满足等差中项条件，不满足则直接返回不匹配，若遍历能完整进行，则返回匹配。

1. 停车场管理问题：

在main中实现基本的停车场管理逻辑，在Parkinglot中实现对应的输出函数，都是逻辑判断，并无专门的主体算法。

程序：

1. 数制转换问题：

#include <iostream>

void conversion(int a);

int main()

{

int a;

std::cout << "请输入一个非负十进制数：";

std::cin >> a;

conversion(a);

return 0;

}

void conversion(int a) {

int times = 0;

int outCome = 0;

int temp = a;

if (a > 7) {

while (temp != 0) {

outCome = temp / 8;

temp /= 8;

times++;

}

int\* arr = new int[times];

for (int i = 1; i <= times; i++) {

int out = a % 8;

a /= 8;

arr[times - i] = out;

}

std::cout << "对应的八进制数为：";

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

std::cout << arr[i];

delete[] arr;

}

else {

std::cout << "对应的八进制数为：" << a;

}

}

1. 括号匹配的检验：

#include <iostream>

#include <string>

#include <stack>

bool match(std::string s);

int convert(char c);

int main()

{

std::cout << "请输入中括号和小括号组成的表达式：";

std::string s;

std::cin >> s;

if (match(s))

std::cout << "匹配" << std::endl;

else

std::cout << "此串括号匹配不合法" << std::endl;

return 0;

}

int convert(char c) {

int numOfBracket;

switch (c) {

case '[':

numOfBracket = 0;

break;

case '(':

numOfBracket = 1;

break;

case ')':

numOfBracket = 2;

break;

case ']':

numOfBracket = 3;

break;

default:

numOfBracket = -1;

break;

}

return numOfBracket;

}

bool match(std::string s) {

int isMatch = false;

if (s.length() % 2 != 0)

return isMatch;

std::stack<char> chars;

for (int i = 0; i < s.length(); i++) {

char c = s.c\_str()[i];

if (c == '(' || c == '[')

chars.push(c);

else if (c == ')' || c == ']') {

if (convert(c) + convert(chars.top()) != 3)

return isMatch;

else

chars.pop();

}

}

if (chars.empty())

isMatch = true;

return isMatch;

}

1. 停车场管理问题：

Car.h

#pragma once

#include <iostream>

#include <string>

#include <sstream>

class Car {

private:

char flag;

int time;

int number;

double carSize;

public:

Car();

Car(char flag, int time, int number, double carSize);

char getFlag();

int getTime();

void setTime(int time);

int getNumber();

double getCarSize();

std::string getCarType();

std::string toString();

};

Car.cpp

#include "Car.h"

Car::Car() {

flag = 'A';

time = 0;

number = 1;

carSize = 1;

}

Car::Car(char flag, int number, int time, double carSize) {

this->flag = flag;

this->number = number;

this->time = time;

this->carSize = carSize;

}

char Car::getFlag() { return flag; }

int Car::getTime() { return time; }

void Car::setTime(int time) {

this->time = time;

}

int Car::getNumber() { return number; }

double Car::getCarSize() { return carSize; }

std::string Car::getCarType() {

std::string type;

if (carSize == 1)

type = "小汽车";

else if (carSize == 1.5)

type = "客车";

else if (carSize == 3)

type = "十轮卡车";

return type;

}

std::string Car::toString() {

std::string flagString(1, flag);

return ("车的状态为：" + flagString + "车的序号为：" + std::to\_string(number) + "车的到达时间为：" + std::to\_string(time));

}

Queue.h

#pragma once

#include <iostream>

#include "Car.h"

class Node {

public:

Car car;

Node\* next;

};

class Queue {

private:

Node \*front;

Node \*rear;

public:

Queue();

~Queue();

bool empty();

void enQueue(Car c);

void deQueue();

Car getFrontCar();

void showQueue();

int getSize();

};

Queue.cpp

#include "Queue.h"

Queue::Queue() {

Node\* p = new Node;

p->next = NULL;

front = p;

rear = p;

}

Queue::~Queue() {

while (front) {

rear = front->next;

delete front;

front = rear;

