**江苏科技大学**

**课程设计报告**

课程名称: 算法与数据结构课程设计

学 院: 计算机学院

班 级： 1822107101

学 号： 182210710119

姓 名： 陈 四 贵

指导老师: 刘 嘎 琼

设计时间: 2019-12-13至2019-12-16

目 录

**[一、线性表 1](#_Toc30725)**

题1 [1](#_Toc31544)

[实践目的： 1](#_Toc19886)

[实践内容： 1](#_Toc19886)

[实现代码： 1](#_Toc19886)

[实践结果： 3](#_Toc19274)

[实践小结： 4](#_Toc17931)

[题2 4](#_Toc19886)

[实践目的： 4](#_Toc19899)

[实现代码： 4](#_Toc5831)

[实践结果： 6](#_Toc892)

[实践小结： 6](#_Toc27964)

**[二、数组、栈和队列 7](#_Toc18962)**

[题3 7](#_Toc19886)

[实践目的： 7](#_Toc10306)

[实践内容： 7](#_Toc9926)

[实现代码： 7](#_Toc14960)

[实践结果： 9](#_Toc645)

[实践小结： 9](#_Toc20221)

[题4 9](#_Toc10306)

[实践目的： 10](#_Toc12702)

[实践内容： 10](#_Toc20241)

[实现代码： 10](#_Toc3686)

[实践结果： 12](#_Toc4685)

[实践小结： 13](#_Toc26128)

**[三、树 13](#_Toc7371)**

[题5 1](#_Toc10306)3

[实践目的： 13](#_Toc12607)

[实现代码： 13](#_Toc7656)

[实践结果： 15](#_Toc7697)

[实践小结： 15](#_Toc16020)

**一、线性表**

题1、建立一个单链表，显示链表中每个节点的数据，并做删除和插入处理。

**实践目的：**

掌握线性表在链式存储结构下的基本运算的实现。

**实践内容：**

1、功能

* 1. 建立以带头结点的单链表
  2. 显示链表中每个结点的数据
  3. 在单链表中指定位置插入指定数据并输出单链表中所有数据
  4. 删除单链表中指定的结点并输出单链表中所有数据

2、输入要求

输入单链表中所有数据，插入的数据元素的位置、值，要删除的数据元素的位置。

3、测试数据

单链表中所有数据：12，23，56，21，8，10，15，67，90，32

插入的数据元素的位置、值：1，28

要删除的数据元素的位置：10

**实现代码：**

#include <iostream>

using namespace std;

typedef struct Node

{

int data;

struct Node \*next;

}\*ListNode;

typedef ListNode \*LinkList;

ListNode head, p, q;

void CreateList(int \*a,int n) {

head = new Node;

head->next = NULL;

p = new Node;

head->next = p;

p->data = a[0];

p->next = NULL;

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

q = new Node;

q->data = a[i];

q->next = NULL;

p->next = q;

p = q;

}

}

void PrintList() {

p = head->next;

cout << "单链表所有数据依次为：";

while (p != NULL) {

cout << p->data << "\t";

p = p->next;

}

cout << "\n";

}

void InsertList(int location, int value) {

p = head;

int counter = 0;

while (counter != location - 1) {

p = p->next;

counter++;

}

q = new Node;

q->data = value;

q->next = p->next;

p->next = q;

p = head->next;

cout << "在第" << location << "个位置插入" << value << "后，所有数据依次为：";

while (p != NULL) {

cout << p->data << "\t";

p = p->next;

}

cout << "\n";

}

void DeleteList(int location) {

p = head;

int counter = 0;

while ((counter != location - 1) && p != NULL) {

p = p->next;

counter++;

}

p->next = p->next->next;

p = head->next;

cout << "删除第" << location << "个位置的节点后，单链表数据为：";

while (p != NULL) {

cout << p->data << "\t";

p = p->next;

}

}

int main() {

int n,location;

cout << "请输入单链表中数据的个数:";

cin >> n;

int \*a = new int[n];

cout << "请依次输入单链表中的数据:";

for (int i = 0; i < n; i++) cin >> a[i];

