上海理工大学光电信息与计算机工程学院

**《数据结构》实验报告**

****

**专　　业 计科**

**学生姓名　　 王秋实**

**学　 号 2135070824**

**年　　级 大二**

**指导教师 黄义萍**

**成 绩：**

**教师签字：**

报告格式要求

1. 正文字体中文为宋体，小四，行距为固定值16磅，西文为Times New Rome, 小四，行距为固定值16磅。
2. 章节标题为加粗宋体，三号，段前段后各16磅，行距为单倍行距。
3. 打印时需双面打印。

目录

[实验1和2 顺序表与单链表的综合应用 4](#_Toc66971694)

[实验3和4 堆栈应用与队列应用 5](#_Toc66971695)

[实验5和6 二叉树应用之哈夫曼树 6](#_Toc66971696)

[实验7 使用链表做排序和反转，队列应用 7](#_Toc66971697)

[实验8 二叉树搜索树 8](#_Toc66971698)

**数据结构实验一（线性表）**

**<通讯录的制作> 设计要求**

【问题描述】

编写一个小型通讯录管理系统，完成相关人员信息的建立、显示、查找、插入、删除等操作。要有良好的界面和较强的容错性。

【基本要求】

应包含以下几方面的功能：

1. 输入信息
2. 显示信息
3. 查找信息，以姓名为关键字
4. 删除信息
5. 信息存储到磁盘文件
6. 信息从磁盘文件导入

【数据】

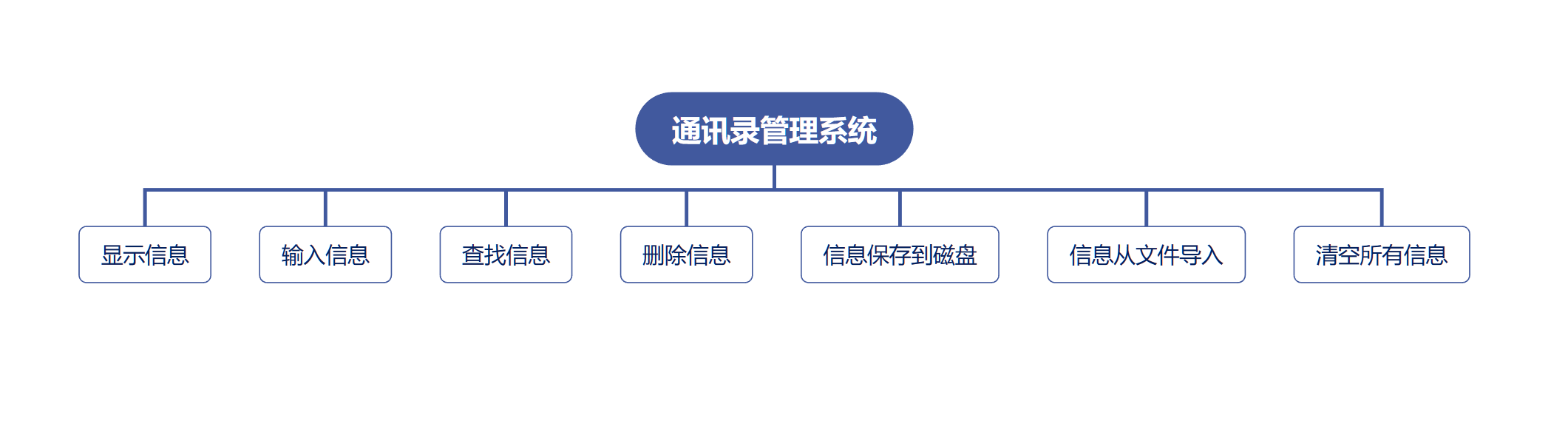
通讯录的每条信息至少包含以下几项内容：

1. 姓名
2. 街道
3. 城市
4. 国家
5. 电话号码

【实现方法】

使用双向链表进行设计。需要用到C语言中文件读写函数进行磁盘文件访问。

**程序功能框图**

****

**数据结构说明**

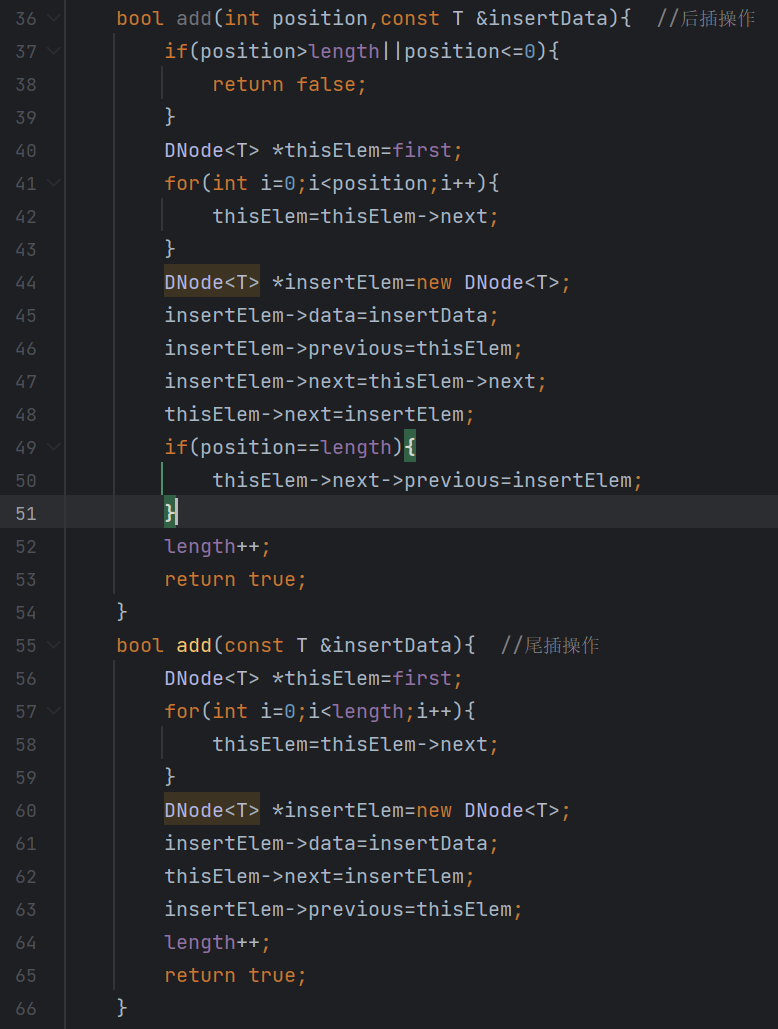
本程序使用双向链表存储，创建了DLinkedList.h头文件用于实现双向链表，其中定义了DNode数据类型和DLinkedList<T>类。前者有用于存储数据的data部分，和分别指向前驱元素和后继节点的previous和next指针。后者定义了头结点和线性表长度，构造和析构函数，print，后插和尾插add，remove，replace，isEmpty，clear和get函数，用于对链表的各种操作。

**重要算法核心代码**

1、构造函数和析构函数

构造函数会根据传入的数据类型申请一片对应的内存空间，同时将头结点指针指向该内存。

析构函数会先执行一遍链表的清空节点操作，再把头结点删除，完成对链表的完全清理。



2、给定位置的后插操作和尾插操作

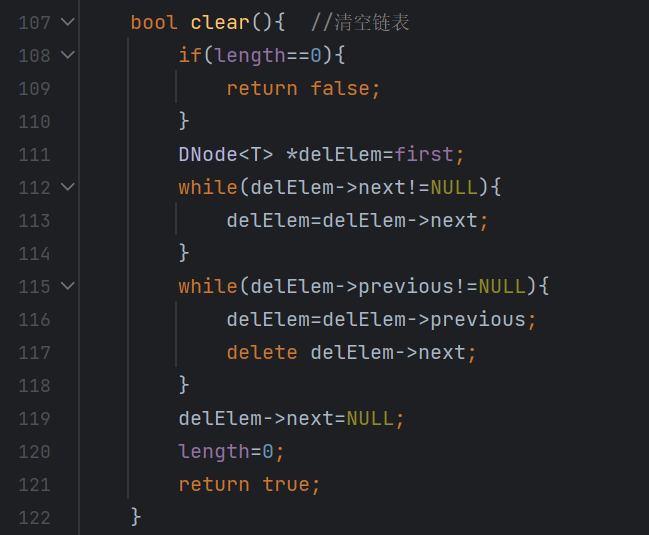
后插操作首先会根据位置定位到要后插的元素，然后更改前后指针以及插入元素的指针，此时会进行判尾操作，如果是在尾部插入，那么更改后继结点的指针操作不会执行