}

}

void Queue::enQueue(Car c) {

Node\* pNew = new Node;

pNew->car = c;

rear->next = pNew;

rear = pNew;

}

void Queue::deQueue() {

Node\* p = front->next;

front->next = front->next->next;

if (rear == p)

rear = front;

delete p;

}

Car Queue::getFrontCar() {

return front->car;

}

bool Queue::empty() {

return (front == rear);

}

void Queue::showQueue() {

Node\* p = front->next;

if (p == NULL)

std::cout << "队列为空" << std::endl;

while (p != NULL) {

std::cout << p->car.toString();

p = p->next;

}

std::cout << std::endl;

}

int Queue::getSize() {

int length = 0;

Node\* p1 = front;

while (p1 != rear) {

p1 = p1->next;

length++;

}

p1 = NULL;

return length;

}

Stack.h

#pragma once

#include <iostream>

constexpr auto MAX\_SIZE = 10;

constexpr auto INCRESEMENT = 10;

template <typename T>

class Stack {

private:

T data;

T\* base;

int size;

double pointer;

public:

Stack();

~Stack();

T pop();

void push(T data);

bool empty();

T peek();

int getSize();

double getPointer();

void enPointer(double growth);

void dePointer(double down);

void show();

};

template <typename T>

Stack<T>::Stack() {

base = new T[MAX\_SIZE];

data = T();

size = 0;

pointer = 0;

}

template <typename T>

Stack<T>::~Stack() {

delete base;

base = NULL;

}

template <typename T>

T Stack<T>::pop() {

T ret = base[size - 1];

size--;

return ret;

}

template <typename T>

void Stack<T>::push(T data) {

double tmpSize = size++;

if (tmpSize > MAX\_SIZE) {

T\* tmp;

tmp = new T[MAX\_SIZE + INCRESEMENT];

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

tmp[i] = base[i];

delete base;

base = tmp;

tmp = NULL;

}

size--;

base[size] = data;

size++;

}

template <typename T>

bool Stack<T>::empty() {

return (size == 0);

}

template <typename T>

T Stack<T>::peek() {

return (base[(size)-1]);

}

template <typename T>

int Stack<T>::getSize() {

return this->size;

}

template <typename T>

double Stack<T>::getPointer() {

return pointer;

}

template <typename T>

void Stack<T>::dePointer(double down) {

pointer -= down;

}

template <typename T>

void Stack<T>::enPointer(double growth) {

pointer += growth;

}

template <typename T>

void Stack<T>::show() {

for (int i = 0; i < this->size; i++) {

std::cout << this->base[i] << " ";

}

}

Parkinglot.h

#pragma once

#include <iostream>

#include "Car.h"

#include "Stack.h"

class Parkinglot {

public:

Stack<Car>\* pStack;

Parkinglot();

~Parkinglot();

bool isEmpty();

void showArrived(Car c);

void showLeaved(Car c);

};

Parkinglot.cpp

#include "Parkinglot.h"

Parkinglot::Parkinglot() {

Stack<Car>\* pStack = new Stack<Car>;

this->pStack = pStack;

pStack = NULL;

}

Parkinglot::~Parkinglot() {

delete pStack;

pStack = NULL;

}

bool Parkinglot::isEmpty() {

return (pStack->empty());

}

void Parkinglot::showArrived(Car c) {

std::cout << "车在停车场中的位置为：" << pStack->getPointer() << std::endl;

pStack->push(c);

pStack->enPointer(c.getCarSize());

}

void Parkinglot::showLeaved(Car c) {

Stack<Car> tmpStack;

while (pStack->peek().getNumber() != c.getNumber()) {

Car tmpCar = pStack->pop();

pStack->dePointer(tmpCar.getCarSize());

tmpStack.push(tmpCar);

}

Car leavedCar = pStack->pop();

pStack->dePointer(leavedCar.getCarSize());

int lastTime = (c.getTime() - leavedCar.getTime());

std::cout << "此车在停车场停留时间为：" << lastTime

<< "，车的类型是" << c.getCarType()