CreateList(a,n);

PrintList();

cout << "请输入您想插入数据元素的位置、值:";

cin >> location >> n;

InsertList(location,n );

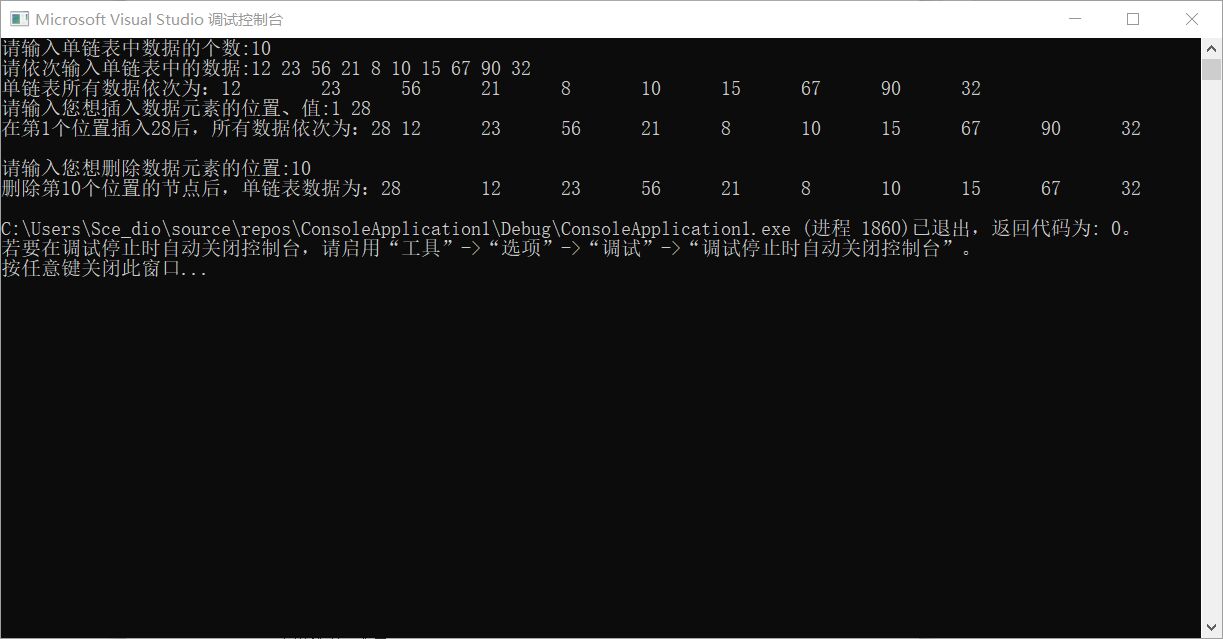
cout << "请输入您想删除数据元素的位置:";

cin >> location;

DeleteList(location);

}

**实践结果：**



**实践小结：**

1. 单链表前设头节点，可以使在第一个节点和其余的节点之前插入节点的操作一致，不用特殊处理。
2. 若要在第i个位置插入或删除节点，则应使工作指针指向第i-1个节点，因为在插入操作中，要建立被插入节点的前一个节点与插入节点的后继关系；在删除操作中，要使被删除节点的前一个节点的后继更迭为被删除节点的后一个节点。简而言之这就是建立前驱后继关系的需要。
3. typedef关键字的作用是声明一个类型的别名，而不是创建一个类型的对象。此外，C语言中常使用”typedef+结构体名+标识符”的形式声明对象。
4. 为了实现用户输入数据的多样性，即想输多少数据就输多少数据。可以定义一个变量保存用户输入数据的数量，鉴于一般数组的声明不能采用变量作为长度，可以采用int \*a=new int[n]的形式实现功能。
5. 由于单链表的随机存储、顺序存取特性，在找位置时需使用一个变量充当计数器记录位置。

