**计算机程序设计基础（C++)**

**实验报告**

专业班级：软件工程2402

学 号： 8209240206

姓 名： 王适涵

**实验报告成绩：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验** | **实验一** | **实验二** | **实验三** | **实验四** | **实验五** | **总评** |
| **成绩** |  |  |  |  |  |  |

**批阅教师：**

**实验三 函数**

**一、实验目的**

本实验主要培养、训练学生对函数的理解，要求：

1. 掌握函数的定义、声明的方法；

2. 掌握函数的编写要求；

3. 掌握函数的调用方法；

4. 掌握函数参数的传递方法；

5. 掌握变量的作用域；

6. 掌握多文件编程方法。

**二、实验内容与要求**

1、输入自然数m和n，

（1）求他们的最大公约数（或称最大公因数）。

要求输入、输出在主函数中进行，求公约数由函数实现。

1. 在函数中求最大公约数与最小公倍数。（提示：使用引用参数）

2. 编写程序满足：声明一个函数，判断一个整数是否为素数，使用如下函数头：

bool is\_prime(int num) ,如果num是素数函数返回true，否则返回false；

利用函数is\_prime找出前200个素数，并按每行10个输出：

     2     3      5      7    11    13    17    19    23    29

3、编程实现摄氏温度到华氏温度的转换：

编写一个头文件，包含下面两个函数：

double celsius\_to\_fah(double cel)    //摄氏温度到华氏温度

double fahrenheit\_to\_cels(double fah) //华氏温度到摄氏温度

实现头文件，并编写测试程序，调用函数显示如下结果：

Celsius    Fahrenheit   |   Fahrenheit       Celsius

40.0       105.0        |   120.0            48.89

39.0       102.0        |   110.0            43.33

……       ……        |   ……             ……

31.0        87.8        |   30.0             -1.11

（测试程序为主模块，即main( )函数所在的CPP文件，头文件mytemperature.h只有函数声明；函数定义写在另一CPP文件mytemperature.cpp）

4、创建名为mytriangle.h的头文件，包括：

bool is\_valid(double side1,double side2,double side3)

double\_area(double side1,double side2, double side3)

面积=sqrt(s(s-side1)(s-side2)(s-side3))

其中s=(side1+side2+side3)/2

写测试程序：读取三角形三边长，如输入合法，计算面积，否则输出错误信息。

（测试程序为主模块，即main( )函数所在的CPP文件，头文件mytriangle.h只有函数声明；函数定义写在另一CPP文件mytriangle.cpp）

**3与4选一个完成**

5、猴子吃桃：猴子第一天摘若干桃子，当即吃了一半，还不过瘾，又吃了一个。第二天又将剩下的桃子吃掉一半，又多吃一个，以后每天如此，到第10天，发现只剩最后一个桃子，问，第一天猴子共摘多少桃子（用递归实现）。

**三、实验思考题**

1. 本实验中函数中返回的值为什么与函数类型一致？

C++ 是一种**强类型语言**，它要求每个变量、每个表达式都有明确的数据类型，以便编译器能够在编译阶段进行类型检查。如果函数返回值与声明的返回类型不一致，会导致以下问题：

* **类型不匹配错误**：编译器无法自动识别或转换不兼容的类型。
* **数据丢失**：如果返回值类型不匹配，可能会发生数据截断或丢失。

## 同时****遵循函数调用约定****

C++ 中的函数调用遵循特定的**调用约定**

* 函数在返回时，返回值会被存储在特定的寄存器或内存位置中。
* 返回值类型决定了函数的调用者如何获取和处理返回值。

如果返回值类型与函数声明不一致，会导致调用者无法正确地从寄存器或内存中读取返回值，从而引发**未定义行为**。

1. 本实验中主函数调用函数时采用的是何种传递方式？

在本实验中，主函数在调用函数的时候采取的是三种传递方式中的值传递，值传递的机制则是函数在调用的过程当中接收到的只是一个值的拷贝，对形参的修改不会影响到其实参值。

1. **算法分析，程序结果**

**1.代码如下：**

#include<iostream>

using namespace std;

//设置一个void类型的函数用于求出最大公约数和最小公倍数

void gcdgcm(int a, int b, int& gcd, int& gcm)