尾插操作类似于后插操作，会直接定位到链表尾再执行插入操作



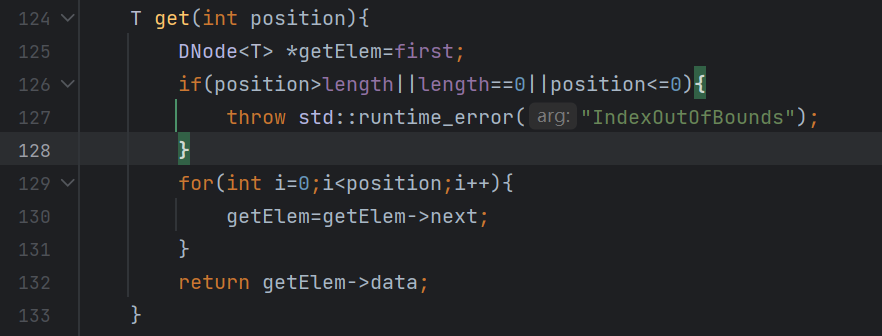
3、给定位置删除、修改信息

和插入操作的第一步类似，先要定位删除/修改的元素，然后对节点进行删除/修改操作并改变前后指针。不同的是删除前要对链表判空，如果是空链表则返回false。



4、清空链表

先对链表判空，如果已空则不需要清空。若不空，则定位到链表的末尾，从末尾开始逐个删除只保留头结点

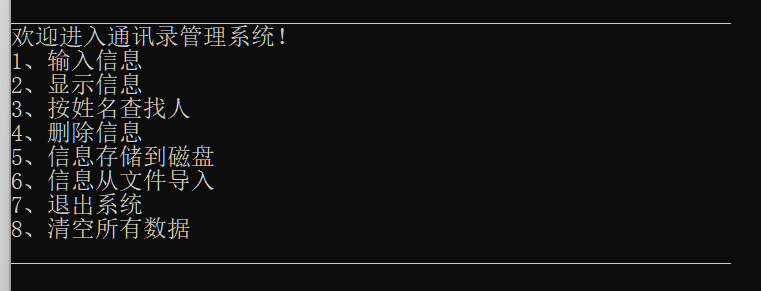


5、给定位置获取信息

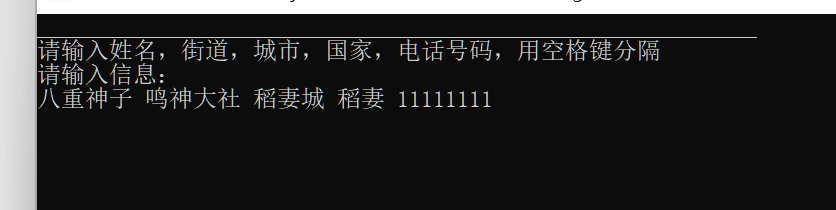
给定不合法的位置会抛出异常，若给定位置合法，则返回该位置对应的数据

**测试运行界面**

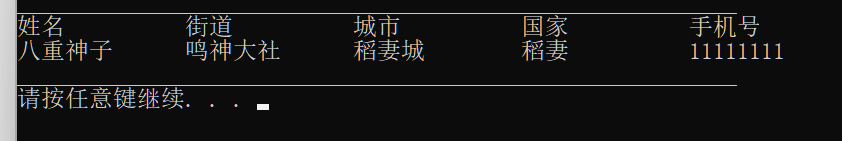
1. **主界面**

****

1. **输入信息**

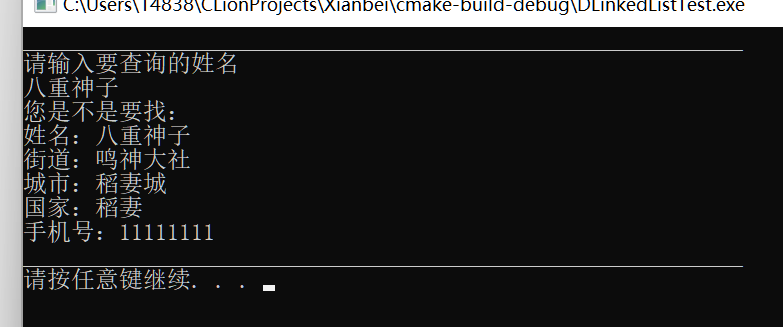
****

**显示：**

****

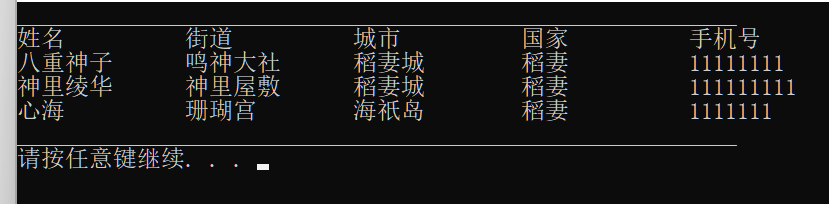
1. **按姓名查找人**

****

****

1. **删除信息**

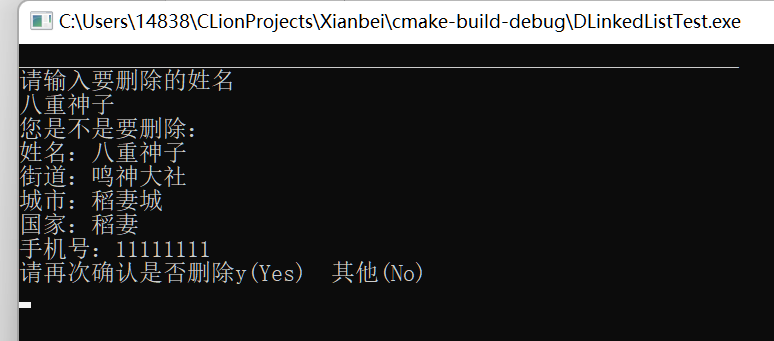
**现有的信息：**

****

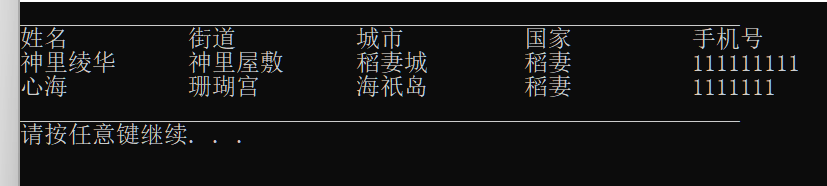
**输入需要删除的人名：**

****

**确认信息：**

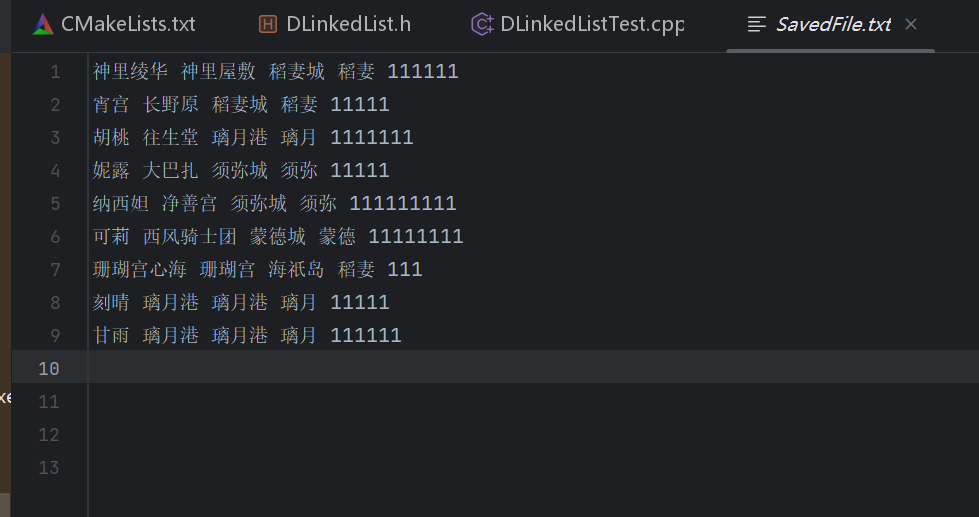
****

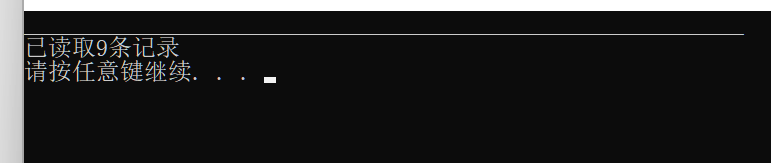
**输入y后即可删除**

****

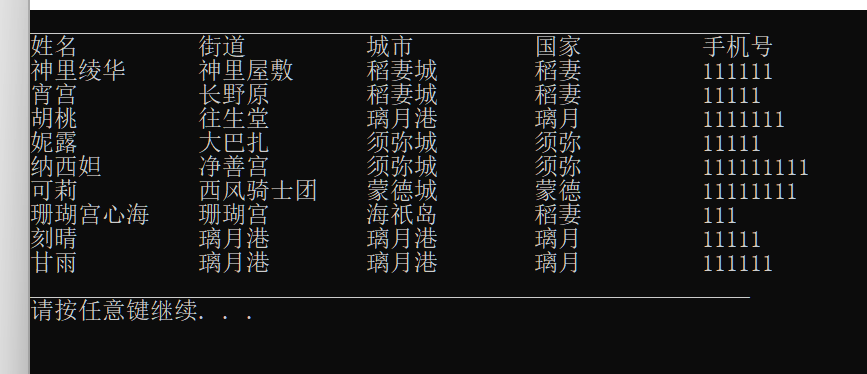
1. **信息从磁盘中读取**

**默认从程序存放目录下读取SavedFile.txt**

****

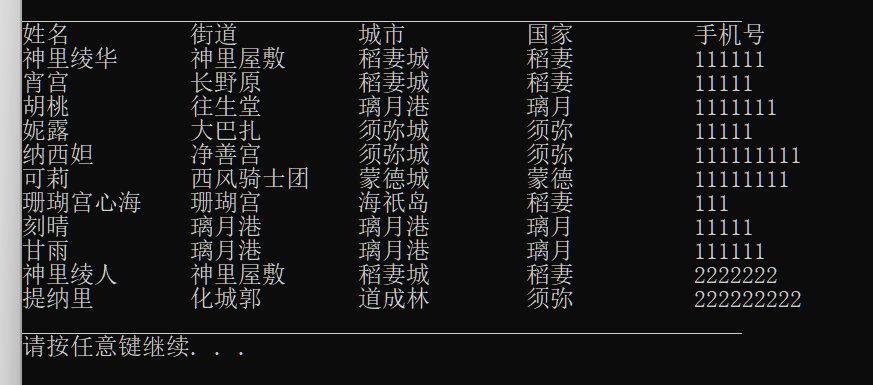
****

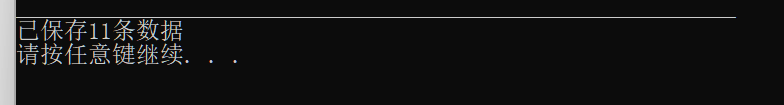
**可以在显示菜单里看到已读取的内容**

****

1. **信息写入磁盘**

**和上次相比，添加了两条信息**

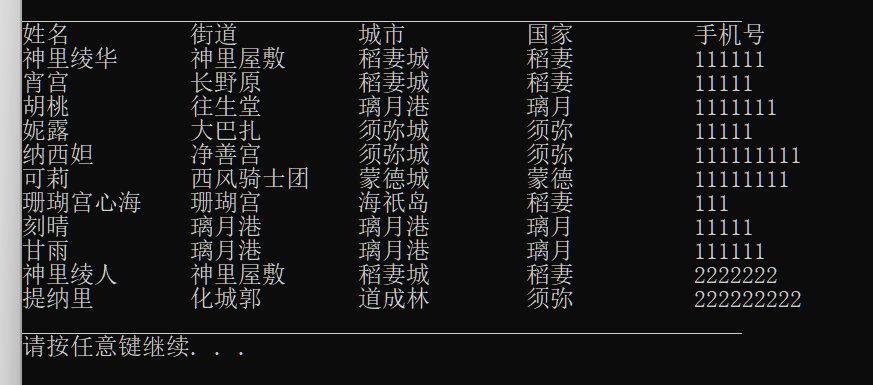
****

****

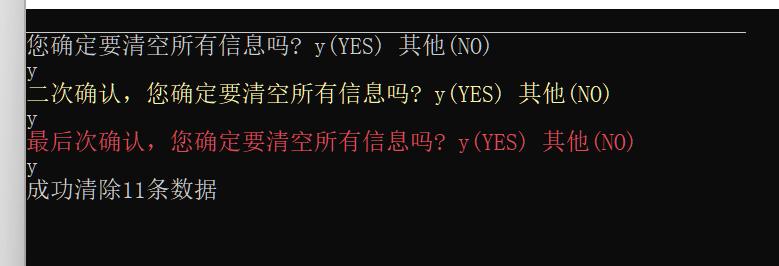
**文件已被修改，默认也是写入SavedFile.txt**

****

1. **清空所有数据**

****

**清空时需要三次确认**

****

****

**完整源程序：**

**头文件部分**

**#pragma once**

**#ifndef XIANBEI\_DLINKEDLIST\_H**

**#define XIANBEI\_DLINKEDLIST\_H**

**#include "iostream"**

**#include "exception"**

**template<typename T>**

**struct DNode{**

**T data;**

**DNode<T> \*previous=NULL;**

**DNode<T> \*next=NULL;**

**};**

**template<typename T>**

**class DLinkedList {**

**public:**

**int length;**

**DLinkedList(){ //构造函数**

**length=0;**

**first=new DNode<T>;**

**}**

**~DLinkedList(){ //析构函数**

**clear();**

**if(first!=NULL){**

**delete first;**

**first=NULL;**

**}**

**}**

**void print(){ //打印链表数据**

**DNode<T> \*thisElem;**

**thisElem=first->next;**

**while(thisElem!=NULL){**

**std::cout<<thisElem->data<<std::endl;**

**thisElem=thisElem->next;**

**}**

**}**

**bool add(int position,const T &insertData){ //后插操作**

**if(position>length||position<=0){**

**return false;**

**}**

**DNode<T> \*thisElem=first;**

**for(int i=0;i<position;i++){**

**thisElem=thisElem->next;**

**}**

**DNode<T> \*insertElem=new DNode<T>;**

**insertElem->data=insertData;**

**insertElem->previous=thisElem;**

**insertElem->next=thisElem->next;**

**thisElem->next=insertElem;**

**if(position==length){**

**thisElem->next->previous=insertElem;**

**}**

**length++;**

**return true;**

**}**

**bool add(const T &insertData){ //尾插操作**

**DNode<T> \*thisElem=first;**

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

**thisElem=thisElem->next;**

**}**

**DNode<T> \*insertElem=new DNode<T>;**

**insertElem->data=insertData;**

**thisElem->next=insertElem;**

**insertElem->previous=thisElem;**

**length++;**

**return true;**

**}**

**bool remove(int position){ //给定位置删除**

**if(position>length||length==0||position<=0){**

**return false;**

**}**

**DNode<T> \*delElem=first;**

**for(int i=0;i<position;i++){**

**delElem=delElem->next;**

**}**

**if(position==length){**

**DNode<T> \*previousElem=delElem->previous;**

**delete delElem;**

**previousElem->next=NULL;**

**}else{**

**DNode<T> \*nextElem=delElem->next;**

**DNode<T> \*previousElem=delElem->previous;**

**delete delElem;**

**previousElem->next=nextElem;**

**nextElem->previous=previousElem;**

**}**

**length--;**

**return true;**

**}**

**bool replace(int position,const T &insertData){ //给定位置修改值**

**if(position>length||length==0||position<=0){**

**return false;**

**}**

**DNode<T> \*replaceElem=first;**

**for(int i=0;i<position;i++){**

**replaceElem=replaceElem->next;**

**}**

**replaceElem->data=insertData;**

**return true;**

**}**

**bool isEmpty(){ //判空**

**if(length==0){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**bool clear(){ //清空链表**

**if(length==0){**

**return false;**

**}**

**DNode<T> \*delElem=first;**

**while(delElem->next!=NULL){**

**delElem=delElem->next;**

**}**

**while(delElem->previous!=NULL){**

**delElem=delElem->previous;**

**delete delElem->next;**

**}**

**delElem->next=NULL;**

**length=0;**

**return true;**

**}**

**T get(int position){**

**DNode<T> \*getElem=first;**

**if(position>length||length==0||position<=0){**

**throw std::runtime\_error("IndexOutOfBounds");**

**}**

**for(int i=0;i<position;i++){**

**getElem=getElem->next;**

**}**

**return getElem->data;**

**}**

**private:**

**DNode<T> \*first;**

**};**

**测试类部分（通讯录管理系统）**

**#include "iostream"**

**#include "string"**

**#include "DLinkedList.h"**

**#include "windows.h"**

**#include "fstream"**

**using namespace std;**

**class Person{**

**public:**

**string name;**

**string road;**

**string city;**

**string nation;**

**string phone;**

**Person(){};**

**Person(string name,string road,string city,string nation,string phone){**

**this->name=name;**

**this->road=road;**

**this->city=city;**

**this->nation=nation;**

**this->phone=phone;**

**}**

**};**

**DLinkedList<Person> d;**

**void Menu(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**cout<<"欢迎进入通讯录管理系统！"<<endl;**

**cout<<"1、输入信息"<<endl;**

**cout<<"2、显示信息"<<endl;**

**cout<<"3、按姓名查找人"<<endl;**

**cout<<"4、删除信息"<<endl;**

**cout<<"5、信息存储到磁盘"<<endl;**

**cout<<"6、信息从文件导入"<<endl;**

**cout<<"7、退出系统"<<endl;**

**cout<<"8、清空所有数据"<<endl;**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**}**

**void Input(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**cout<<"请输入姓名，街道，城市，国家，电话号码，用空格键分隔"<<endl;**

**cout<<"请输入信息："<<endl;**

**string name,road,city,nation,phone;**

**cin>>name>>road>>city>>nation>>phone;**

**Person p=\*new Person(name,road,city,nation,phone);**

**bool isRepeat=false;**

**for(int i=1;i<=d.length;i++){**

**Person p1=d.get(i);**

**if(p1.name==p.name){**

**isRepeat=true;**

**break;**

**}**

**}**

**if(isRepeat){**

**cout<<"同名记录已存在"<<endl;**

**}else{**

**d.add(p);**

**cout<<"添加成功！"