**实验报告 - N皇后问题**

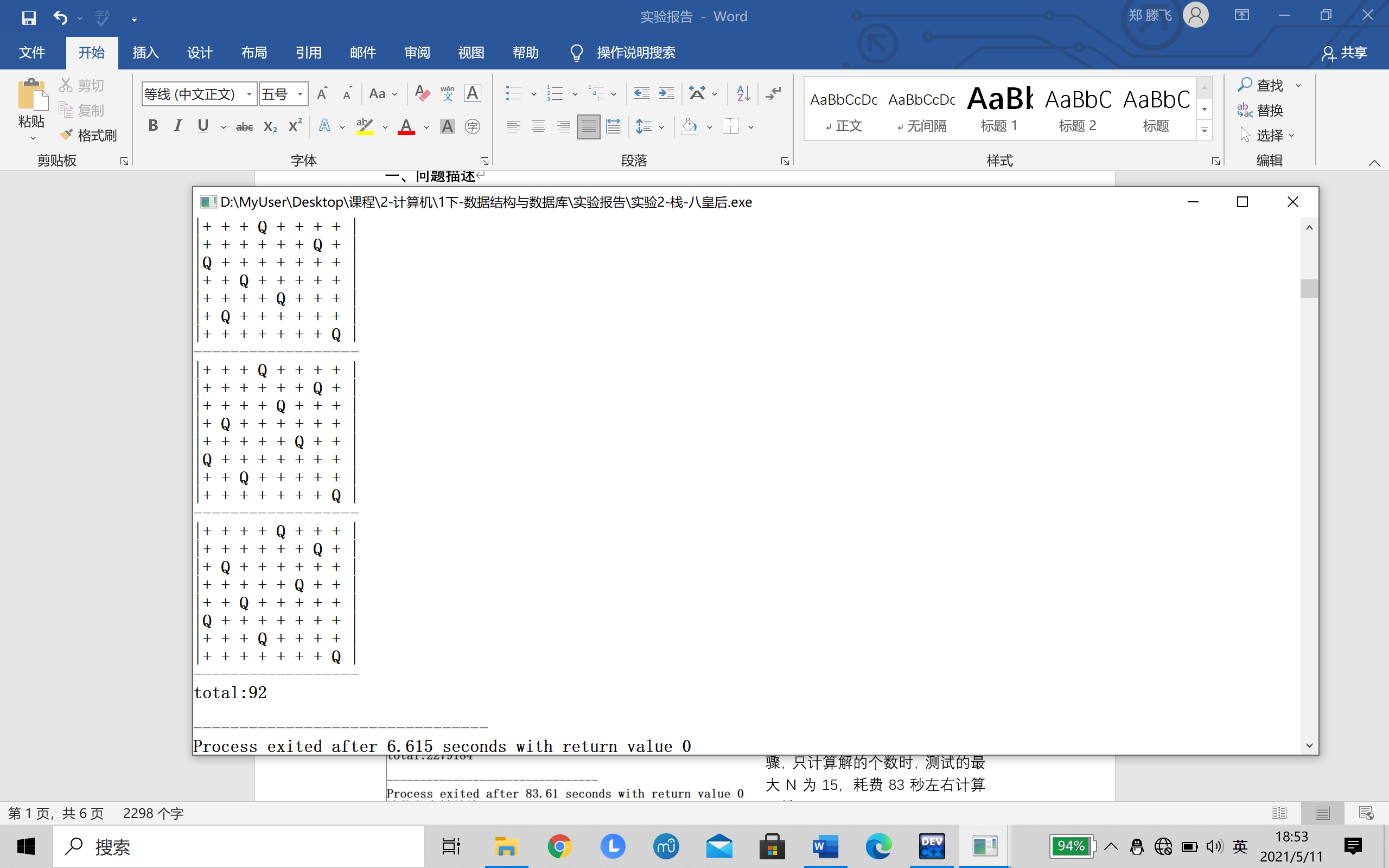