题2、约瑟夫环（Joseph）问题的一种描述是：编号1，2，┉，n的n个人按顺时针方向围坐一圈，每个人持有一个密码（正整数），一开始，任选一个正整数作为报数上线值m，从第一个人开始按顺时针方向自1开始顺序报数，报到m时停止报数。报m的人出列，将他的密码作为新的m值，从他在顺时针方向上的下一个人开始重新从1开始报数，如此下去，直至所有人全部出列为止。设计一个程序求出出列顺序。

**实践目的：**

练习线性表的实际场景应用。

**实现代码：**

#include <iostream>

using namespace std;

typedef struct Node

{

int id, key;

struct Node \*next;

}\*ListNode;

typedef ListNode \*LinkList;

ListNode head, feet, p, q;

void CreateList(int n, int \*a) {

head = new Node;

head->id = 1;

head->key = a[0];

head->next = NULL;

p = head;

for (int i = 2; i <= n; i++) {

q = new Node;

q->id = i;

q->key = a[i - 1];

q->next = NULL;

p->next = q;

p = q;

}

p->next = head;

feet = p;

}

void Johnsphus(int m, int n) {

while (1 > 0) {

p = head;

q = feet;

for (int num = 1; num < m; num++) {

q = q->next;

p = p->next;

}

if (p->next == q) {

cout << p->id << "\t";

q->next=q;

cout << q->id;

delete q;

break;

}

cout << p->id << "\t";

m = p->key;

q->next = p->next;

p = q->next;

head = p;

feet = q;

}

}

void PrintList() {

p = head;

cout << "游戏者的序号\t对应密码\n";

do {

cout << p->id << "\t\t" << p->key << "\n";

p = p->next;

} while (p != head);

cout << "\n";

}

int main() {

int n, m;

cout << "请输入玩约瑟夫游戏的人数：";

cin >> n;

int \*a = new int[n];

cout << "请依次输入每个人所设置的密码:\n";

for (int i = 0; i <= n - 1; i++) {

cin >> a[i];

}

CreateList(n, a);

PrintList();

cout << "请指示一个正整数作为报数上线值:";

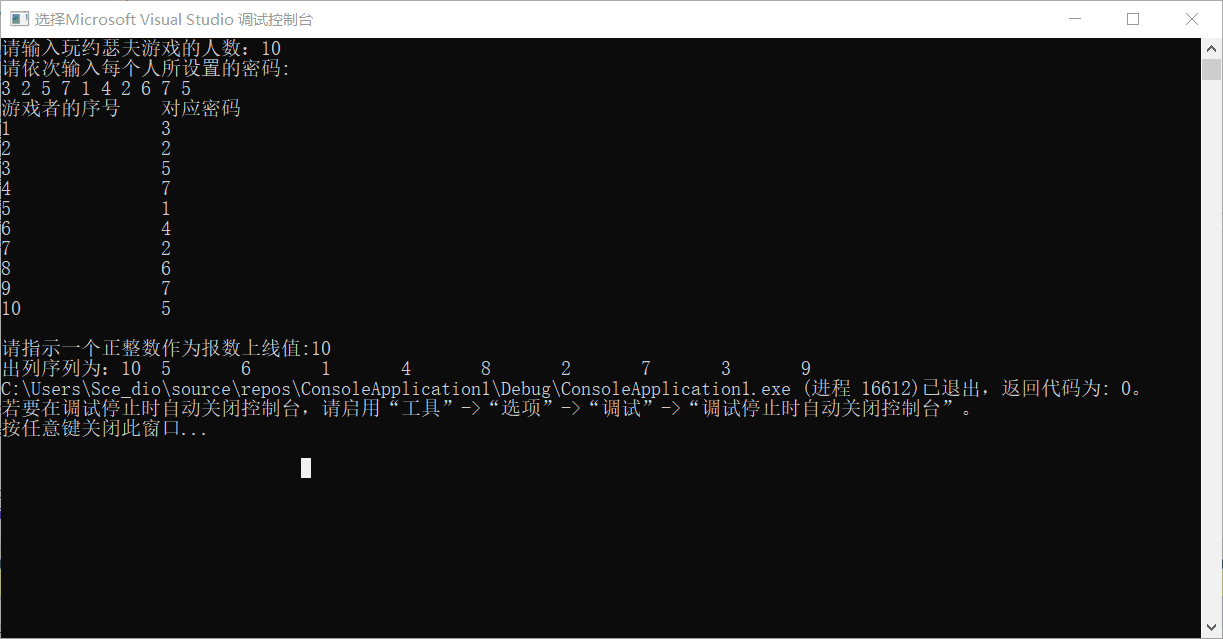
cin >> m;

cout << "出列序列为：";