<<endl;**

**}**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**Sleep(1000);**

**system("cls");**

**Sleep(300);**

**}**

**void PrintAll(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**cout<<"姓名 街道 城市 国家 手机号 "<<endl;**

**for(int i=1;i<=d.length;i++){**

**Person p=d.get(i);**

**printf("%-14s%-14s%-14s%-14s%-14s\n",p.name.c\_str(),p.road.c\_str(),p.city.c\_str(),p.nation.c\_str(),p.phone.c\_str());**

**}**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**system("pause");**

**}**

**void Search(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**cout<<"请输入要查询的姓名"<<endl;**

**string name;**

**cin>>name;**

**bool find=false;**

**Person p;**

**for(int i=1;i<=d.length;i++){**

**p=d.get(i);**

**if(name==p.name){**

**find=true;**

**break;**

**}**

**}**

**if(find){**

**cout<<"您是不是要找："<<endl;**

**cout<<"姓名："<<p.name<<endl;**

**cout<<"街道："<<p.road<<endl;**

**cout<<"城市："<<p.city<<endl;**

**cout<<"国家："<<p.nation<<endl;**

**cout<<"手机号："<<p.phone<<endl;**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**system("pause");**

**}else{**

**cout<<"您要找的人不存在"<<endl;**

**Sleep(3000);**

**}**

**}**

**void Delete(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**cout<<"请输入要删除的姓名"<<endl;**

**string name;**

**cin>>name;**

**bool find=false;**

**Person p;**

**int position=0;**

**for(int i=1;i<=d.length;i++){**

**p=d.get(i);**

**if(name==p.name){**

**find=true;**

**position=i;**

**break;**

**}**

**}**

**if(find){**

**cout<<"您是不是要删除："<<endl;**

**cout<<"姓名："<<p.name<<endl;**

**cout<<"街道："<<p.road<<endl;**

**cout<<"城市："<<p.city<<endl;**

**cout<<"国家："<<p.nation<<endl;**

**cout<<"手机号："<<p.phone<<endl;**

**cout<<"请再次确认是否删除y(Yes) 其他(No)"<<endl;**

**char option;**

**cin>>option;**

**if(option=='y'){**

**d.remove(position);**

**cout<<"删除成功！"<<endl;**

**}else{**

**cout<<"您取消了操作"<<endl;**

**}**

**}else{**

**cout<<"您要删除的人不存在"<<endl;**

**}**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**system("pause");**

**}**

**void Save(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**ofstream wFile("SavedFile.txt",ios::out);**

**if(!wFile){**

**cerr<<"文件写入失败，请检查程序写入权限或磁盘坏道!"<<endl;**

**}else{**

**for(int i=1;i<=d.length;i++){**

**Person p=d.get(i);**

**wFile<<p.name<<" "<<p.road<<" "<<p.city<<" "<<p.nation<<" "<<p.phone<<endl;**

**}**

**cout<<"已保存"<<d.length<<"条数据"<<endl;**

**}**

**system("pause");**

**}**

**void InputFromFile(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**ifstream rFile("SavedFile.txt",ios::in);**

**if(!rFile){**

**cerr<<"文件读取失败，请检查程序读取权限或磁盘坏道!"<<endl;**

**}else{**

**int i=0;**

**for(;;i++){**

**string name;**

**string road;**

**string city;**

**string nation;**

**string phone;**

**rFile>>name>>road>>city>>nation>>phone;**

**if(name==""){**

**break;**

**}**

**Person p(name,road,city,nation,phone);**

**d.add(p);**

**}**

**cout<<"已读取"<<i<<"条记录"<<endl;**

**system("pause");**

**}**

**}**

**void Clear(){**

**cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;**

**string confirm1;**

**cout<<"您确定要清空所有信息吗? y(YES) 其他(NO)"<<endl;**

**cin>>confirm1;**

**if(confirm1!="y"){**

**cout<<"用户取消操作"<<endl;**

**return;**

**}**

**string confirm2;**

**cout<<"\033[33;1m二次确认，您确定要清空所有信息吗? y(YES) 其他(NO)\033[0m"<<endl;**

**cin>>confirm2;**

**if(confirm1!="y"){**

**cout<<"用户取消操作"<<endl;**

**return;**

**}**

**string confirm3;**

**cout<<"\033[31;1m最后次确认，您确定要清空所有信息吗? y(YES) 其他(NO)\033[0m"<<endl;**

**cin>>confirm3;**

**if(confirm1!="y"){**

**cout<<"用户取消操作"<<endl;**

**return;**

**}**

**int size=d.length;**

**d.clear();**

**cout<<"成功清除"<<size<<"条数据"<<endl;**

**Sleep(3000);**

**}**

**int main(){**

**while(true){**

**system("cls");**

**Menu();**

**int choose;**

**cin>>choose;**

**switch (choose) {**

**case 1:**

**system("cls");**

**Input();**

**break;**

**case 2:**

**system("cls");**

**PrintAll();**

**break;**

**case 3:**

**system("cls");**

**Search();**

**break;**

**case 4:**

**system("cls");**

**Delete();**

**break;**

**case 5:**

**system("cls");**

**Save();**

**break;**

**case 6:**

**system("cls");**

**InputFromFile();**

**break;**

**case 7:**

**system("cls");**

**cout<<"感谢您的使用"<<endl;**

**system("pause");**

**exit(0);**

**case 8:**

**system("cls");**

**Clear();**

**break;**

**default:**

**system("cls");**

**cerr<<"您输入的指令有误，请重新输入"<<endl;**

**system("pause");**

**break;**

**}**

**}**

**}**

**数据结构实验二（堆栈与队列）**

1. **堆栈应用(1)**

【问题描述】

输入任一表达式，“#”为表达式的结束符，试写一判断表达式中圆括号（“(”与“)”）是否配对的算法。用C语言实现。

【测试情况】

Input the expression string ended with '#' (length≤80):

(a+((a+b)\*c)-d/c)\*e#

Matched

Input the expression string ended with '#' (length≤80):

((a+((a+b)\*c)-d/c)\*e#

Unmatched

1. **堆栈应用(2)**

【问题描述】

算术表达式求值问题：约定表达式中只包含加减乘除4种算术运算（表达式中允许出现圆括号），所有运算对象均为简单变量，输入表达式的结束符为“#”，按回车键后即可得到运算结果。用C语言实现。

1. **队列应用**

【问题描述】

对顺序循环队列，将自然数按序入队、出队。具体的操作是：队列未满时，入队、入队、出队（即做连续两次入队操作之后，做一次出队操作），输出出队元素的值；队列满时，执行连续的出队操作，输出出队元素的值（应与队列未满时所输出的有不同标识），直至队列为空。编写算法实现以上操作。用C语言实现。

【测试情况】

假设顺序循环队列使用的数组大小为20

运行程序得到的实际输出如下：

1\*2\*3\*4\*5\*6\*7\*8\*9\*10\*11\*12\*13\*14\*15\*16\*17\*18\*19#20#21#22#23#24#25#26#27#28#29#30#31#32#33#34#35#36#37#