<< "，应缴纳的金额为：" << (lastTime \* c.getCarSize() \* 10) << std::endl;

while (!tmpStack.empty()) {

pStack->push(tmpStack.peek());

pStack->enPointer(tmpStack.peek().getCarSize());

tmpStack.pop();

}

}

#include "Car.h"

#include "Queue.h"

#include "Parkinglot.h"

#include <iostream>

int main() {

Parkinglot p;

Queue q;

char flag;

int number, time;

double size;

std::cin >> flag >> number >> time >> size;

while (flag != 'E') {

Car c(flag, number, time, size);

if (flag == 'A') {

if (p.pStack->getPointer() + size <= MAX\_SIZE)

p.showArrived(c);

else

q.enQueue(c);

}

else if (flag == 'D') {

p.showLeaved(c);

if (!q.empty()) {

if (q.getFrontCar().getCarSize() + p.pStack->getPointer() <= MAX\_SIZE) {

Car tmpCar = q.getFrontCar();

tmpCar.setTime(c.getTime());

p.showArrived(tmpCar);

}

}

}

std::cin >> flag >> number >> time >> size;

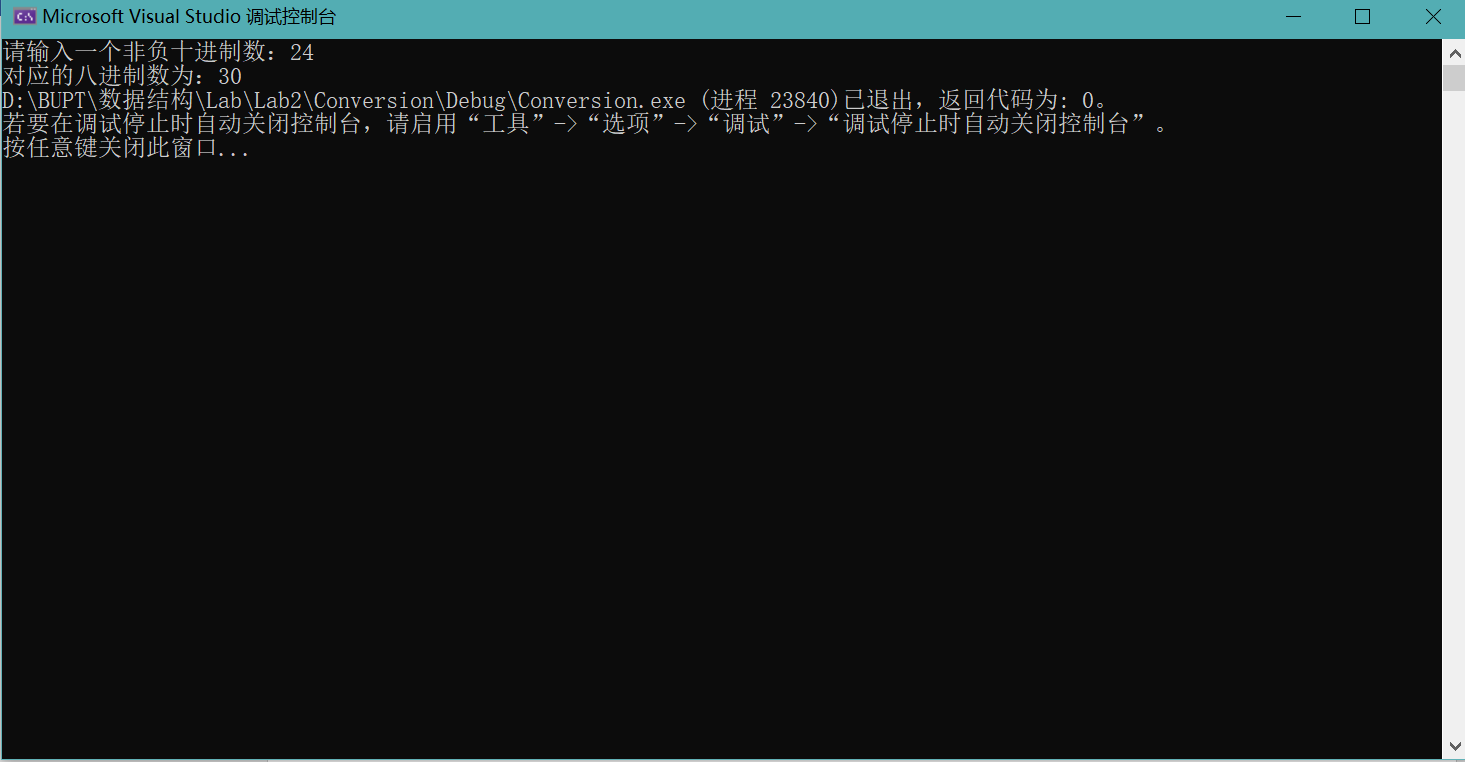
}

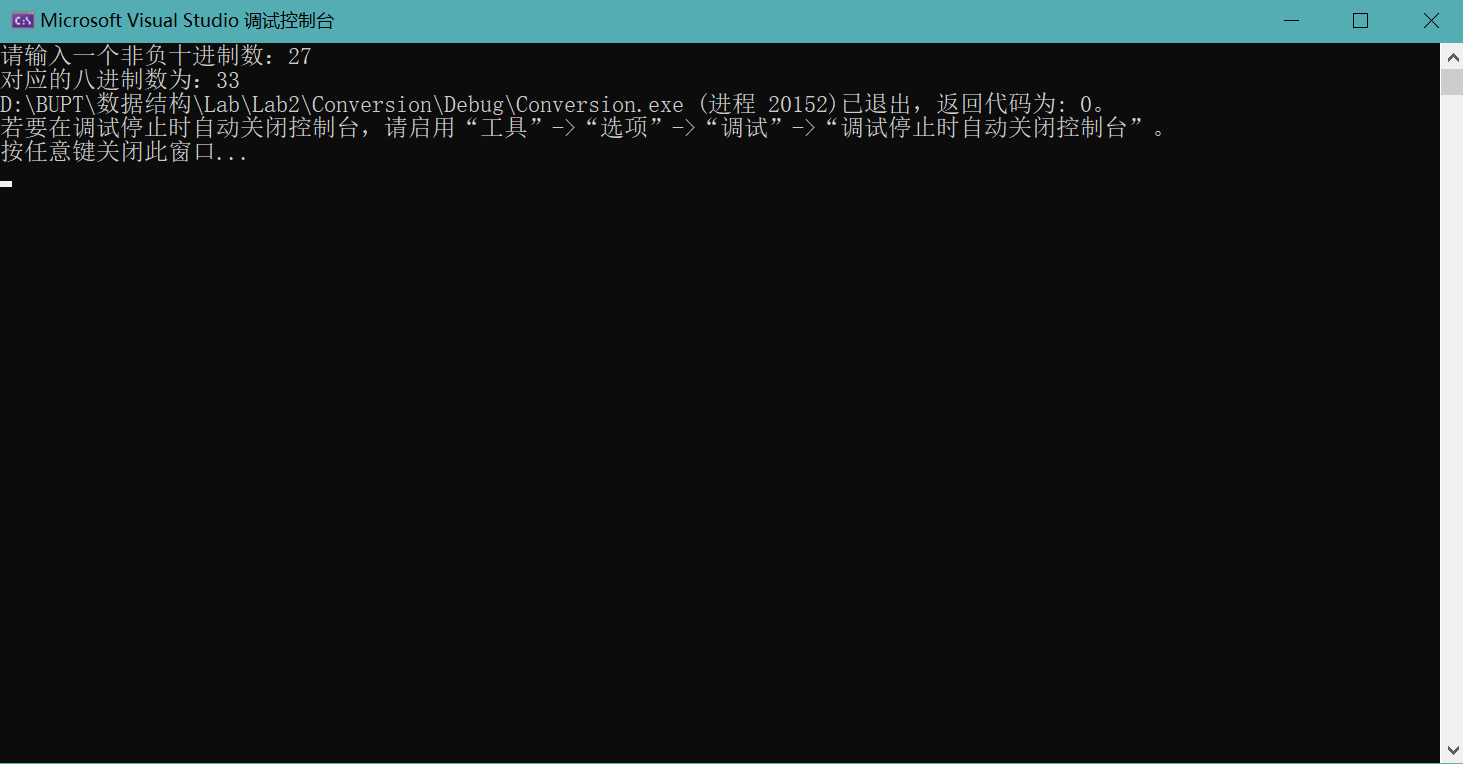
return 0;