Johnsphus(m, n);

}

**实践结果：**



**实践小结：**

1. 由于该游戏每个人有自己设置的密码，又需要输出出列人的序列，需围成环，无论是顺序存储还是链接存储，都应该封装一个有两个数据域和一个指针域的结构体，以构成“约瑟夫环”的数据元素。
2. 第一个设置值需另外设定，也就是说需要定义一个变量来保存第一次开始时的出列数，出列后，出列数再设置为出列人设置的密码，继续进行游戏。
3. 由于每个函数都需要使用若干工作指针，因此我们可以将工作指针声明在全局范围内，便于我们使用。但是我们在每次使用工作指针时一定要赋初值，否则就可能出现一些本不该出现的错误。
4. 约瑟夫环既然是环，那就是有头有尾，若第一个出列的就是头节点所代表的人，则需有一个指针指向头节点的前一个节点，因此，可以采用head标识头，feet标识尾。
5. 出列的一般情况：q是环尾，p是环头，q->next=p->next即可，但在出列至只剩最后两个节点时，沿用一般情况的删除节点操作是不方便的，需特殊化处理，即找到当前应删除的节点，输出它的序列号，使q->next指向q自己，再输出该节点的序列号即可。

**二、数组、栈和队列**

题3、用n×n矩阵M表示一个迷宫，0和1分别表示通路和墙壁。迷宫的入口地点下标为（1，1），出口点下标为（n，n）。试求出从入口点到达出口点的一条通

**实践目的：**

掌握递归算法在二位数组中的实际应用。

**实践内容：**

[基本要求]

利用二维数组存储迷宫，采用递归算法实现。

[测试数据]

自己设计一个8≤n≤20的数据，并把测试迷宫打印出来，程序能够求出一条这样的通路。

**实现代码：**

#include <iostream>

#include<stdlib.h>

#define m 8

#define n 8

using namespace std;

int maze[m][n] = {

{0,0,0,1,1,1,0,1},

{0,1,0,1,1,1,0,1},

{0,0,1,1,0,0,1,0},

{1,0,0,0,0,1,0,1},

{1,1,1,0,0,1,1,1},

{1,0,1,1,0,0,1,0},

{1,0,0,0,0,0,0,1},

{0,0,1,1,1,0,0,0}

};

int MAZE(int x, int y)

{

if (x<0 || x>m || y<0 || y>n) {

return 0;

}

int a, b;

int tag = 0;

if (x == 7 && y == 7)

return 1;

for (int i = 0; i < 4; i++)

{

a = x + Move[i][0];

b = y + Move[i][1];

if (maze[a][b] == 0)

{

maze[a][b] = 2;

tag = MAZE(a, b);

if (tag) {

cout<<"("<<a+1<<","<<b+1<<")←";

return 1;

}

}

}

return 0;

}

void Print(){

cout<<"迷宫打印如下（0表示通路，1表示障碍）：\n";

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

for(int j=0;j<n;j++){

cout<<maze[i][j]<<"\t";

if(j!=0&&j%(n-1)==0) cout<<endl;

}

cout<<endl;

}

int main()

{

Print();

cout<<"递归求迷宫的通路：\n";