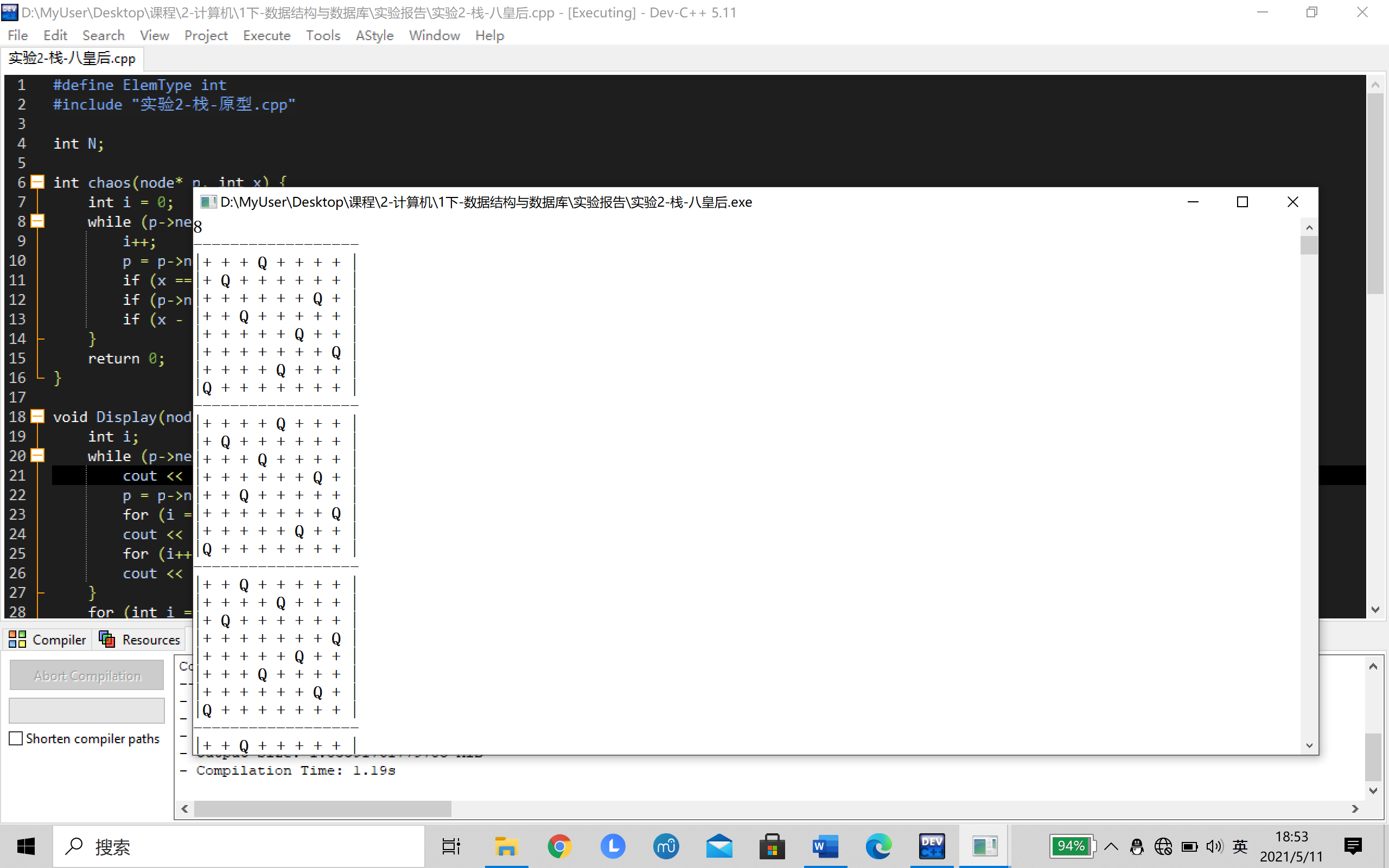