{

int i = a;

int j = b;

while (j != 0) {

int temp = j;

j = i % j;

i = temp;

}

gcd = i;//求得最大公约数

gcm = a \* b / gcd;//求最小公倍数

}

int main()

{

int a, b;

cout << "请输入两个自然数a，b:"<< endl;

cin >> a >> b;

int gcd\_result, gcm\_result;

gcdgcm(a, b, gcd\_result, gcm\_result);//调用函数

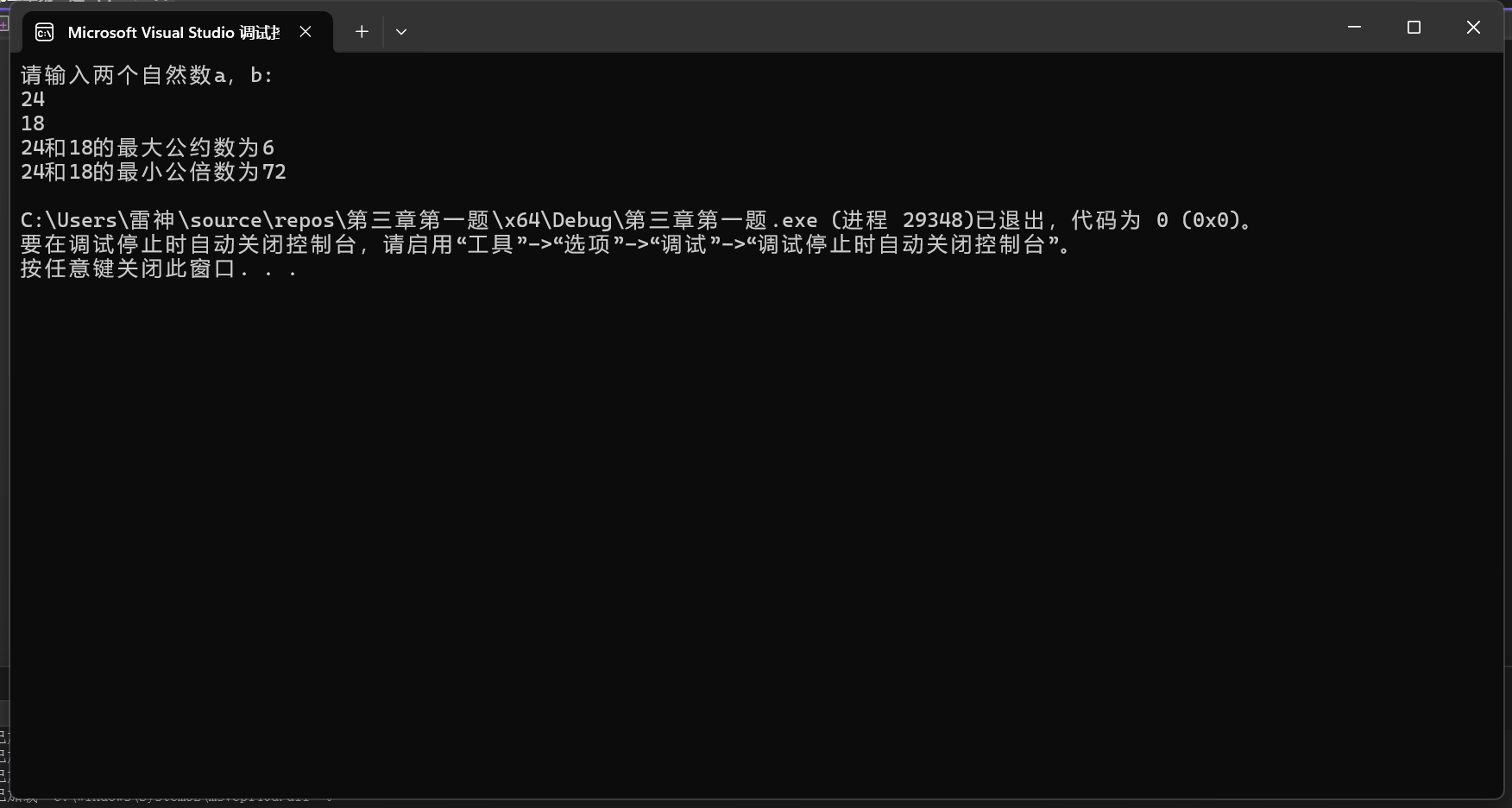
cout << a << "和" << b << "的最大公约数为" << gcd\_result << endl;

cout << a << "和" << b << "的最小公倍数为" << gcm\_result << endl;

return 0;