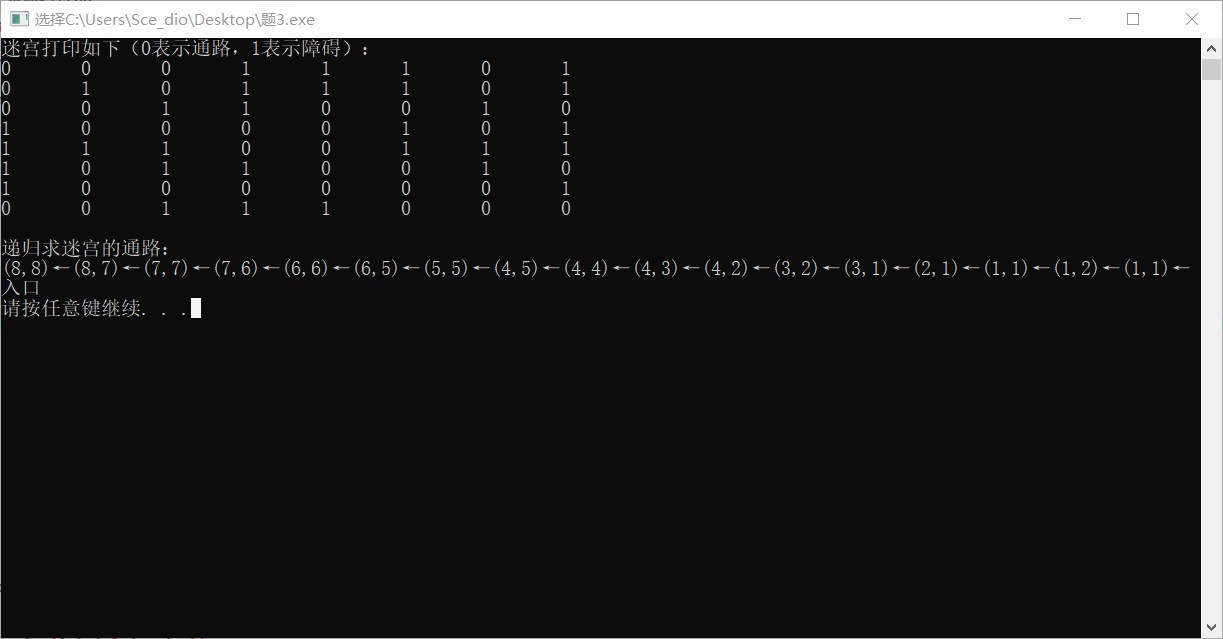
MAZE(0, 0);

cout<<"(1,1)←入口\n";

system("pause");

}

**实践结果：**



**实践小结：**

1. 由于递归过程是从最底层开始层层递归完成，而且我们只有最终到达终点才知道是否成功抵达，因此坐标输出是个逆序过程。考虑问题也是从底层开始考虑较好。
2. 应选择某点为源点，向四个方向发起试探，需要注意的是，下标越界的话不需要发起试探。如果路口通顺且没搜索过则进入该路口，否则什么都不做。为把通路与走过的路加以区别，可以将其赋为2，以示区别。
3. 因为要层层递归判断路口是否通顺，函数的返回值会是很好的判断依据。因此将bool作为递归函数的返回值类型会是很好的选择。

题4、停车场管理。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不离开停车场时必须按它停留的时间长短交费。试为停车场编制按上述要求进行管理的模拟程序。

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“离开”信息，汽车牌照号码以及到达或离去的时刻。与每一组输入数据信息相对应的输出数据为：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；若是车辆离去，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，队列以链表结构实现。（提示：需另设一个栈，临时停放为给要离去的汽车让路从停车场退出来的汽车，也用顺序存储结构实现。）

**实践目的：**

掌握栈和队列的实际应用。

**实践内容：**

［测试数据］

设n＝２，输入数据为：（‘Ａ’，１，５），（‘Ａ’，２，１０），（‘Ｄ’，１，１５），（‘Ａ’，３，２０），（‘Ａ’，４，２５），（‘Ｄ’，２，３５），（‘Ｄ’，４，４０），（‘Ｅ’，０，０）。其中：‘Ａ’表示到达，‘Ｄ’表示离去，‘Ｅ’表示输入结束。