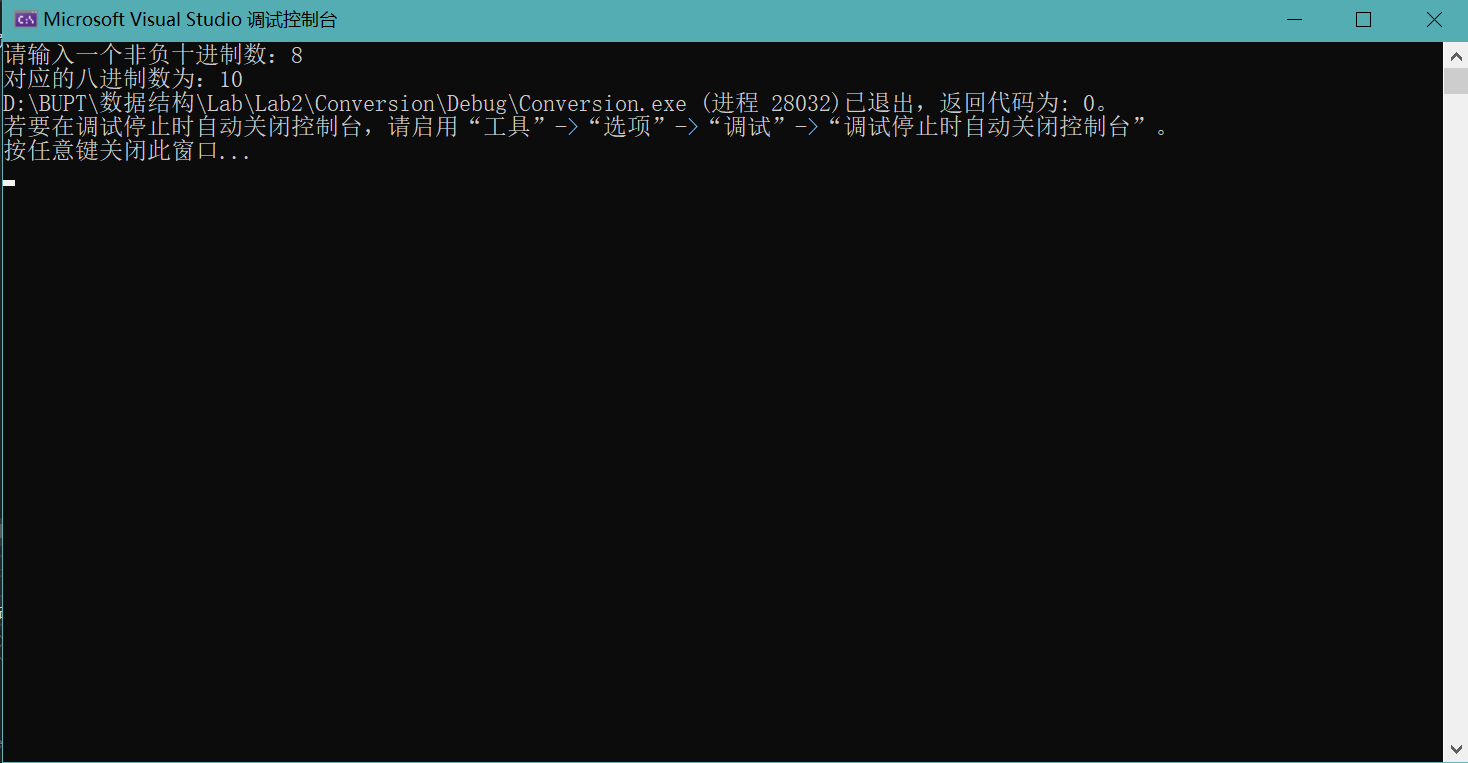
}

仿真结果和运行结果：

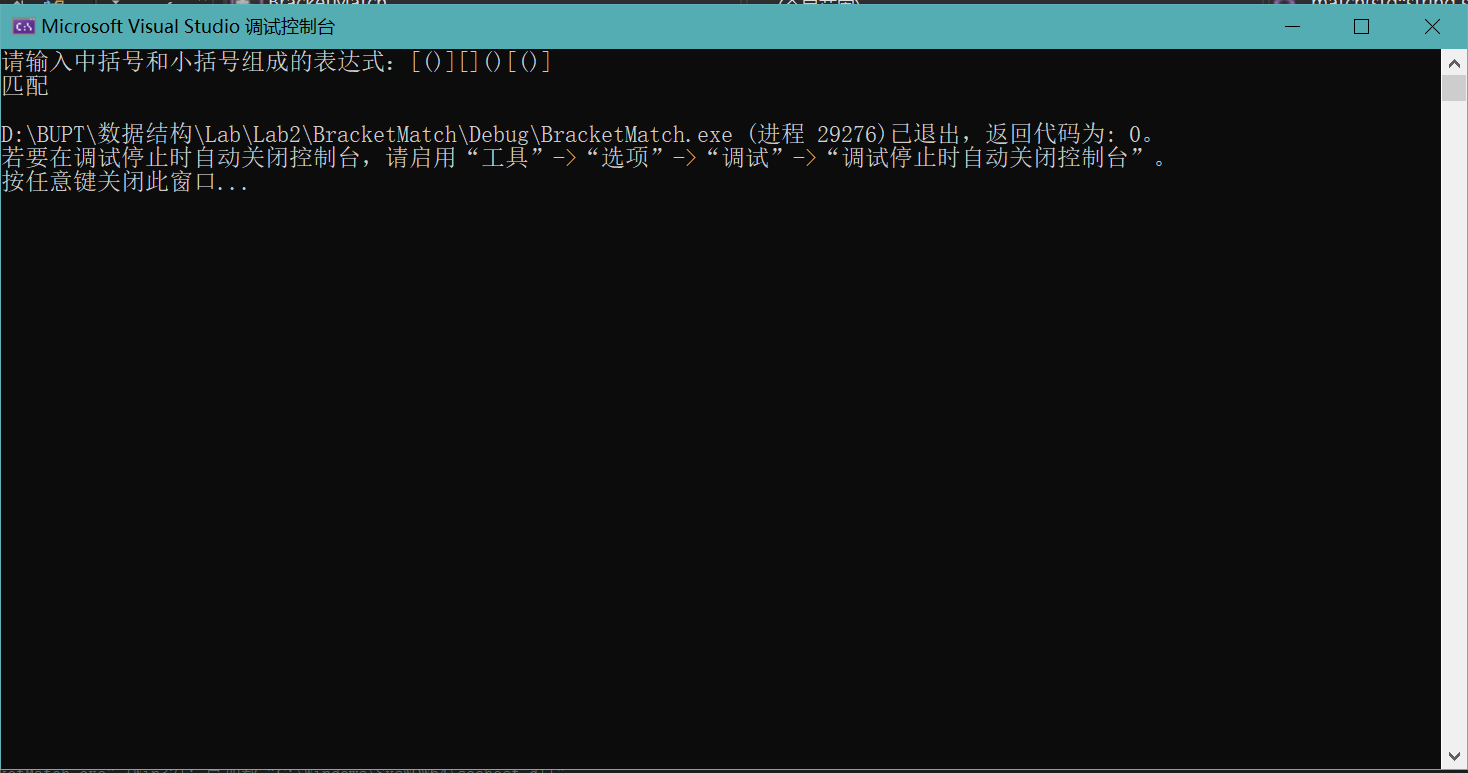
1. 数制转换问题：

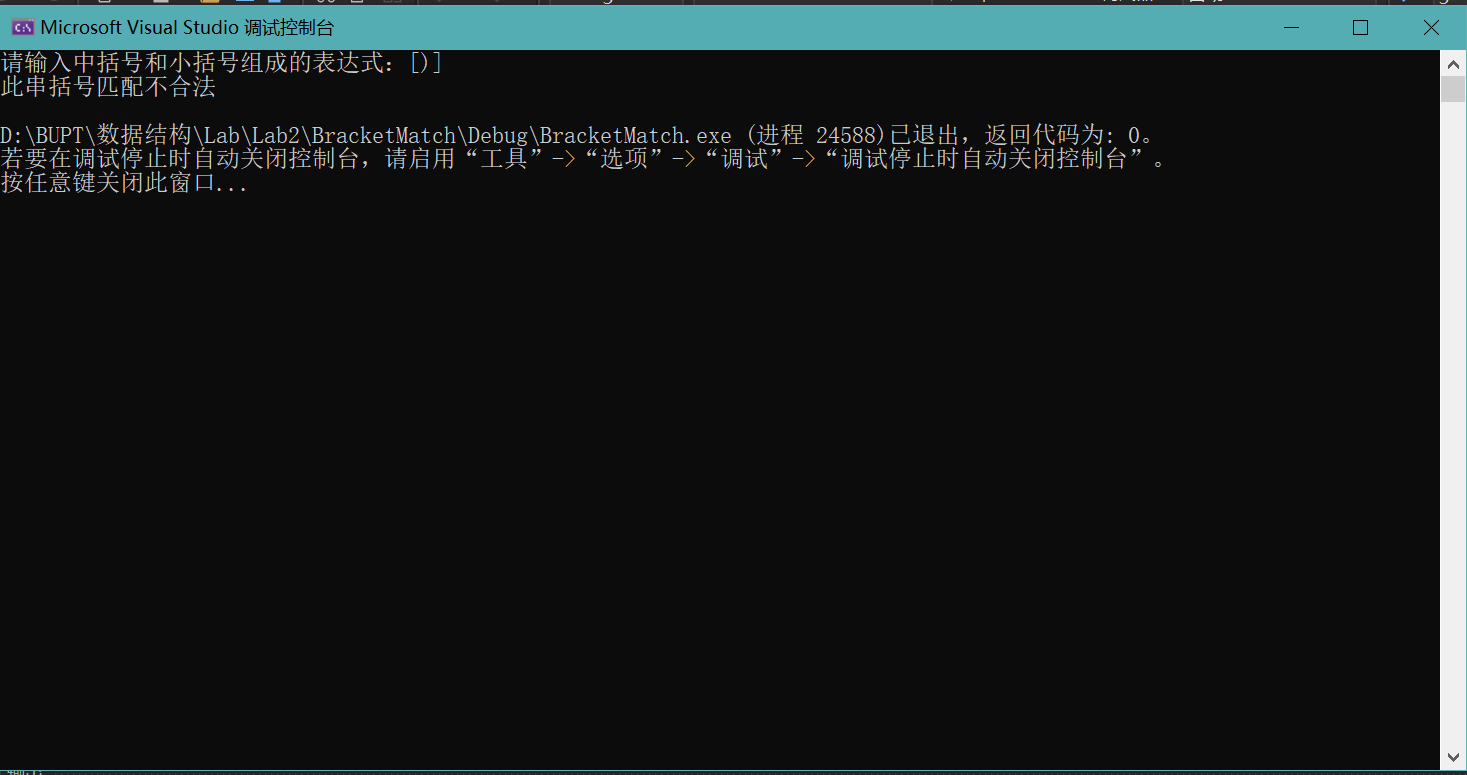