}

运行结果：



1. 代码如下：

#include <iostream>

#include <iomanip>

using namespace std;

// 判断一个整数是否为素数的函数

bool is\_prime(int num) {

if (num <= 1) {

return false;

}//判断整数为1的时候既不属于素数也不属于合数的情况

if (num <= 3) {

return true;

}//排除2和3均为素数的情况

if (num % 2 == 0 || num % 3 == 0) {

return false;

}

for (int i = 5; i \* i <= num; i += 6) {

if (num % i == 0 || num % (i + 2) == 0) {

return false;

} }

return true;

}

int main() {

int count = 0;

int num = 2;

cout << "前200个素数如下：" << endl;

while (count < 200) {

if (is\_prime(num)) {

cout << setw(6) << num;

count++;

if (count % 10 == 0) {

cout << endl;

}

}

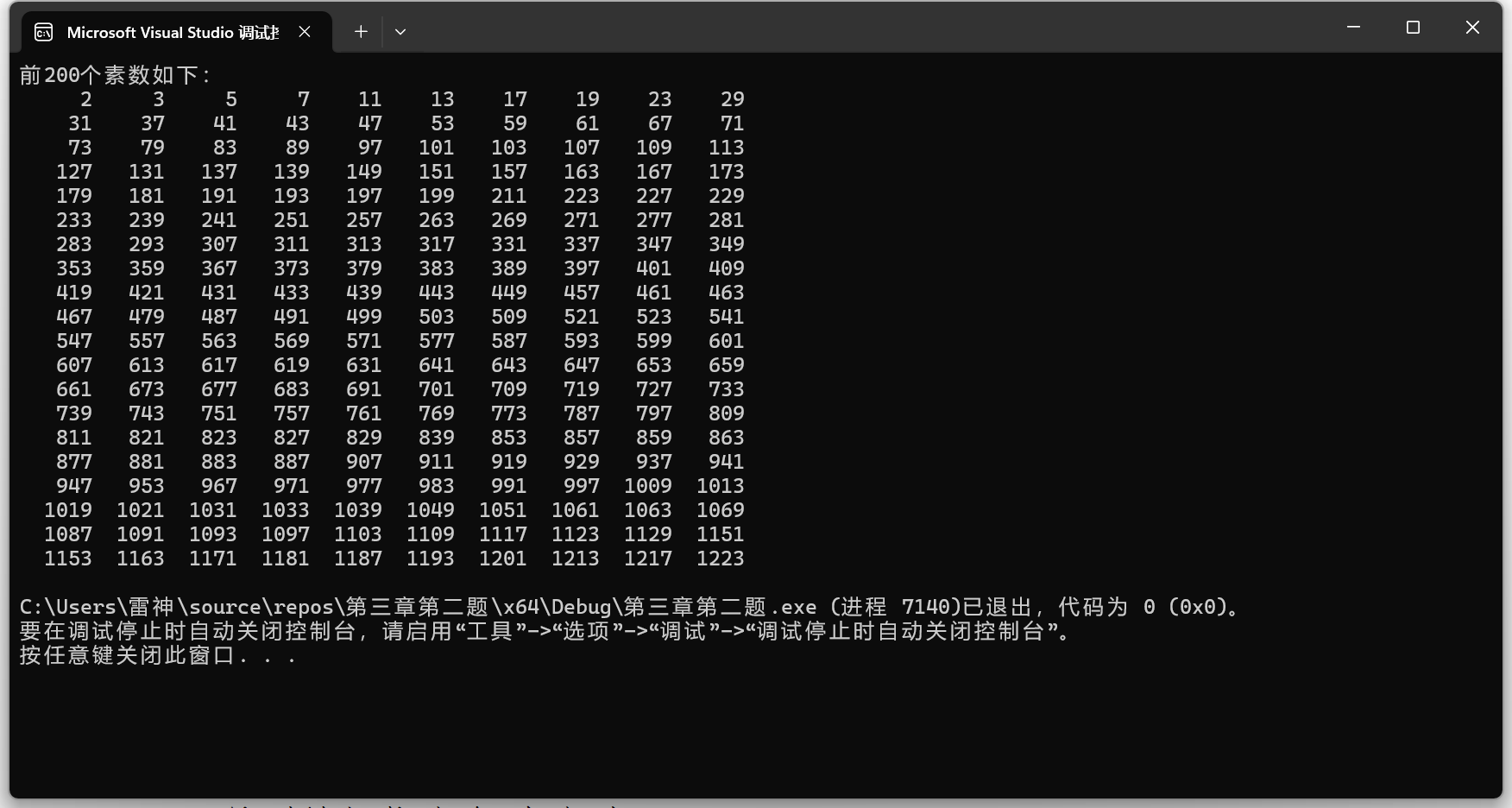
num++;