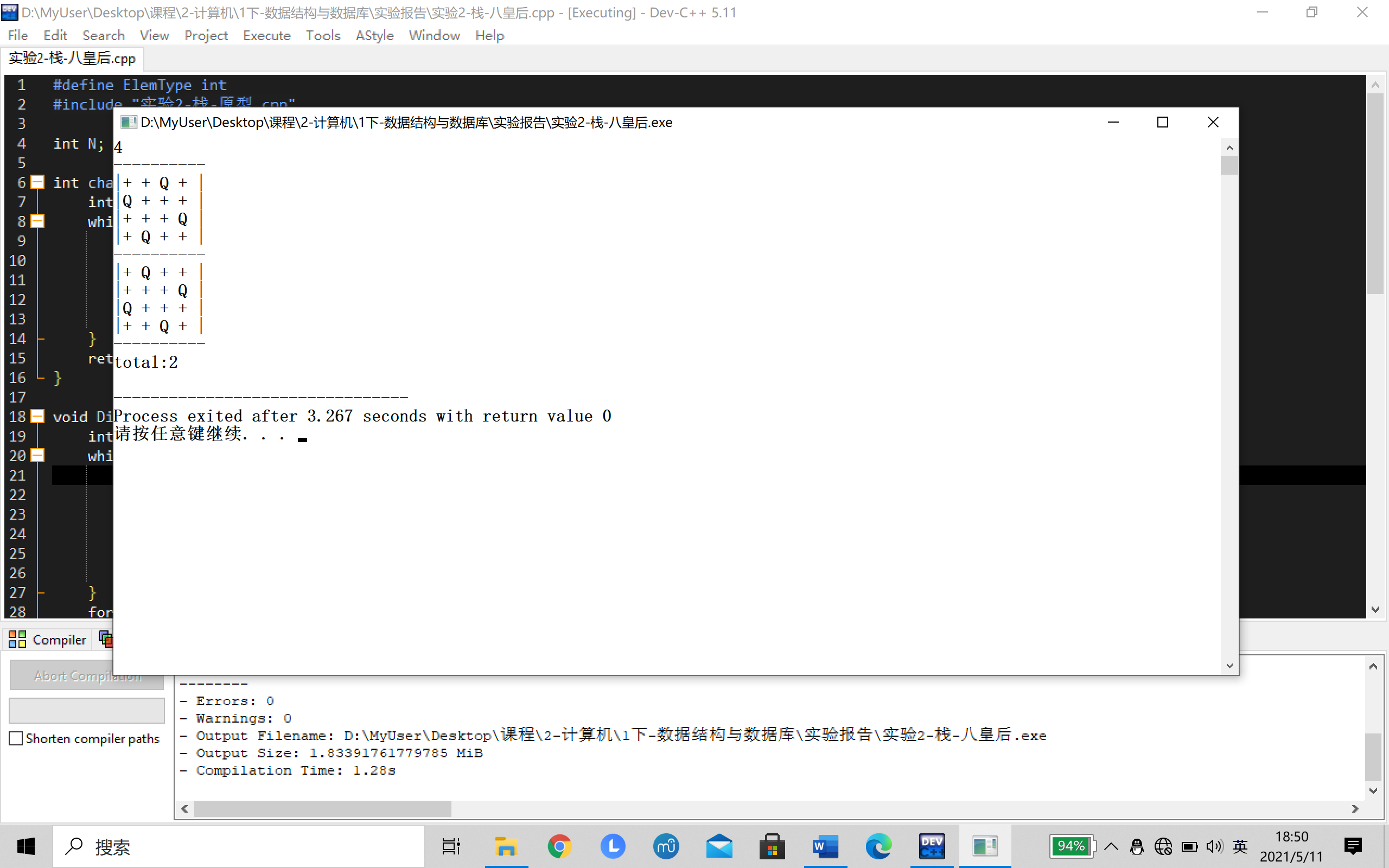
PB20000296 郑滕飞