其中：（‘Ａ’，１，５）表示1号牌照车在5这个时刻到达，而（‘Ｄ’，１，１５）表示1号牌照车在15这个时刻离去。

**实现代码：**

#include <iostream>

using namespace std;

const int n = 2;

const int m = 5;

int a, c, key1 = 0, key2 = 0, key3 = 0, i = 0, Percost = 3, LastExit;

char b, d;

struct Cars {

int ID;

char State;

int Time;

Cars \*next;

}\*p, \*q, \*r, \*head, \*near;

Cars \*Stack1[n], \*Stack2[m];

void parkingmode() {

head = new Cars;

cout << "请按时间先后顺序输入汽车的状态、汽车牌照、汽车进行该状态的时刻（格式：A 1 1）：";

cin >> b >> a >> c;

head->ID = a;

head->State = b;

head->Time = c;

p = head;

Stack1[0] = head;

key1++;

cout << "牌照为" << Stack1[key1 - 1]->ID << "的车辆停在停车场从北往南第" << key1 << "个位置。\n";

do {

i = 0;

cout << "请按时间先后顺序输入汽车的状态、汽车牌照、汽车进行该状态的时刻（格式：A 1 1）：";

cin >> b >> a >> c;

if (b == 'E') {

cout << "程序结束";

break;

}

if (b == 'A') {

q = new Cars;

q->ID = a;

q->State = b;

q->Time = c;

if (key1 - 1 < n - 1) {

key1++;

Stack1[key1 - 1] = q;

for (int j = 0; j <= key1 - 1; j++)

cout << "牌照为" << Stack1[j]->ID << "的车辆停在停车场从北往南第" << j + 1 << "个位置。\n";

}

else {

if (key3 == 0) {

head = q;

head->next = NULL;

r = head;

key3++;

}

else {

q->next = NULL;

r->next = q;

key3++;

}

for (int j = 0; j <= key1 - 1; j++)

cout << "牌照为" << Stack1[j]->ID << "的车辆停在停车场从北往南第" << j + 1 << "个位置。\n";

cout << "牌照为" << q->ID << "的车辆停在便道第" << key3 << "个位置。\n";

}

}

if (b == 'D') {

while (Stack1[i]->ID != a) {

i++;

}

for (int j = key1 - 1; j > i; j--) {

Stack2[key2++] = Stack1[j];

Stack1[j] = NULL;

}

cout << "牌照为" << Stack1[i]->ID << "的车辆停留了" << (c - Stack1[i]->Time) << "小时，应收费" << (c - Stack1[i]->Time)\*Percost << "元。\n";

for (int j = key2 - 1, tmp = i; j >= i; j--) {

Stack1[tmp++] = Stack2[j];

Stack2[j] = NULL;

key2--;}

key1--;

if (key3 == 0) { }

else {

LastExit = c;

key1++;

Stack1[key1 - 1] = head;

Stack1[key1 - 1]->Time = LastExit;

head = head->next;