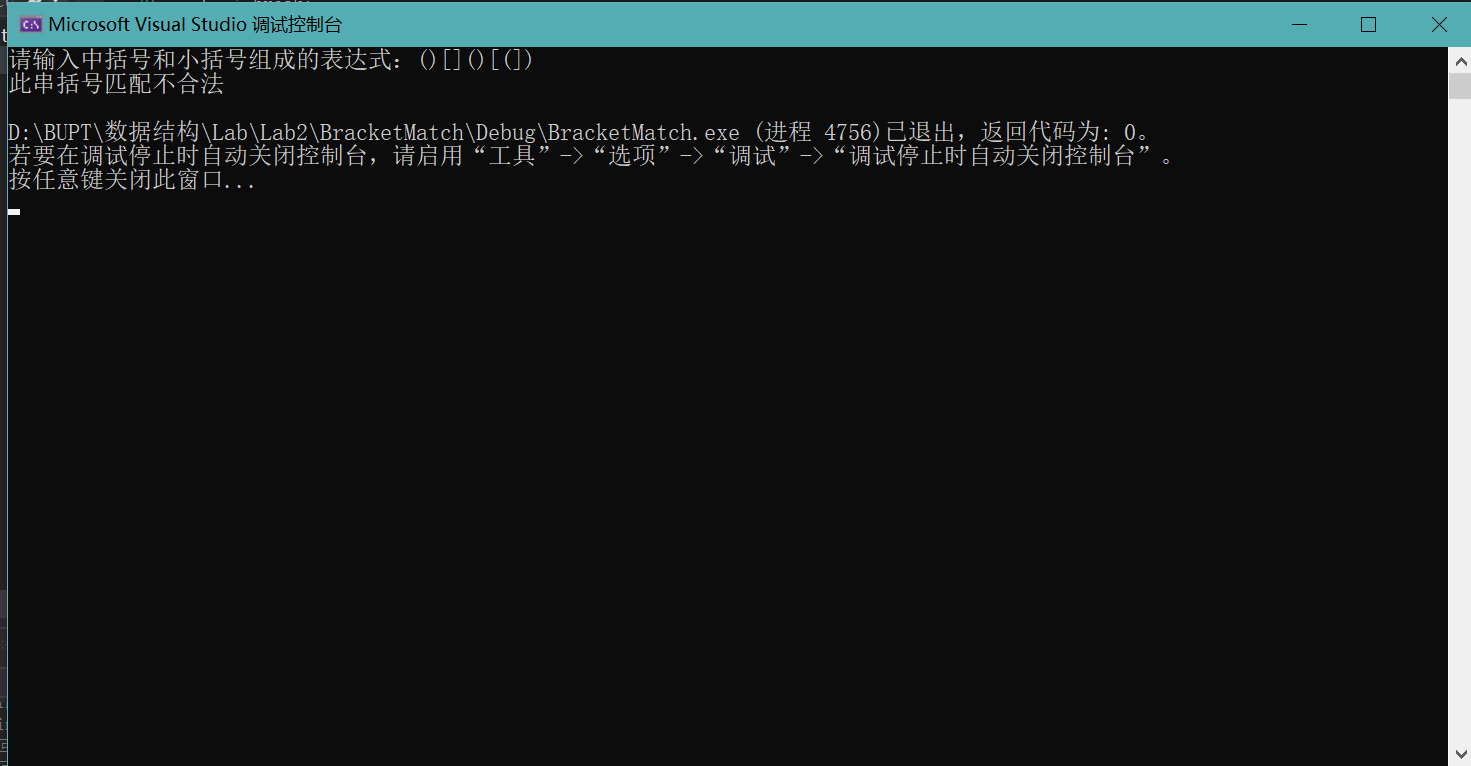




1. 括号匹配的检验：







1. 停车场管理问题：



调试心得：

1. 有些简单的逆序输出问题不一定要使用栈，直接使用数组就能解决问题
2. 解决匹配问题的主题思路应该是，一旦有某处不匹配则一定不匹配，这样可以节省很多时间，对于ASCII码不相邻的符号匹配问题，可以手动编写函数使其转化为简单的数字上的等差中项问题。
3. 对于大的项目首先要使用分治法将项目分成小块，一步步分割问题，可以先在主函数中编写框架以及需要的函数，然后分别去实现对应的函数。

处理使用到栈的问题时要注意一旦使用pop函数就会弹出栈顶元素，尤其是在判断时，可能本来不想pop只是想判断栈顶元素，但因为使用pop操作就会将栈顶元素误弹，所以要将peek函数和pop函数分开使用，pop只用来弹栈，peek只用来获取栈顶元素的状态。

类似的，还有size++操作，如果想要判断size++之后的值与某个值的关系，一定要在判断完成之后实现size--，否则size会因为用在判断中的size++而隐式加一。