1. **问题描述**
2. 实验题目：利用链表实现栈的基本操作，并以此解决N皇后问题。

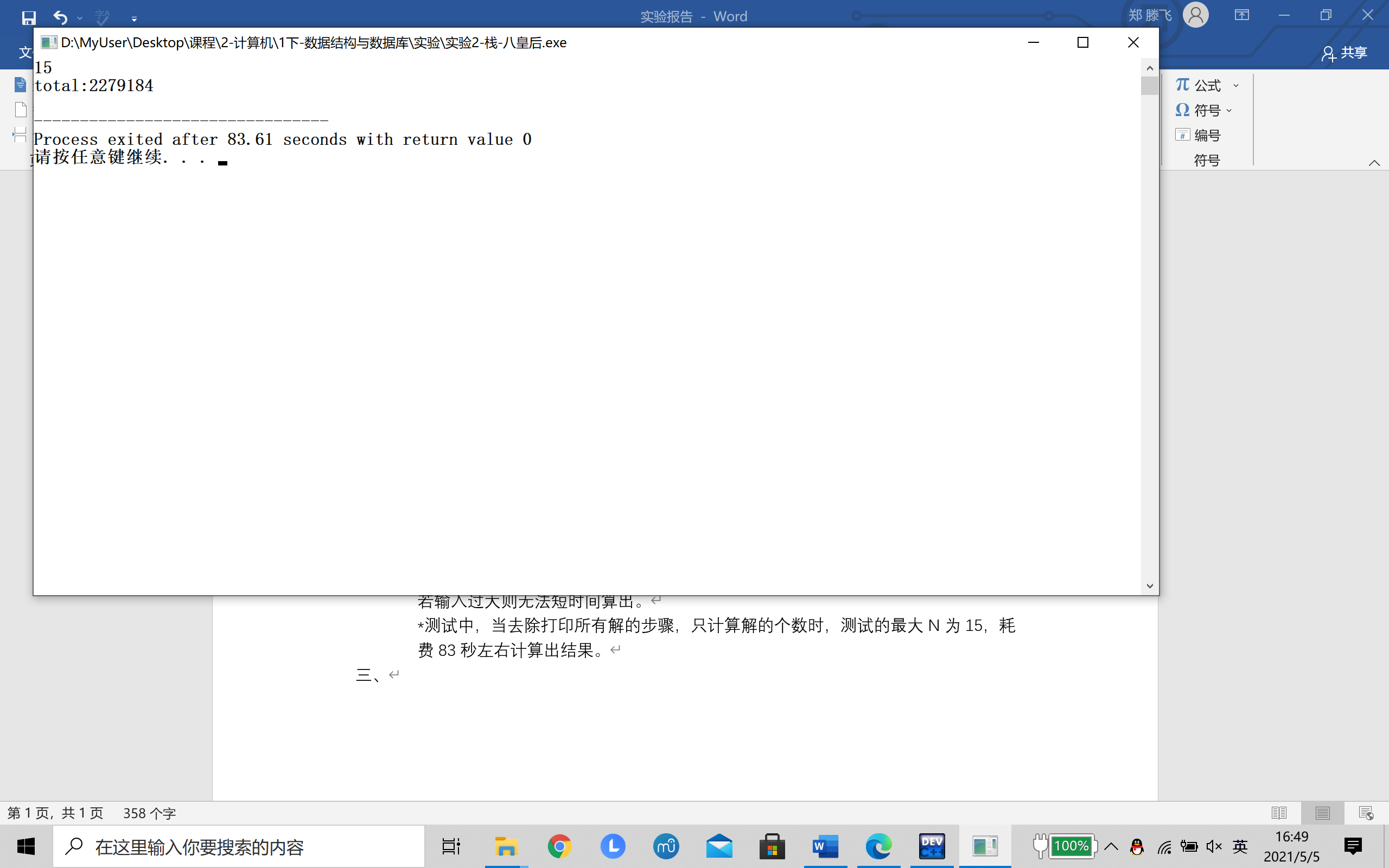
\*八皇后问题：国际象棋的棋盘为8×8的正方形，每个“皇后”可以攻击同行、同列及对角上的全部棋子，要求在棋盘上放置8个皇后，不能互相攻击，问放法的种数。