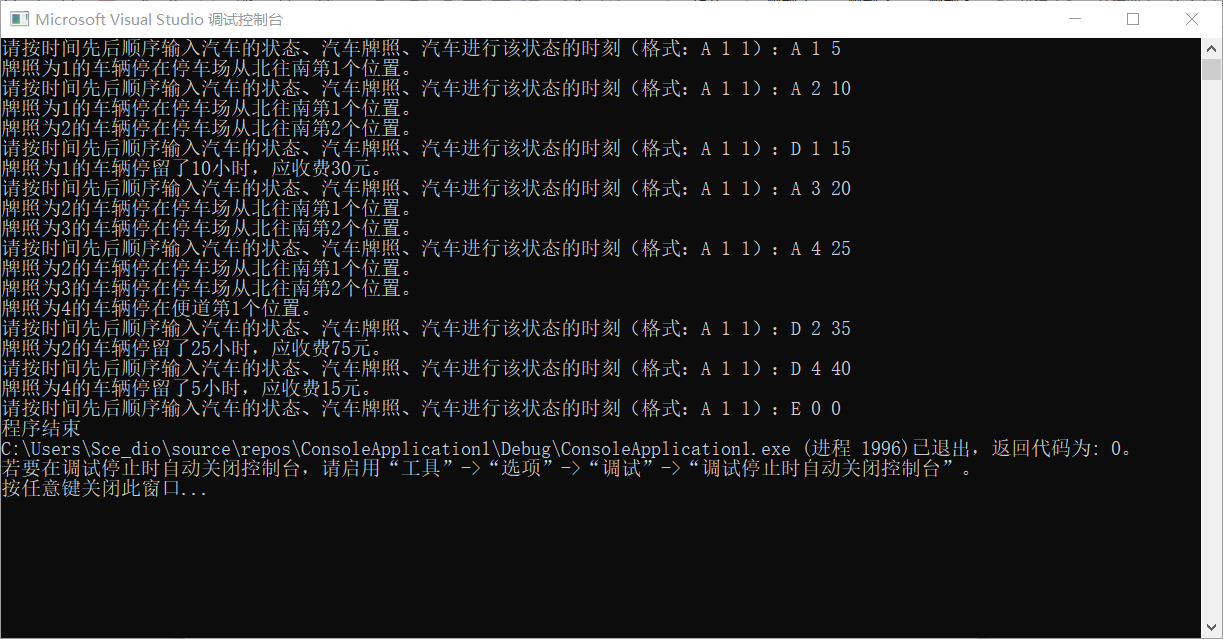
key3--;}}

} while (1 > 0);}

int main() {

parkingmode();}

**实践结果：**



**实践小结：**

1. 停车场的位置是动态变换的，最北边位置的车辆撤离后，后续车辆都应依次递补。可以定义几个变量记录停车场、便道的车辆数，停车场满了车辆才在外等。
2. 封装的结构体内的变量时间是记录车辆停进停车场的时间，但是我们在创建节点是统一把车辆到达的时间赋给变量时间，因此，若某车辆停留在停车场外等候，进入停车场时应对time重新赋值。
3. Key1,key2,key3表示栈、队列车辆数量，而i,j是下标，使用时应尤其注意。
4. 等候队列可以采用链队列的形式，等候车辆数目也不确定，便于车辆的移动。

**三、树**

题5、请使用数组输入二叉树的结点数据，按照提示以链表结构创建二叉树，完成后将链表内容输出。

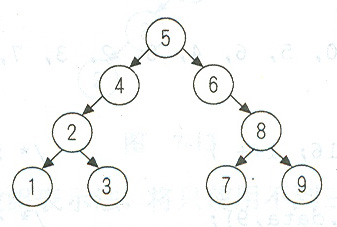
提示：创建二叉树结点数据的策略有三个，如下：

将第一个要创建的元素插入成为根节点。

将元素值与结点值比较，如果元素值大于结点值，将此元素送往结点的右儿子结点，如果右儿子结点不是空的，需要重复比较，否则创建结点将元素值插入。

如果元素值小于结点值，将此元素送往结点的左儿子结点，如果左儿子结点不是空的，需要重复比较，否则创建结点将此元素值插入。

例如：二叉树结点值输入的数据顺序是5，6，4，8，2，3，7，1，9。按照上述策略创建的二叉树，如下图所示：



**实践目的：**

掌握树内容排列的实际应用。

**实现代码：**

#include <iostream>

using namespace std;

int n, m;

struct BiSortTreeNode {

int data;

BiSortTreeNode \*lchild;

BiSortTreeNode \*rchild;

}\*first,\*p,\*q,\*r;

void seekLocation() {

if (m < (p->data)) {

if (p->lchild != NULL) p = p->lchild;

else { p->lchild = q; }

seekLocation();

}

if (m > (p->data)) {

if (p->rchild != NULL) p = p->rchild;

else { p->rchild = q; }

seekLocation();

}

}

void CreatTree() {

cout << "您想排序的数据有多少个？请输入：";

cin >> n;

cout << "请您依次输入您想排序的数据:";

cin >> m;

p = new BiSortTreeNode();

p->data = m;

p->lchild = NULL;

p->rchild = NULL;

first = p;

r = p;

int i = 1;

do {

cin >> m;

q = new BiSortTreeNode();

q->data = m;

seekLocation();

i++;

p = r;