**2、程序功能框图**

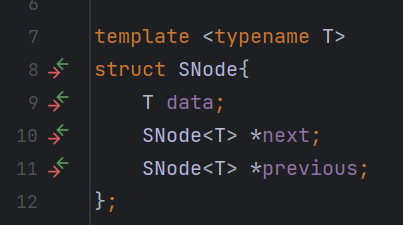
1、用双向循环链栈实现了圆括号的匹配

2、用两个堆栈：操作数栈和运算符栈实现了中缀表达式求值的问题

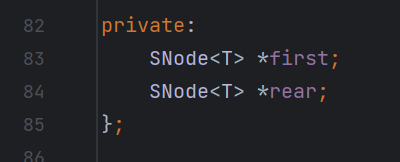
3、用顺序队列实现了规定操做

**3、数据结构说明**

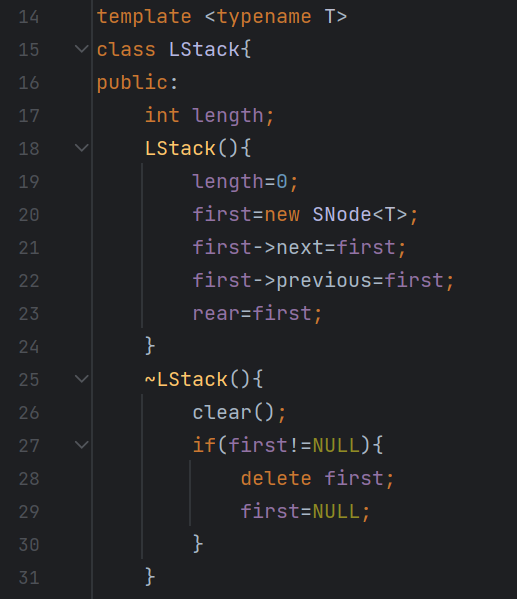
1、链栈



链表节点



私有变量：链栈头结点，链栈尾节点

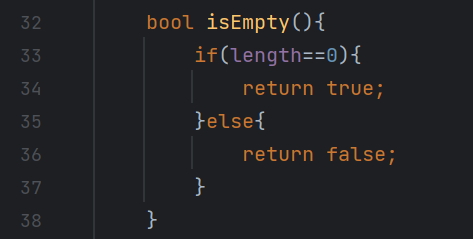


构造方法：

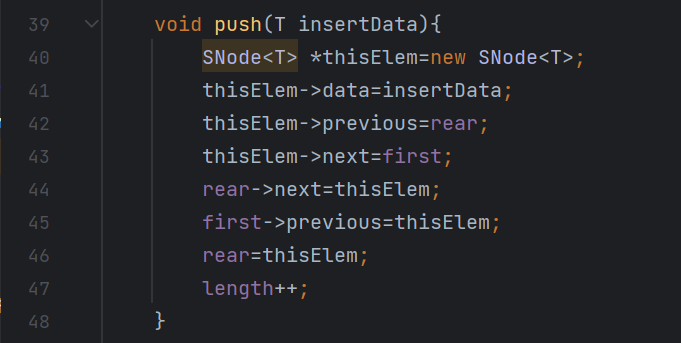
初始化各参量

析构方法：

先调用clear()删除所有节点，然后再删除头结点



判空操作

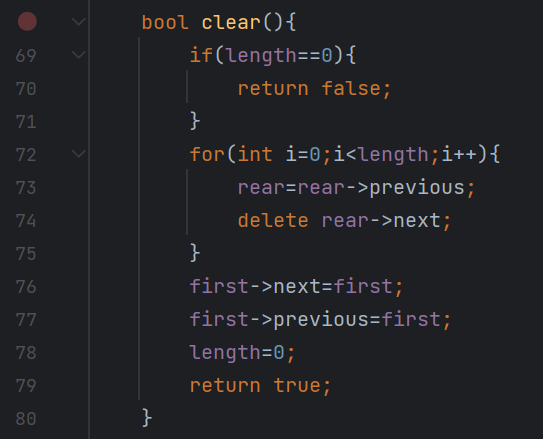


入栈操作，只需更改头尾节点指针即可



查看栈顶元素操作：只返回栈顶元素而不删除

出栈操作：删除并弹出栈顶元素



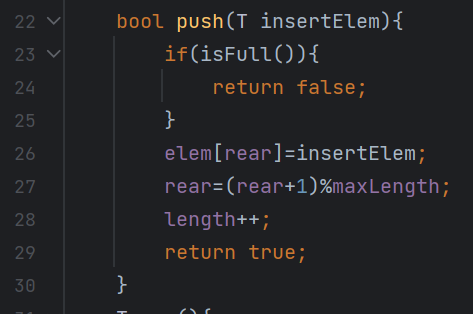
清空操作：

从栈尾到栈首依次删除元素

2、顺序队列



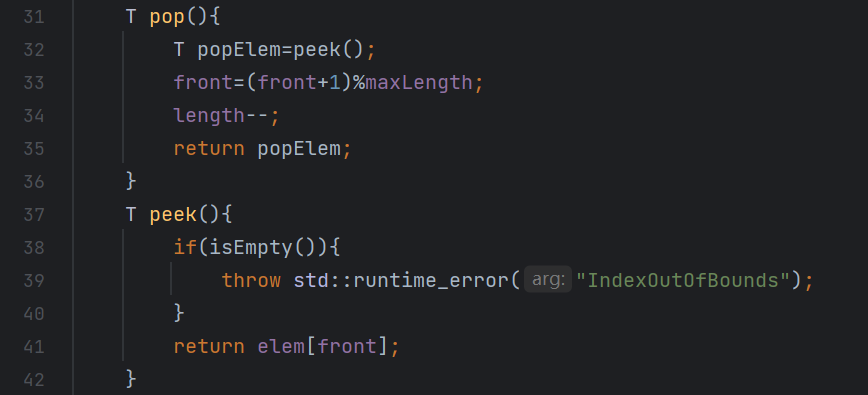
私有变量和构造方法

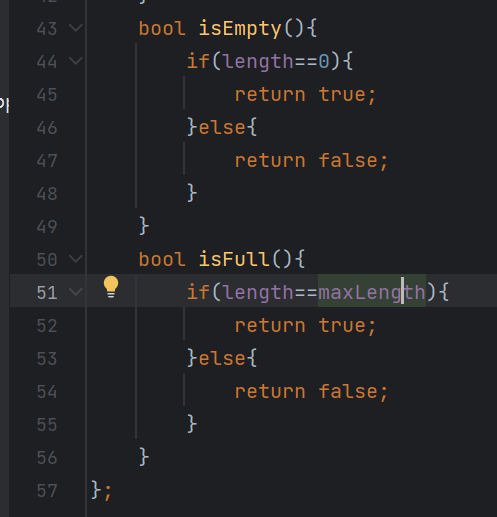


查看队首元素操作

出队操作

入队操作

****

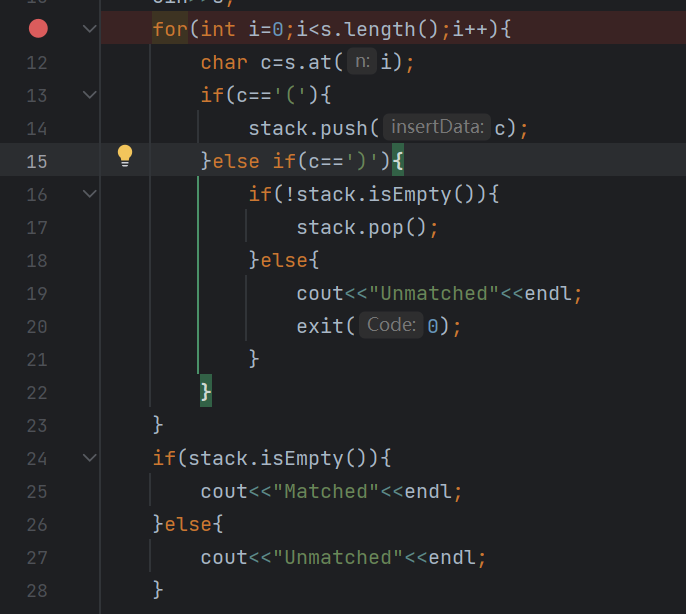
****

判空和判满操作

**4、重要算法核心代码**

1、括号匹配：

遇到左括号就入栈，遇到右括号就出栈。遇到右括号时栈空或读完所有字符后栈不空，说明不匹配，否则匹配

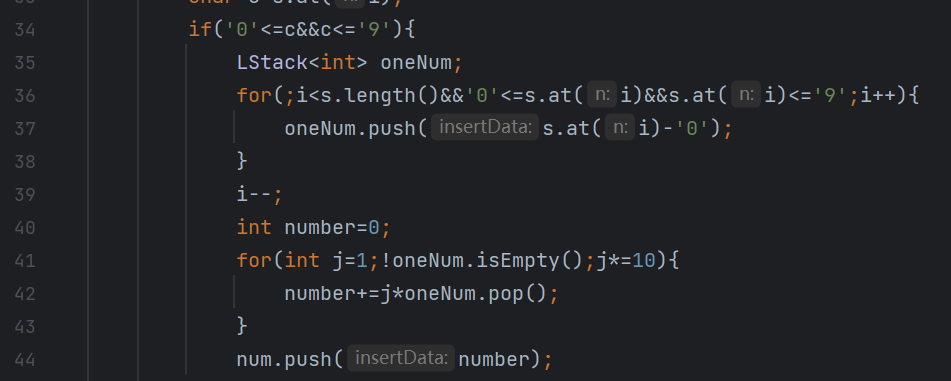


2、表达式求值：

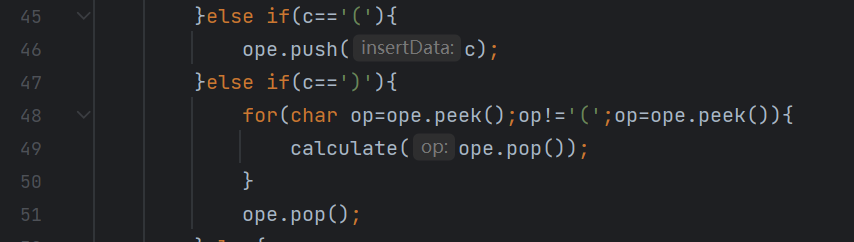
设置两个栈，操作数栈和运算符栈

遵循以下原则：

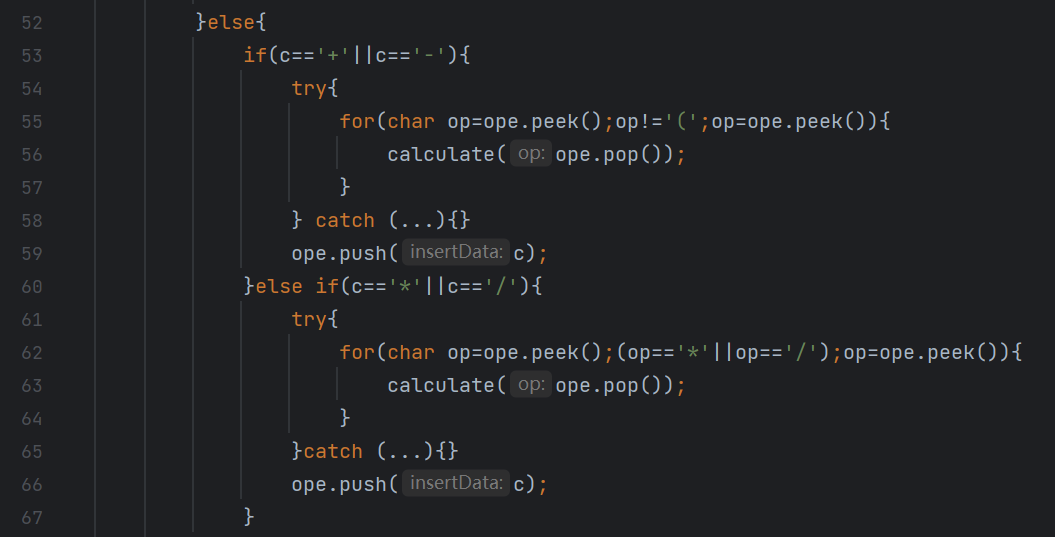
1. 遇到操作数，压入操作数栈中，此操作需要另一个栈读取多位操作数



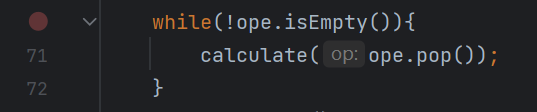
2、遇到左括号，直接压入运算符栈；遇到右括号，依次弹出运算符，被弹出的运算符立即生效，即弹出两个操作数计算出结果后便再次压入操作数栈，直至弹出左括号为止