\*N皇后问题：八皇后问题的拓展，将棋盘改为N×N的正方形，问放置N个皇后的可能方法数。

1. 基本要求：用户输入正整数N，输出N皇后问题的全部解，并统计个数。
2. 测试数据（第一行为输入）（输入8时由于输出过多只展示头部和底部）：



1. **需求分析**
2. 本程序用于计算N皇后问题的全部解及解的个数。
3. 用户理论来说需输入正整数，系统自动校正负数与0的情况（将输出解个数为0），若输入过大则无法短时间算出。
4. 用户输入完毕后，程序自动得出结果。

\*测试中，当去除打印所有解的步骤，只计算解的个数时，测试的最大N为15，耗费83秒左右计算出结果2279184。

1. **概要设计**

为实现上述功能，需要用链表表示栈。由于栈的操作实际为链表操作的简化，直接写出栈的抽象数据类型定义：

**———————————————————————————————————————**

ADT Stack{

数据对象：

数据关系：

基本操作：InitStack(&p); //构造栈

DestroyStack(&p); //销毁栈

Push(&p, a); //元素入栈

Pop(&p, &a); //元素出栈

GetTop(p, &a); //返回栈顶元素

TraverseStack(p); //依次打印栈中所有元素

}ADT Stack

**———————————————————————————————————————**

但是，由于八皇后问题的特殊性，数据类型中的每个元素实际关联之前的所有元素，由此，必须改进栈的数据类型与操作：

**———————————————————————————————————————**

ADT NewStack{

数据对象：ElemType,

数据关系：

基本操作：InitStack(&p); //构造栈

DestroyStack(&p); //销毁栈

Push(&p, a); //元素入栈

Pop(&p, &a); //元素出栈

GetTop(p, &a); //返回栈顶元素

TraverseStack(p); //依次打印栈中所有元素

chaos(p, a); //判定给定元素是否与现有元素冲突

}ADT NewStack

**———————————————————————————————————————**

本程序模块：

主函数模块：用于计算解

打印模块：用于打出所有解

栈模块：实现栈的抽象数据类型

调用关系：

打印模块

主函数模块

栈模块

1. **详细设计**

1、元素类型、结点类型、全局变量建立：

**———————————————————————————————————————**

#define int ElemType

struct node{

ElemType n;

struct node\* next;

};

int N;//由于不止一个模块需使用，将个数N建立为全局变量

**———————————————————————————————————————**

2、栈基本操作：

**———————————————————————————————————————**

int InitStack(node\* &p) {

if (p) return -1; //若已有栈，返回错误

p = new node; //建立头结点

p->next = NULL; //终点置空

return 0;

}//建立空栈

int DestroyStack(node\* &p) {

if (!p) return -1; //判定是否存在

node\* q;

while (p) {

q = p;

p = p->next;

delete q;

}//按顺序删除所有结点

p = NULL; //置空，为下次建栈做准备

return 0;

}//删除栈

int Push(node\* &p, ElemType a) {

if (!p) return -1;

node\* s = new node; //建立新的头节点

p->n = a; //在原头结点处插入新节点

s->next = p;

p = s; //转移头结点

return 0;

}//第一个结点处插入

int Pop(node\* &p, ElemType &a) {

if (!p) return -1;

if (!p->next) return 1; //检测栈是否为空

node\* s = p;

p = p->next;

a = p->n; //取出第一个结点中元素

delete s; //删除原第一个结点

return 0;

}//删除第一个结点

int GetTop(node\* p, ElemType &a) {

if (!p) return -1;

if (!p->next) return 1;

a = p->next->n; //取出第一个结点中元素

return 0;

}//获取第一个元素

int TraverseStack(node\* p) {

if (!p) return -1;

if (!p->next) return 1;

while (p->next) {

p = p->next;

cout << p->n << ' ';

}//按顺序输出所有节点

cout << endl;

return 0;

}//遍历栈

//以下为特殊的函数，以栈中第k个元素代表第k行的皇后所在的列数，以此判定是否冲突

int chaos(node\* p, int x) {

int i = 0;

while (p->next) {//对栈中所有皇后判断

i++;

p = p->next;

if (x == p->n) return 1;//若与新皇后同列，返回“冲突”

if (p->n - x == i) return 1;

if (x - p->n == i) return 1;//若与新皇后共某方向对角线，返回“冲突”