} while (i != n);

}

void PreOrder(BiSortTreeNode \*p) {

if (p == NULL) return;

else {

cout << p->data << "\t";

PreOrder(p->lchild);

PreOrder(p->rchild);

}

}

void Order(BiSortTreeNode \*p) {

if (p == NULL) return;

else {

Order(p->lchild);

cout << p->data << "\t";

Order(p->rchild);

}

}

int main() {

CreatTree();

cout << "前序遍历该二叉树，结果为:";

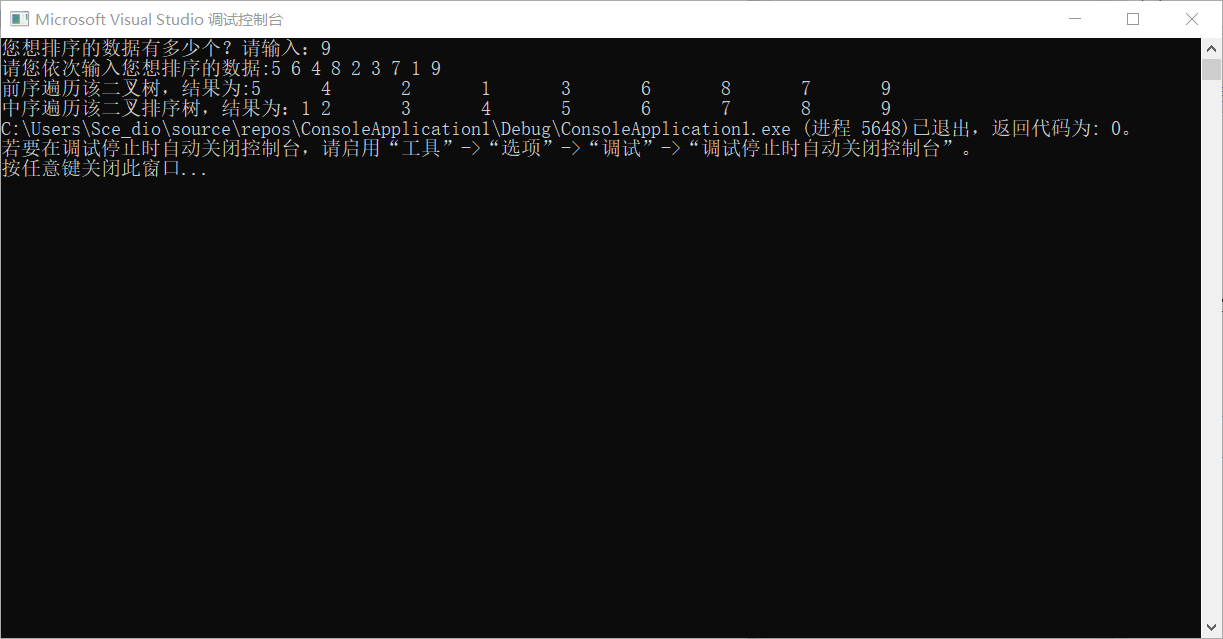
PreOrder(first);

cout << "\n中序遍历该二叉排序树，结果为：";

Order(first);

}

**实践结果：**



**实践小结：**

1. 该种方法构造的二叉树实质上是二叉排序树，左孩子的值都比双亲的值小，右孩子的值都比双亲的值大。根据二叉树的中序遍历，即可对输入数据排序。
2. 如果采用循环实现创建二叉排序树，需要注意循环流程控制。值不可能一直往左下方跑（不可能所有的值都比比较的节点的值小），也不可能一直往右下方跑。因此，在比较过程中应掺杂大、小的比较，在数据较少的情况下，出现问题的几率不大。但如果数据数量较大，很难保证不出问题。因此，循环+递归可能是一种保证每个节点都比大小的好方法。
3. 表示一颗二叉树必须得有“中序遍历+前序遍历/后序遍历”，也即起码需要包含中序遍历在内的两种遍历形式，才能确定唯一一颗二叉树。