}

return 0;

}

运行结果：

****

3.（选择了第三题进行做答）

代码如下：

**Mytemperature.h的程序如下**：

#pragma once

#ifndef MYTEMPERATURE\_H

#define MYTEMPERATURE\_H

// 函数声明摄氏温度到华氏温度

double celsius\_to\_fah(double cel);

// 函数声明华氏温度到摄氏温度

double fahrenheit\_to\_cels(double fah);

#endif

**Mytemperature.cpp的程序如下：**

//调用头文件

#include "mytemperature.h"

// 摄氏温度到华氏温度函数的定义

double celsius\_to\_fah(double cel) {

return cel \* 9.0 / 5.0 + 32.0;

}

// 华氏温度到摄氏温度函数的定义

double fahrenheit\_to\_cels(double fah) {

return (fah - 32.0) \* 5.0 / 9.0;

}

**执行程序如下：**

// 主函数的可执行程序

#include <iostream>

#include <iomanip>

#include "mytemperature.h"

using namespace std;

int main() {

cout << fixed << setprecision(2);//在这里是为了保留两位小数

cout << "Celsius Fahrenheit | Fahrenheit Celsius" << endl;

for (double c = 40.0, f = 120.0; c >= 31.0; --c, f -= 10.0) {

double fah = celsius\_to\_fah(c);

double cel = fahrenheit\_to\_cels(f);