3、遇到运算符，需依次弹出所有优先级大于或等于该运算符的所有运算符并立即生效，直至遇到左括号或者优先级低于该运算符的运算符。



4、最后弹出所有剩余在栈中的运算符并立即生效



1. 队列弹出

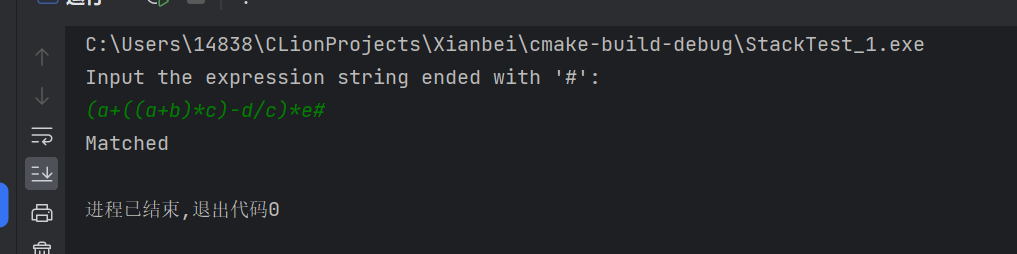
每做一次操作都检查一遍队列是否满，如果没满则执行下一个操作，如果已满

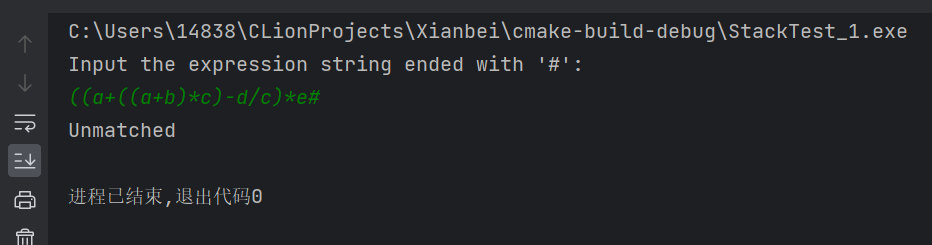
则弹出所有元素并停止



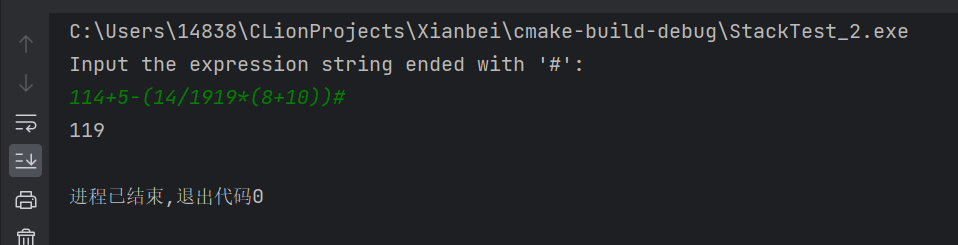
**5、测试运行界面**

1、括号匹配

****

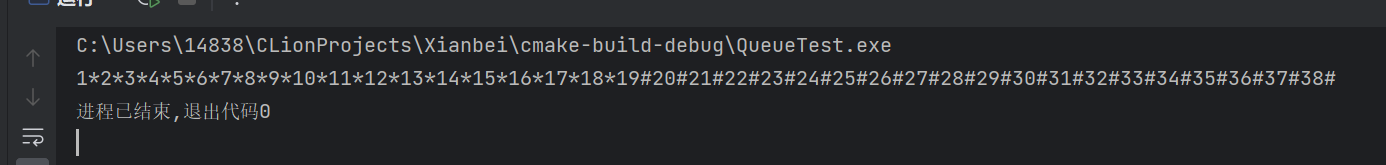


2、表达式求值



1. 队列操作

由于此队列单列一个length变量判空和判满，所以满队列时元素个数为20个，即没有浪费任何一个数组空间



**6、完整源程序**

**链栈头文件：**

#pragma once

#ifndef XIANBEI\_LSTACK\_H

#define XIANBEI\_LSTACK\_H

#include "iostream"

#include "exception"

template <typename T>

struct SNode{

T data;

SNode<T> \*next;

SNode<T> \*previous;

};

template <typename T>

class LStack{

public:

int length;

LStack(){

length=0;

first=new SNode<T>;

first->next=first;

first->previous=first;

rear=first;

}

~LStack(){

clear();

if(first!=NULL){

delete first;

first=NULL;

}

}

bool isEmpty(){

if(length==0){

return true;

}else{

return false;

}

}

void push(T insertData){

SNode<T> \*thisElem=new SNode<T>;

thisElem->data=insertData;

thisElem->previous=rear;

thisElem->next=first;

rear->next=thisElem;

first->previous=thisElem;

rear=thisElem;

length++;

}

T peek(){

if(length==0){

throw std::runtime\_error("IndexOutOfBounds");

}

return rear->data;

}

T pop(){

if(length==0){

throw std::runtime\_error("IndexOutOfBounds");

}

SNode<T> \*popElem=rear;

T popData=rear->data;

rear=popElem->previous;

rear->next=first;

first->previous=rear;

delete popElem;

length--;

return popData;

}

bool clear(){

if(length==0){

return false;

}

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

rear=rear->previous;

delete rear->next;

}

first->next=first;

first->previous=first;

length=0;

return true;

}

private:

SNode<T> \*first;

SNode<T> \*rear;

};

#endif //XIANBEI\_LSTACK\_H

**顺序队列头文件：**

#ifndef XIANBEI\_AQUEUE\_H

#define XIANBEI\_AQUEUE\_H

#include "iostream"

#include "exception"

template <typename T>

class AQueue{ //都什么年代了，还用传统顺序队列

private:

const static int maxLength=20;

T elem[maxLength];

int front;

int rear;

public:

int length;

AQueue(){

front=0;

rear=0;

length=0;

}

bool push(T insertElem){

if(isFull()){

return false;

}

elem[rear]=insertElem;

rear=(rear+1)%maxLength;

length++;

return true;

}

T pop(){

T popElem=peek();

front=(front+1)%maxLength;

length--;

return popElem;

}

T peek(){

if(isEmpty()){

throw std::runtime\_error("IndexOutOfBounds");

}

return elem[front];

}

bool isEmpty(){

if(length==0){

return true;

}else{

return false;

}

}

bool isFull(){

if(length==maxLength){

return true;

}else{

return false;

}

}

};

#endif //XIANBEI\_AQUEUE\_H

**括号匹配源程序：**

#include "iostream"

#include "string"

#include "LStack.h"

using namespace std;

int main(){

LStack<char> stack;

cout<<"Input the expression string ended with '#':"<<endl;

string s;

cin>>s;

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

char c=s.at(i);

if(c=='('){

stack.push(c);

}else if(c==')'){

if(!stack.isEmpty()){

stack.pop();

}else{

cout<<"Unmatched"<<endl;

exit(0);

}

}

}

if(stack.isEmpty()){

cout<<"Matched"<<endl;

}else{

cout<<"Unmatched"<<endl;

}

}

**表达式求值源程序：**

#include "iostream"

#include "string"

#include "LStack.h"

using namespace std;

LStack<int> num;

LStack<char> ope;

void calculate(char op){

int b=num.pop();

int a=num.pop();

switch (op) {

case '+':

num.push(a+b);

break;

case '-':

num.push(a-b);

break;

case '\*':

num.push(a\*b);

break;

case '/':

num.push(a/b);

break;

}

}

int main(){

cout<<"Input the expression string ended with '#':"<<endl;

string s;

cin>>s;

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

char c=s.at(i);

if('0'<=c&&c<='9'){

LStack<int> oneNum;

for(;i<s.length()&&'0'<=s.at(i)&&s.at(i)<='9';i++){

oneNum.push(s.at(i)-'0');

}

i--;

int number=0;

for(int j=1;!oneNum.isEmpty();j\*=10){

number+=j\*oneNum.pop();

}

num.push(number);

}else if(c=='('){

ope.push(c);

}else if(c==')'){

for(char op=ope.peek();op!='(';op=ope.peek()){

calculate(ope.pop());

}

ope.pop();

}else{

if(c=='+'||c=='-'){

try{

for(char op=ope.peek();op!='(';op=ope.peek()){

calculate(ope.pop());

}

} catch (...){}

ope.push(c);

}else if(c=='\*'||c=='/'){

try{

for(char op=ope.peek();(op=='\*'||op=='/');op=ope.peek()){

calculate(ope.pop());

}

}catch (...){}

ope.push(c);

}

}

}

while(!ope.isEmpty()){

calculate(ope.pop());

}

cout<<num.pop()<<endl;

}

**队列操作源程序：**

#include "iostream"

#include "AQueue.h"

using namespace std;

int main(){

AQueue<int> queue;

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

queue.push(i);

if(queue.isFull()){

while(!queue.isEmpty()){

cout<<queue.pop()<<"#";

}

break;

}else{

queue.push(++i);

}

if(queue.isFull()){

while(!queue.isEmpty()){

cout<<queue.pop()<<"#";

}

break;

}else{

cout<<queue.pop()<<"\*";

}

}

}

**数据结构实验三（二叉树）**

**<“哈夫曼树”的设计与实现> 设计要求**

【问题描述】

假设有一段电文由字符集 {A, B, C, D, E, F, G, H} 中的字符组成，各字符在电文中出现的频率由对应次数集 {5，29，7，8，14，23，3，11} 中的数字表示，请设计各字符的哈夫曼编码。

【基本要求】

应包含以下几方面的功能：

1. 设计哈夫曼树。具体构造方法如下：以字符集{A, B, C, D, E, F, G, H} 中的字符作为叶子结点，以各字符在次数集 {5，29，7，8，14，23，3，11} 中对应的次数作为各叶子结点的权值构造一棵哈夫曼树。
2. 设计哈夫曼编码。按照构造出来的哈夫曼树，规定哈夫曼树的左分支为0，右分支为1，则从根结点到每个叶子结点所经过的分支对应的0和1组成的序列便为该结点对应字符的哈夫曼编码。
3. 依次求出每个字符的哈夫曼编码并输出。