}

return 0;//若均安全，返回“不冲突”

}

**———————————————————————————————————————**

3、打印模块

**———————————————————————————————————————**

void Display(node\* p) {

int i;

while (p->next) {

cout << '|';

p = p->next;

for (i = 0; i < p->n; i++) cout << "+ ";

cout << "Q ";//在每行对应位置打印后

for (i++ ; i < N; i++) cout << "+ ";//补全棋盘

cout << '|' << endl;

}

for (int i = 0; i < 2\*N; i++) cout << '-';//打印分割线

cout << endl;//换行

}

**———————————————————————————————————————**

4、主函数

**———————————————————————————————————————**

int main(void) {

node\* p = NULL;

cin >> N;//输入皇后数

int i = 0, a, b;//i用于存储已放入的行数

long n = 0;//n用于存储总个数

InitStack(p);

for (int i = 0; i < 2\*N + 2; i++) cout << '-';//输出第一行边界

cout << endl;

while (1) {//重复循环

a = 0;

while (chaos(p, a)) a++;

Push(p, a);

i++;//从首位开始向前移动，直到不冲突后放入元素（元素未必在棋盘内）

GetTop(p, b);//只判定末元素。程序保证了除末元素外必然在棋盘内。

if (b < N && i == N) {//若末元素在棋盘内，且行数满足要求，则获得解

Display(p);

n++; //打出解并计数

Pop(p, a);

a++;//从下一个情况开始算起

while (chaos(p, a)) a++;

Push(p, a);//放入新的末行元素

}

while (b >= N) {//若出棋盘

Pop(p, b);//退出此元素

if (Pop(p, b)) {//退出上一个元素，若Pop函数返回非0，说明栈空

cout << "total:" << n << endl;

DestroyStack(p);

return 0;//空栈时，打出结果数，结束程序

}

b++;//从下一种情况开始算起

while (chaos(p, b)) b++;

Push(p, b);//放入新的末行元素

i--;//操作中取出两个放回一个，行数减一

GetTop(p, b);继续判定是否出棋盘

}//直到当前全在棋盘内，结束循环

}

}

**———————————————————————————————————————**

1. **调试分析**
2. 时空复杂度分析

对栈的基本操作中，删除栈、遍历栈与判定是否能放入为，其余为。由此，主程序中的时间复杂度为，但值得注意的是，回溯算法中舍弃了大量的不可能情况，因此实际使用时的计算次数应远小于。

由使用栈的最大个数与阶数相同，空间复杂度即为。

1. 有关回溯

算法的核心是回溯部分，而在试图实现回溯时，一开始始终因为无法控制界而难以完成。由于一开始采取的是一旦越界直接回溯的方式，很难不依靠多重循环解决多次回溯。后来则是想到，可以先不管越界放入结果，再进行判定，每次越界后回溯并前进至下一个情况，便只需两重大循环就可以完成回溯。

1. 有关结果判定

判定一组解为结果，除不越界外即要求行数满足，因此新增变量表示行数，每次入栈加一行，出栈减一行，当行数正确且所有皇后均不越界时，即找到了符合要求的解，可以打印出后进行下一种情况。

1. 有关结束条件

由于首先写出的是找一组解的算法，在实际完成时，需要将一组解改为全部解，除新增变量记录解个数外，需考虑结束的条件。实际结束条件应为，第一行的棋子越界，这说明所有情况已经判断完成，因此，此时第一行棋子被取出，栈为空，即可以输出个数，结束程序。

\*算法保证了栈中的所有元素均不冲突，且除末元素外均在棋盘内，因此每次的核心即为放入元素与末元素的越界检测。

1. **使用说明**

程序运行后用户输入正整数N（建议不大于10），程序将依次打出N皇后问题的全部解并统计解的总个数，在最后显示出。

1. **调试结果**

调试结果已在开头给出。

1. **附录**

源程序文件清单：

实验2-栈-原型.cpp

实验2-栈-八皇后.cpp