cout << setw(8) << c << " " << setw(10) << fah

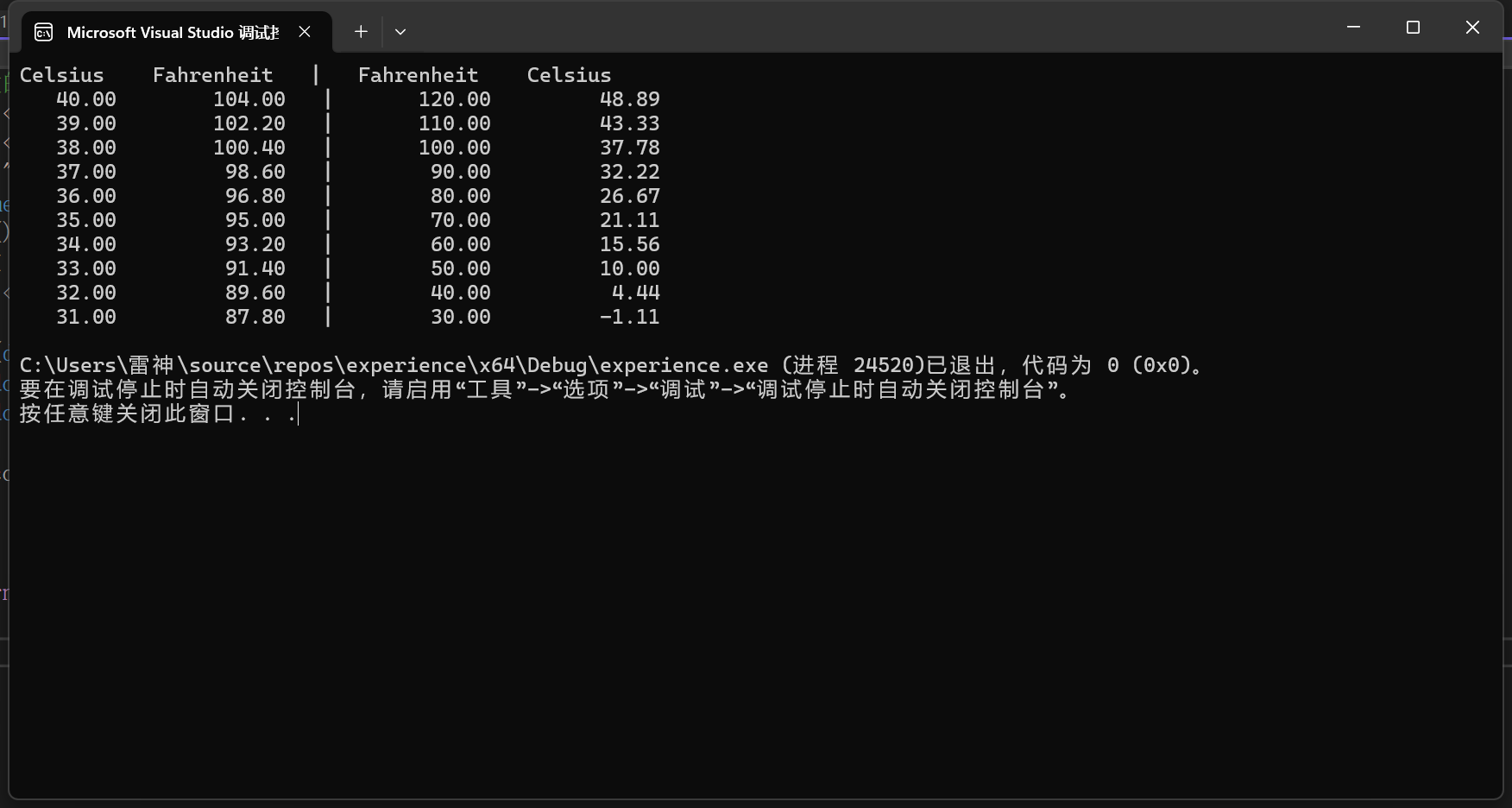
<< " | " << setw(10) << f << " " << setw(10) << cel << endl;//此处设置能够按照题中所给的格式进行输出

}

return 0;

}

运行结果：



5.

代码如下：

#include <iostream>

using namespace std;