【扩展功能】

从键盘上分别输入哈夫曼编码字符的个数以及每个字符对应的权值，程序执行中请一步一步依次显示出哈夫曼树的构造过程，最后输出每个权值对应的哈夫曼编码。

【实现方法】

选择合适的存储结构，要便于从叶子结点找到双亲结点。

**2、程序功能框图：**

输入格式：第一行一个整数n，接下来的i行输入一个字符Ci和一个整数Wi，表示Ci这个字符的权重是Wi

样例输入：

8

A 5

B 29

C 7

D 8

E 14

F 23

G 3

H 11

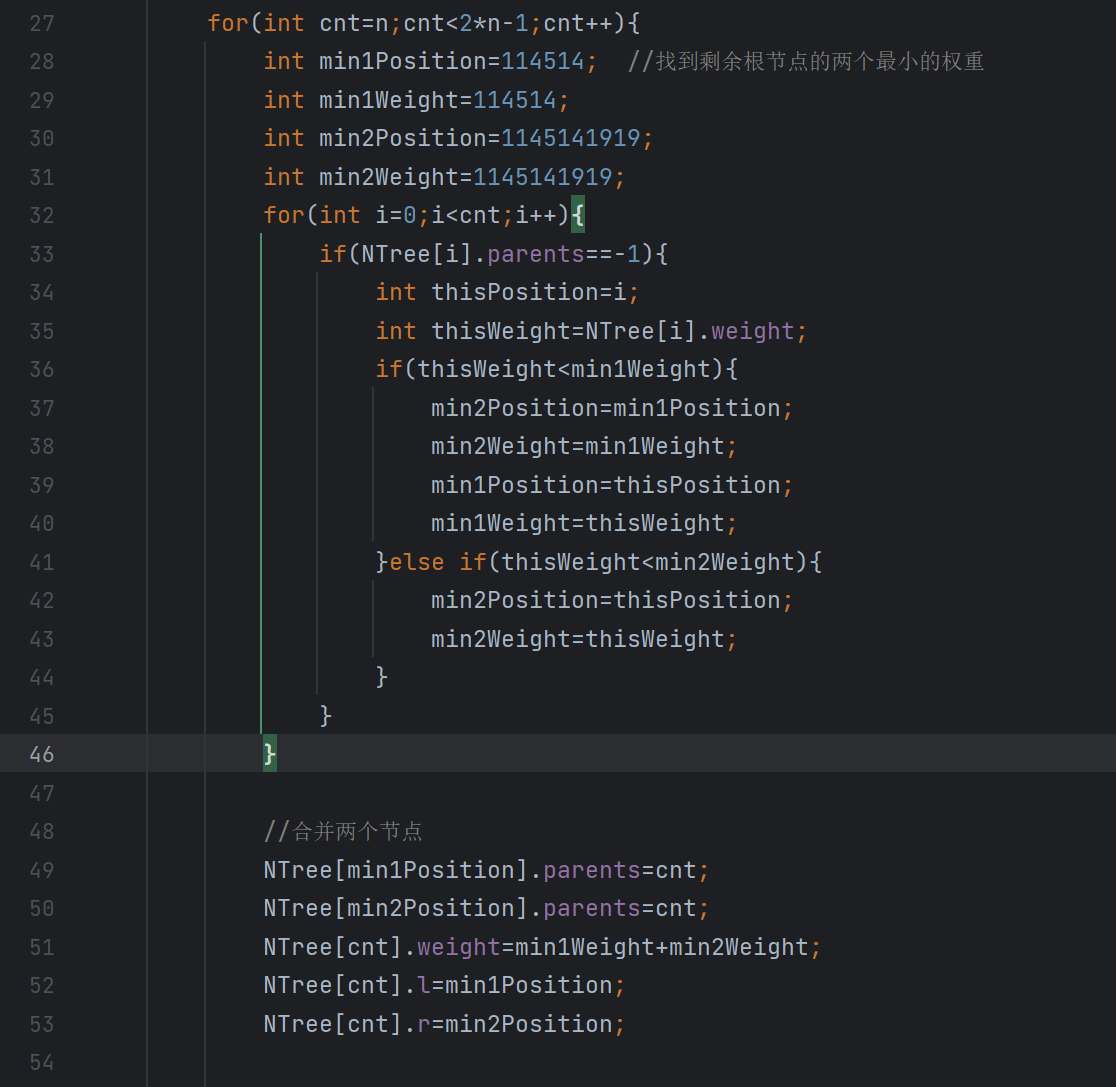
输出格式：第一部分输出每一步剩余结点的权重，第二部分输出字符和对应的哈夫曼编码

**3、数据结构说明**

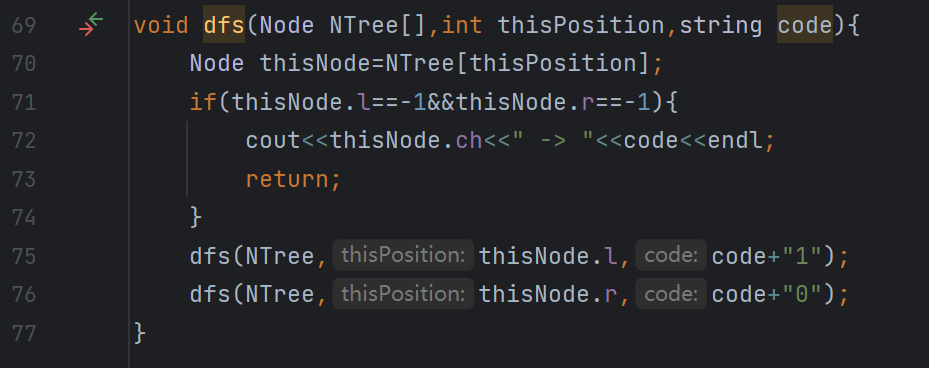
利用贪心算法，每次选择权重最小的两个结点合并，从而构造一棵权重最小的二叉树，然后从根节点开始，左孩子编码为1，右孩子编码为0，叶子结点是包含字符的结点，此时的编码即是该字符的哈夫曼编码。

1. **重要算法核心代码**

构造哈夫曼树：



先找到剩余结点的两个最小权重的结点，然后将它们合并。



dfs前序遍历整个哈夫曼树，碰到叶结点返回该字符的哈夫曼编码

1. **测试运行界面：**

样例输入：

8

A 5

B 29

C 7

D 8

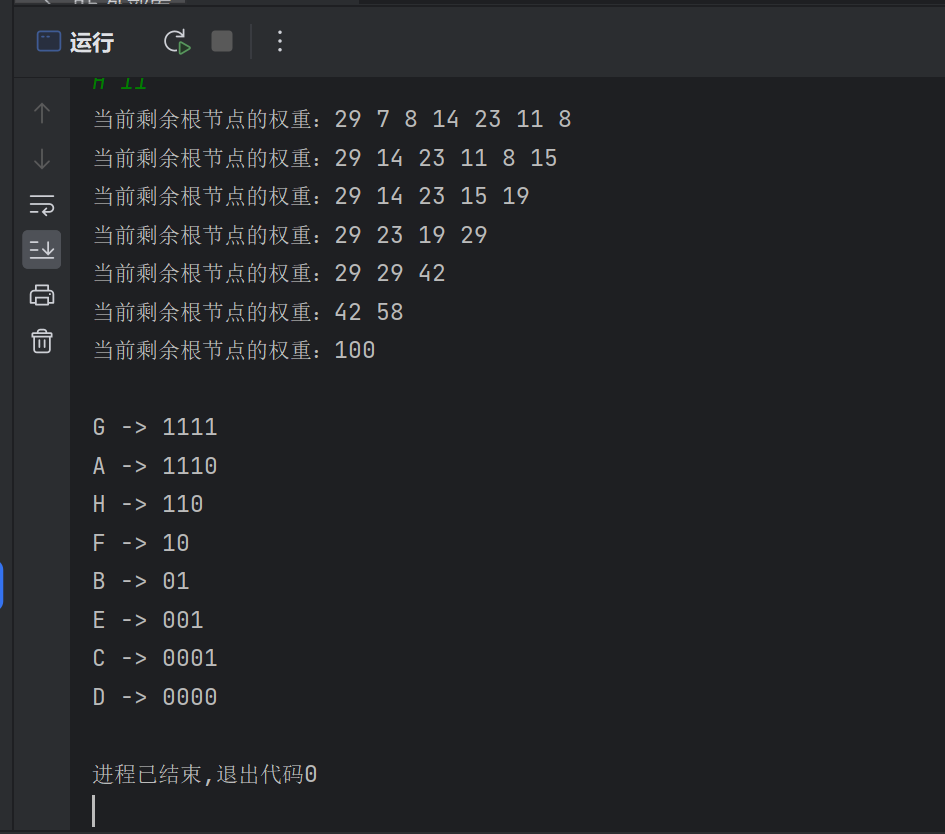
E 14

F 23

G 3

H 11

样例输出：



1. **完整源程序：**

#include "iostream"

using namespace std;

struct Node{

int id=-1;

int weight;

char ch;

int parents=-1;

int l=-1;

int r=-1;

};

void dfs(Node NTree[],int thisNode,string code);

int main(){

int n;

cin>>n;

Node NTree[2\*n-1];

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

char thisChar;

int weight;

cin>>thisChar>>weight;

NTree[i].id=i;

NTree[i].ch=thisChar;

NTree[i].weight=weight;

}

for(int cnt=n;cnt<2\*n-1;cnt++){

int min1Position=114514; //找到剩余根节点的两个最小的权重

int min1Weight=114514;

int min2Position=1145141919;

int min2Weight=1145141919;

for(int i=0;i<cnt;i++){

if(NTree[i].parents==-1){

int thisPosition=i;

int thisWeight=NTree[i].weight;

if(thisWeight<min1Weight){

min2Position=min1Position;

min2Weight=min1Weight;

min1Position=thisPosition;

min1Weight=thisWeight;

}else if(thisWeight<min2Weight){

min2Position=thisPosition;

min2Weight=thisWeight;

}

}

}

//合并两个节点

NTree[min1Position].parents=cnt;

NTree[min2Position].parents=cnt;

NTree[cnt].weight=min1Weight+min2Weight;

NTree[cnt].l=min1Position;

NTree[cnt].r=min2Position;

cout<<"当前剩余根节点的权重：";

for(int i=0;i<=cnt;i++){

if(NTree[i].parents==-1){

cout<<NTree[i].weight<<" ";

}

}

cout<<endl;

}

cout<<endl;

dfs(NTree,2\*n-2,"");

}

void dfs(Node NTree[],int thisPosition,string code){

Node thisNode=NTree[thisPosition];

if(thisNode.l==-1&&thisNode.r==-1){

cout<<thisNode.ch<<" -> "<<code<<endl;

return;

}

dfs(NTree,thisNode.l,code+"1");

dfs(NTree,thisNode.r,code+"0");