// 递归函数，求第n天剩余的桃子数量

int peach(int day) {

if (day == 10) {

return 1; // 第10天剩下1个桃子

}

else {

return 2 \* (peach(day + 1) + 1); // 逆推前一天的桃子数量

}

}

int main() {

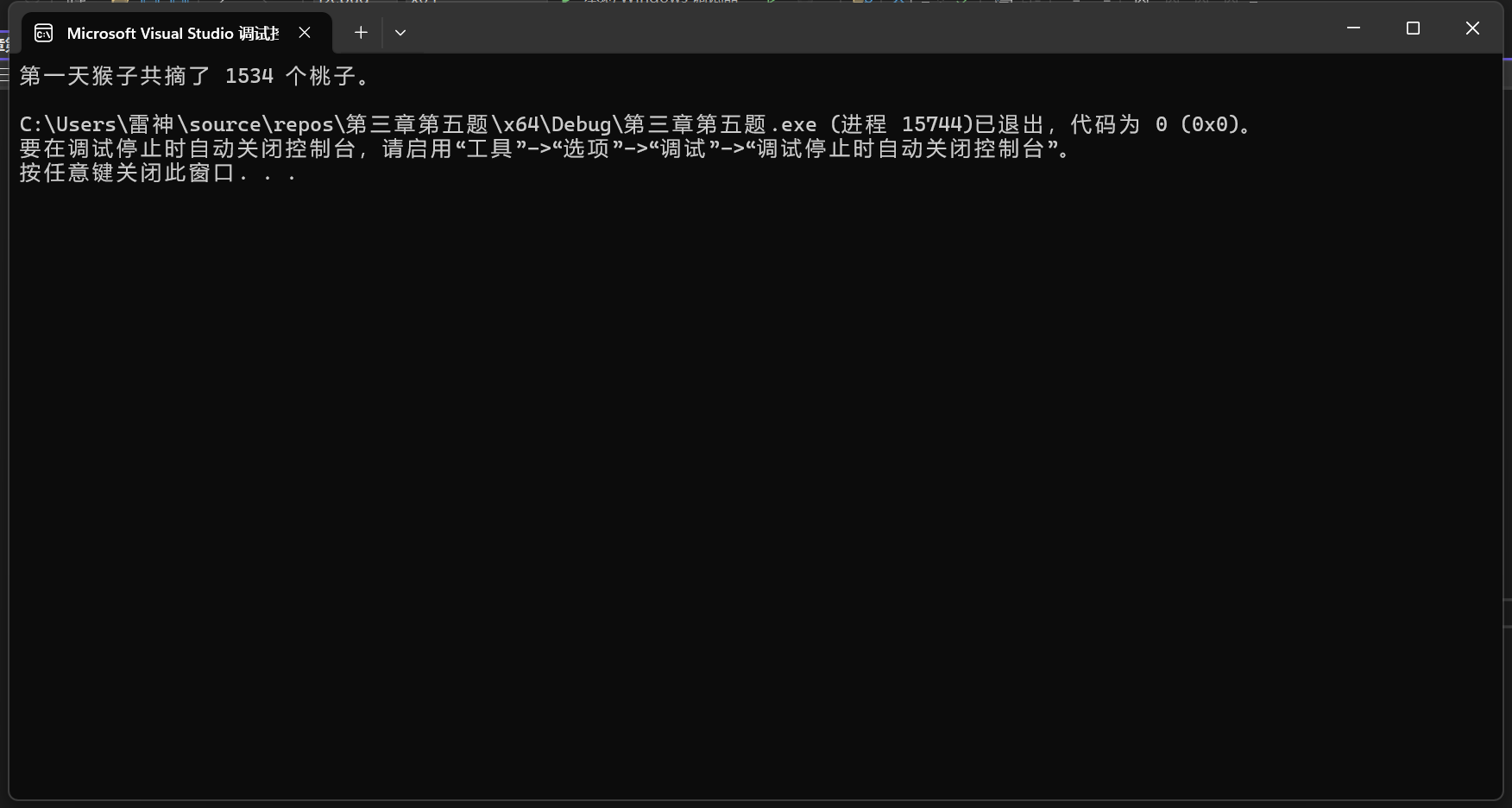
int total\_peaches = peach(1); // 计算第一天的桃子数量

cout << "第一天猴子共摘了 " << total\_peaches << " 个桃子。" <<endl;

return 0;

}

运行结果：



1. **遇到的问题与解决方法**
2. **在编写素数判断函数时，遇到了边界判断不准确的问题**

为了判断一个数是否为素数，需要尝试将它除以比它小的数。不正确的边界条件可能导致不必要的计算或漏掉某些判断。

### 解决方法：

只需要检查 2 到 sqrt(n) 之间的数即可，因为如果一个数 n 不是素数，那么它必然可以分解为两个因数 a 和 b，其中至少有一个小于或等于 sqrt(n)。

**2.在第三题中，可能会遇到头文件重复的提醒导致程序无法运行**

如果没有正确使用头文件保护（如 #ifndef、#define 和 #endif 预处理器指令），可能会导致头文件被多次包含，出现“重复定义”的编译错误。

### 解决方法：

确保头文件中使用了头文件保护（或 #pragma once），这样就能够避免头文件出现头文件重复的编译错误。

且在本题中，如果温度为负数，还需要进行一定的调整使负数的温度也能够正常显示出来问题：

### 解决方法：

确保程序能够正确处理负数。

cout << setw(8) << c << " " << setw(10) << fah

<< " | " << setw(10) << f << " " << setw(10) << cel << endl;

这一串代码就能让温度正常输出，且可以按照题中格式进行输出

**六、体会**

## ****1. 函数的重要性****

函数是程序的基本构建模块。通过学习函数，理解了**代码的模块化和重用性**。将重复的代码封装在函数中，使程序结构更加清晰，也方便了后续的维护和修改。

### 收获：

* 学会了如何将复杂问题分解为多个小问题，每个小问题可以用一个函数来解决。
* 理解了函数可以提高代码的**可读性**和**可维护性**。
* 体会到良好的函数设计能减少代码冗余，提升开发效率。