}

**数据结构实验四（综合练习）**

**1、链表**

【问题描述】

给定两个已升序排序的链表L1和L2，只使用基本的链表操作编写计算L1∩L2的过程。结果链表也要求是升序排序的。

**2、链表/堆栈**

【问题描述】

用两种非递归方法以O(N)的时间复杂度，实现反转一个单链表。

* 做法1：用堆栈来保存链表的遍历，用pop倒序输出，每倒序输出一个值就构造新链表的一个结点。
* 做法2：在原链表上对结点指针重新赋值操作来进行链表的反转。

**3、队列**

【问题描述】

假设以数组Q[m]存放循环队列中的元素，同时以rear和length 分别指示环形队列中的队尾位置和队列中所含元素的个数：

1）求队列中第一个元素的实际位置。

2）给出该循环队列的队空条件和队满条件，并写出相应的插入(enqueue)和删除(delqueue)元素的操作。

**4、二叉树**

【问题描述】

判断一个二叉树是不是二叉搜索树。

二叉搜索树必须满足：

1)或者是空树

2)如果不是空树，则对于任意一个节点

* 左子树（如果有）中所有节点都小于该节点
* 右子树（如果有）中所有节点都大于该节点
* 左子树（如果有）也是二叉搜索树
* 右子树（如果有）也是二叉搜索树

**2、程序功能框图**

1、链表：

输入格式：第一行两个整数n m，表示两个链表分别有n项和m项，接下来的两行分别有n和m个数，第k行第i个数表示第k-1个链表的第i项的值

输出格式：共一行，输出两个链表相同的元素，用空格分开

2、链表/堆栈：

输入格式：第一行一个整数n，表示链表共有n项，第二行n个整数，表示链表的n项。

输出格式：共一行，将链表翻转后输出链表，用空格分开

3、队列：

输入格式：第一行一个整数n，表示链表共有n项，第二行n个整数，表示链表的n项。

输出格式：输出队头位置，是否队空，是否队满

4、二叉树

输入格式：第一行一个整数n，接下来的n行中，第i行有三个整数，分别表示序号为i的结点的结点值，左子树和右子树的结点序号

输出格式：一行，是二叉搜索树输出yes，不是输出no

**3、数据结构说明**

1、链表

开了三个链表，分别表示两个初始链表和结果链表，两个指针指向链表的下一个元素，遍历整个链表即可找出相同的元素

2、链表/堆栈：

使用一个双向循环链表，表示初始的元素序列，然后将链表的元素从尾到头依次弹出到另一个链表中，从而实现链表的翻转

3、队列

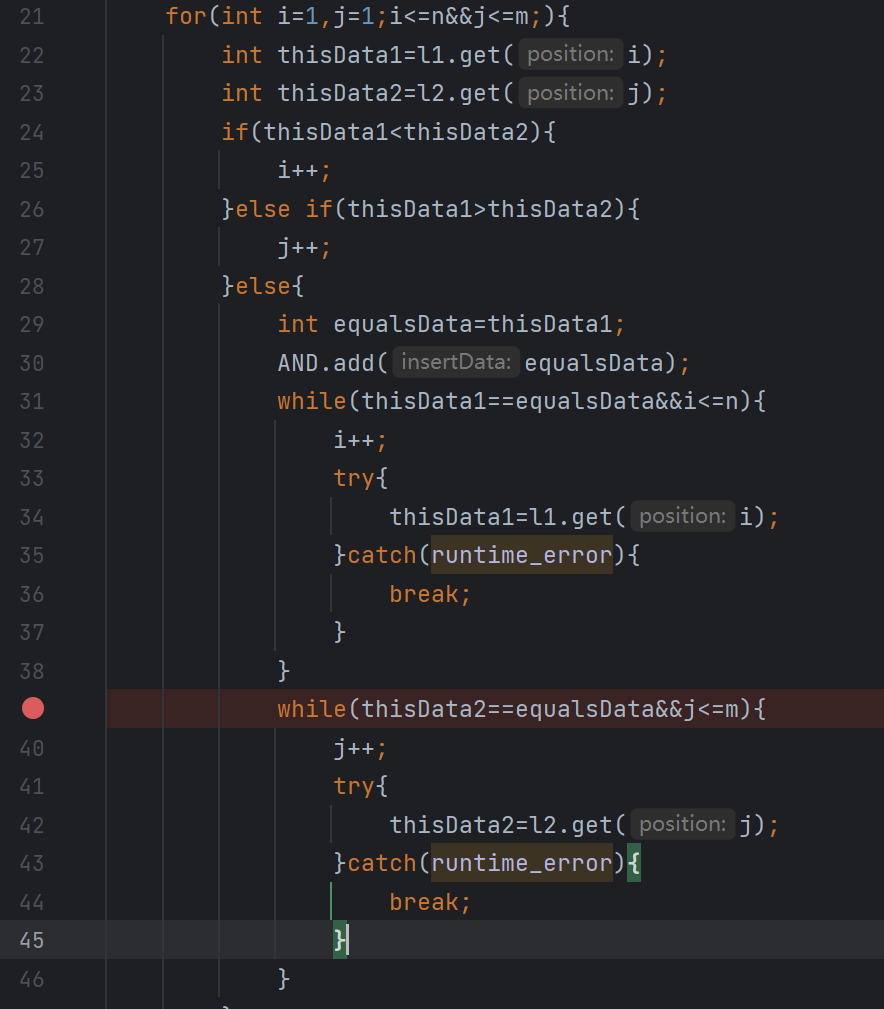
开一个队列，用一个rear指针指向队列下一个位置，用length表示队列的长度，队列的front指针用(rear-length+maxLength)%maxLength表示

4、二叉树

开一个二叉树，中序遍历二叉树并输出元素值，然后判断中序序列是否为严格单调递增序列

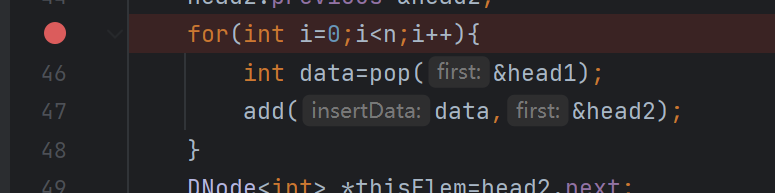
**4、重要算法核心代码**

1、链表



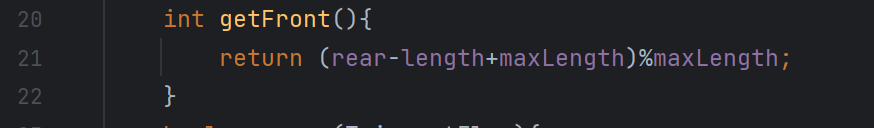
从头到尾遍历整个链表，用两个指针指向下一个要访问的元素thisData1和thisData2，如果1比2大，则2的指针右移，如果2比1大，则1的指针右移，如果1和2相等，则输出该值并将1和2的指针移向下一个比这个数大的值的位置

2、链表/堆栈

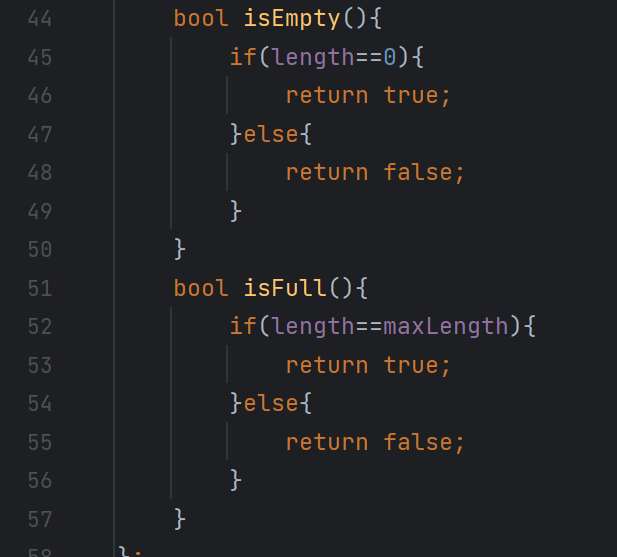


把链表模拟成栈，依次弹出链表的元素并压入另一个链表中

3、队列



计算front指针的方法



判队空队满的方法

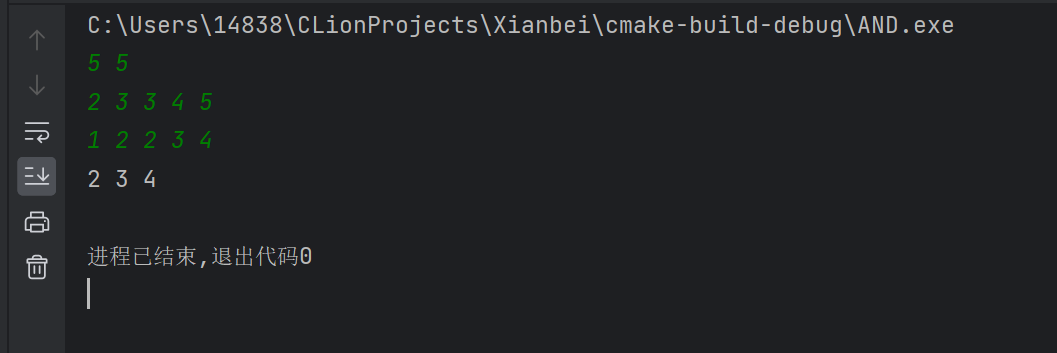
4、二叉树



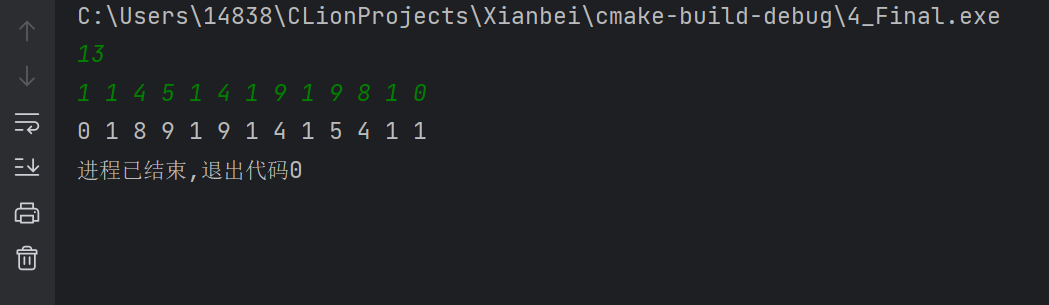
对二叉树中序遍历，依次输出中序遍历结果

**4、测试运行界面**

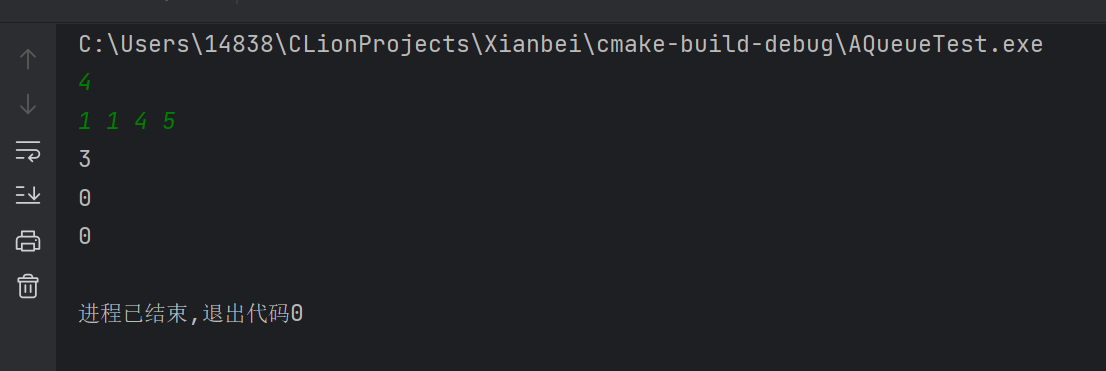
1、链表：



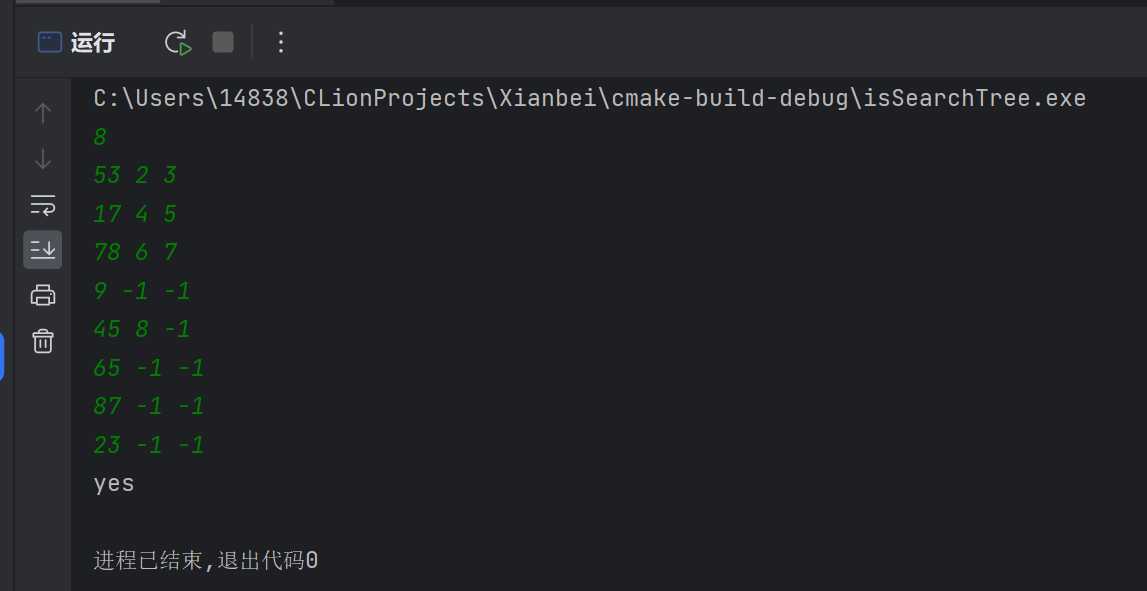
2、链表/队列：

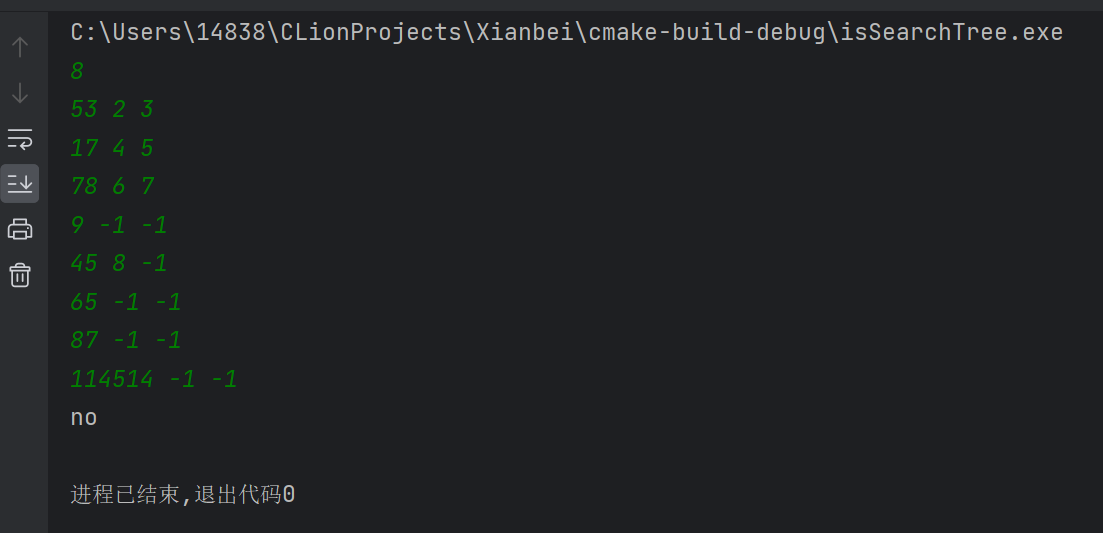


3、队列



4、二叉树





**5、源代码**

1、链表

#pragma once

#ifndef XIANBEI\_DLINKEDLIST\_H

#define XIANBEI\_DLINKEDLIST\_H

#include "iostream"

#include "exception"

template<typename T>

struct DNode{

T data;

DNode<T> \*previous=NULL;

DNode<T> \*next=NULL;

};

template<typename T>

class DLinkedList {

public:

int length;

DLinkedList(){ //构造函数

length=0;

first=new DNode<T>;

}

~DLinkedList(){ //析构函数

clear();

if(first!=NULL){

delete first;

first=NULL;

}

}

void print(){ //打印链表数据

DNode<T> \*thisElem;

thisElem=first->next;

while(thisElem!=NULL){

std::cout<<thisElem->data<<" ";

thisElem=thisElem->next;

}

std::cout<<std::endl;

}

bool add(int position,T &insertData){ //后插操作

if(position>length||position<=0){

return false;

}

DNode<T> \*thisElem=first;

for(int i=0;i<position;i++){

thisElem=thisElem->next;

}

DNode<T> \*insertElem=new DNode<T>;

insertElem->data=insertData;

insertElem->previous=thisElem;

insertElem->next=thisElem->next;

thisElem->next=insertElem;

if(position==length){

thisElem->next->previous=insertElem;

}

length++;

return true;

}

bool add(T insertData){ //尾插操作

DNode<T> \*thisElem=first;

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

thisElem=thisElem->next;

}

DNode<T> \*insertElem=new DNode<T>;

insertElem->data=insertData;

thisElem->next=insertElem;

insertElem->previous=thisElem;

length++;

return true;

}

bool remove(int position){ //给定位置删除

if(position>length||length==0||position<=0){

return false;

}

DNode<T> \*delElem=first;

for(int i=0;i<position;i++){

delElem=delElem->next;

}

if(position==length){

DNode<T> \*previousElem=delElem->previous;

delete delElem;

previousElem->next=NULL;

}else{

DNode<T> \*nextElem=delElem->next;

DNode<T> \*previousElem=delElem->previous;

delete delElem;

previousElem->next=nextElem;

nextElem->previous=previousElem;

}

length--;

return true;

}

bool replace(int position,T &insertData){ //给定位置修改值

if(position>length||length==0||position<=0){

return false;

}

DNode<T> \*replaceElem=first;

for(int i=0;i<position;i++){

replaceElem=replaceElem->next;

}

replaceElem->data=insertData;

return true;

}

bool isEmpty(){ //判空

if(length==0){

return true;

}else{

return false;

}

}

bool clear(){ //清空链表

if(length==0){

return false;

}

DNode<T> \*delElem=first;

while(delElem->next!=NULL){

delElem=delElem->next;

}

while(delElem->previous!=NULL){

delElem=delElem->previous;

delete delElem->next;

}

delElem->next=NULL;

length=0;

return true;

}

T get(int position){

DNode<T> \*getElem=first;

if(position>length||length==0||position<=0){

throw std::runtime\_error("IndexOutOfBounds");

}

for(int i=0;i<position;i++){

getElem=getElem->next;

}

return getElem->data;

}

private:

DNode<T> \*first;

};

#endif //XIANBEI\_DLINKEDLIST\_H

#include "iostream"

#include "DLinkedList.h"

using namespace std;

int main(){

int n,m;

DLinkedList<int> l1;

DLinkedList<int> l2;

cin>>n>>m;

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

int thisData;

cin>>thisData;

l1.add(thisData);

}

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

int thisData;

cin>>thisData;

l2.add(thisData);

}

DLinkedList<int> AND;

for(int i=1,j=1;i<=n&&j<=m;){

int thisData1=l1.get(i);

int thisData2=l2.get(j);

if(thisData1<thisData2){

i++;

}else if(thisData1>thisData2){

j++;

}else{

int equalsData=thisData1;

AND.add(equalsData);

while(thisData1==equalsData&&i<=n){

i++;

try{

thisData1=l1.get(i);

}catch(runtime\_error){

break;

}

}

while(thisData2==equalsData&&j<=m){

j++;

try{

thisData2=l2.get(j);

}catch(runtime\_error){

break;

}

}

}

}

AND.print();

}

2#、链表/堆栈

#include "iostream"

using namespace std;

template<typename T>

struct DNode {

T data;

DNode<T> \*previous = NULL;

DNode<T> \*next = NULL;

};

template<typename T>

bool add(T insertData,DNode<T> \*first){ //尾插操作

DNode<T> \*thisElem=first->previous;

DNode<T> \*insertElem=new DNode<T>;

insertElem->data=insertData;

insertElem->previous=thisElem;

insertElem->next=first;

thisElem->next=insertElem;

first->previous=insertElem;

return true;

}

template<typename T>

T pop(DNode<T> \*first){

T getData=first->previous->data;

DNode<T> \*lastElem=first->previous->previous;

delete lastElem->next;

lastElem->next=first;

first->previous=lastElem;

return getData;

}

int main(){

int n;

cin>>n;

DNode<int> head1;

head1.next=&head1;

head1.previous=&head1;

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

int thisData;

cin>>thisData;

add(thisData,&head1);

}

DNode<int> head2;

head2.next=&head2;

head2.previous=&head2;

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

int data=pop(&head1);

add(data,&head2);

}

DNode<int> \*thisElem=head2.next;

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

cout<<thisElem->data<<" ";

thisElem=thisElem->next;

}

}

3、队列

#ifndef XIANBEI\_AQUEUE\_H

#define XIANBEI\_AQUEUE\_H

#include "iostream"

#include "exception"

template <typename T>

class AQueue{ //都什么年代了，还用传统顺序队列

private:

const static int maxLength=5;

T elem[maxLength];

int rear;

public:

int length;

AQueue(){

rear=0;

length=0;

}

int getFront(){

return (rear-length+maxLength)%maxLength;

}

bool enqueue(T insertElem){

if(isFull()){

return false;

}

elem[rear]=insertElem;

rear=(rear+1)%maxLength;

length++;

return true;

}

T dequeue(){

T popElem=peek();

length--;

int front=getFront();

return popElem;

}

T peek(){

if(isEmpty()){

throw std::runtime\_error("IndexOutOfBounds");

}

return elem[getFront()];

}

bool isEmpty(){

if(length==0){

return true;

}else{

return false;

}

}

bool isFull(){

if(length==maxLength){

return true;

}else{

return false;

}

}

};

#endif //XIANBEI\_AQUEUE\_H

#include "AQueue.h"

#include "iostream"

using namespace std;

int main(){

int n;

cin>>n;

AQueue<int> queue;

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

int thisData;

cin>>thisData;

queue.enqueue(thisData);

}

queue.dequeue();

queue.dequeue();

queue.dequeue();

//找到front指针：

cout<<queue.getFront()<<endl;

//队空条件：length==0

cout<<queue.isEmpty()<<endl;

//队满条件：length==maxLength

cout<<queue.isFull()<<endl;

}

4、二叉树

#include "iostream"

#define MAXN 114514

#define MINN -114514

using namespace std;

class Node{

public:

int id;

int data;

int l=-1;

int r=-1;

Node(int id,int data,int l,int r){

this->data=data;

this->id=id;

this->l=l;

this->r=r;

}

Node(){}

};

int mid[MAXN];

bool isFill[MAXN];

int cnt=0;

Node sTree[MAXN];

void dfs(int thisPosition){

Node thisNode=sTree[thisPosition];

if(thisNode.l!=-1){

dfs(thisNode.l);

}

mid[cnt]=thisNode.data;

isFill[cnt]=true;

cnt++;

if(thisNode.r!=-1){

dfs(thisNode.r);

}

}

int main(){

for(int i=0;i<MAXN;i++){

isFill[i]=false;

}

int n;

cin>>n;

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

int data,l,r;

cin>>data>>l>>r;

sTree[i]=Node(i,data,l,r);

}

dfs(1);

bool isSearchTree=true;

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

if(isFill[i]){

if(mid[i]<=mid[i-1]){

isSearchTree=false;

break;

}

}

}

if(isSearchTree){

cout<<"yes"<<endl;

}else{

cout<<"no"<<endl;

}

}