## ****2. 函数的重载与默认参数****

### 体会：

C++ 支持函数重载和默认参数，这大大提高了函数的灵活性。

### 收获：

* **函数重载**：理解了如何根据不同的参数类型或数量定义多个同名函数，使程序更具扩展性。
* **默认参数**：在某些情况下，可以减少函数调用时传递的参数数量，使代码更简洁。

## ****3. 函数的作用域与生命周期****

### 体会：

学习了函数的局部变量、全局变量和静态变量，理解了它们的作用域和生命周期。

### 收获：

* **局部变量**：只在函数内部生效，函数结束后变量被销毁，内存被释放。
* **全局变量**：可以在整个程序中访问，但滥用全局变量会导致代码难以维护。
* **静态变量**：具有函数内的局部作用域，但生命周期是整个程序运行期间。

通过学习 C++ 中的函数，深刻体会到函数是编程的**核心基础**，掌握函数的定义、调用、重载、递归、作用域等知识，不仅有助于写出**高效**、**简洁**的代码，更帮助我们理解**面向对象**和**模块化编程**的重要思想。在学习过程中，最大的体会是：**良好的函数设计是编写高质量代码的关键。**

**实验四 数组与指针**

【**实验目的**】

1、进一步加深对数组的理解，掌握数组的定义方法；

2、掌握数组的处理方法、数组作为函数参数的使用方法，以及搜索与排序的应用。

3、掌握指针的概念、指针变量定义格式以及指针的运算；

4、掌握指针与数组、函数的关系；

5、理解内存动态分配的含义、熟练掌握内存动态分配方法；

6、掌握递归函数的定义方法。

【实验内容与步骤】

**（一）数组**

1、打印不同的数：

编写一个程序，读入10个数，输出其中不同的数（即如果一个数出现多次，只打印一次）。

提示：读入的数如果是一个新的值，则将其存入一个数组。否则，将其丢弃。输入完毕后，数组中保存的就是不同的数。

下面是一个运行样例：

Enter

Enter ten numbers: 1 2 3 2 1 6 3 4 5 2

The distinct numbers are: 1 2 3 6 4 5

2、起泡排序：

利用起泡排序算法编写一个排序函数。起泡排序算法分若干趟对数组进行处理。每趟处理中，对相邻元素进行比较。若为降序，则交换；否则，保持原顺序。此技术被称为起泡排序（bubble sort）或下沉排序（sinking sort），因为较小的值逐渐地“冒泡”到上部，而较大值逐渐下沉到底部。

算法可描述如下：

bool changed = true;

do

{

changed = false;

for (int j = 0; j < listSize – 1; j++)

if (list[j] > list[j+1])

{

swap list[j] with list[j+1];

changed = true;

}

} while (changed);

很明显，循环结束后，列表变为升序。容易证明do循环最多执行listSize – 1次。

编写测试程序，读入一个含有10个双精度数字的数组，调用函数并显示排列后的数字。

3、游戏：存物柜问题：

一个学校有100个存物柜，100个学生。开学第一天所有存物柜都是关闭的。第一个学生（记为S1）来到学校后，打开所有的存物柜。第二个学生S2，从第二个存物柜（记为L2）开始，每隔两个存物柜，将它们关闭。第三个学生S3从第三个存物柜L3开始，每隔三个，将它们的状态改变（开着的关上，关着的打开）。学生S4，从L4开始，每隔四个改变它们的状态。学生S5，从L5开始，每隔五个改变状态。依此类推，直至学生S100改变L100的状态。

当所有学生完成这个过程，那些存物柜是开着的？编写一个程序求解此问题，显示所有开着的柜子号码，号码之间用一个空格隔开。

提示：使用一个100个布尔型元素的数组，每个元素代表存物柜是开（true）或关（false）。最初所有的储物柜都是关闭的。

4、合并两个排列好的数组：

编写如下函数，合并两个排列好的数组，形成一个新的排列好的数组。

void merge(const int list1[], int size1, const int list2[], int size2, int list3[])

使用size1+size2次比较实现函数。编写测试程序，提示用户输入两个排列好的数组，并显示合并以后的数组。下面是一个运行样例。注意，输入数据的第一个数字是数组的元素数，而不是数组的一部分。假定数组大小不超过80。

Enter

Enter

Enter list1: 5 1 5 16 61 111

Enter list1: 4 2 4 5 6

The merged list is 1 2 4 5 5 6 16 61 111

5、检验子串：

编写如下函数，检验C字符串s1是否是C字符串s2的子串。如果匹配，返回s1在s2中的下标，否则返回–1。

int indexOf(const char s1[], const char s2[])

编写测试程序，读入两个C字符串，检验C字符串s1是否是C字符串s2的子串。下面是程序的运行样例：

Enter

Enter

Enter

Enter the first string: welcome

Enter the second string: We welcome you!

indexOf(“welcome”, “We welcome you!”) is 3

Enter

Enter the first string: welcome

Enter the second string: We invite you!

indexOf(“welcome”, “We invite you!”) is –1

6、字符串中每个字母出现的次数：

请使用如下函数头编写函数，数出字符串中每个字母出现的次数。

void count(const char s[], int counts[])

counts是一个有26个元素的整数数组。const[0]，const[1]，…，const[25]分别记录a，b，…，z出现的次数。字母不分大小写，例如字母A和字母a都被看作a。

编写测试程序，读入字符串并调用count函数，显示非零的次数。下面是程序的一个运行样例：

Enter

Enter a string: Welcome to New York!

c: 1 times

e: 3 times

k: 1 times

l: 1 times

m: 1 times

n: 1 times

o: 3 times

r: 1 times

t: 1 times

w: 2 times

y: 1 times

**（二）指针**

1、上机验证下列程序的运行结果（有错误的话自己补充完善）

(1) void main()

｛

int i,j,\*pi,\*pj; //此处的\*表示定义指针变量，而非间接运算符

pi=&i;

pj=&j;

i=5;j=7;

cout<<i<<’\t’<<j<<’\t’<<pi<<’\t’<<pj;

cout<<&i<<’\t’<<\*&i<<’\t’<<&j<<’\t’<<\*&j;

}

运行结果：

上述结果中，pi与&i,pj与&j是地址值，随编译程序而变化，不确定。

(2) int main() //C语言程序，要了解

{

int a[]={1,2,3};

int \*p,i;

p=a; //将数组a首地址送给p

for (i=0;i<3;i++)

printf("%d,%d,%d,%d\n",a[i],p[i],\*(p+i),\*(a+i)); //与cout功能差不多

}

运行结果：

1,1,1,1

2,2,2,2

3,3,3,3

通过这两道题目，希望学生掌握数组元素与指向数组的指针的不同。

a[i]表示数组中下标为i的元素。

a[i]←p[i]←\*(p+i)←\*(a+i)

a是数组名，表示数组首地址，(p+i)表示数组中第i个元素的地址，\*(p+i) 相当于a[i]。

(3)通过如下的问题理解递归函数的定义与调用（递归未讲，可以后做）

//#include “stdio.h”

void f(char \*st,int i)

{

st[i]=’\0’;

cout<<st; // printf(“%s\n”,st);

if (i>1) f(st,i-1);

}

void main()

{

char st[]=”abcd”;

f(st,4);

}

补充完整，运行时输出为\_\_\_\_\_\_\_\_

(4)下面程序的主函数中能保证p[0]输出1，p[1]输出2吗？如何修改以保证之（提示：在函数f中使用new生成动态数组；在main中用delete释放。）

#include<iostream>

using namespace std;

int \*f()

{

int list[]={1,2,3,4};

return list;

}

void main()

{

int \*p=f();

cout<<p[0]<<endl;

cout<<p[1]<<endl;

}

2、程序设计

(1)编写函数检查字符串s1是否为字符串s2的子串，若是，返回第一次匹配的下标，否则返回-1。在主程序中输入字符串s1与s2，调用函数实现。

函数原型：int indexof(const char \*s1,const char \*s2);

(2)编写一个函数将以字符串形式表示的一个16进制数转换为10进制数，并在主函数中测试。函数原型 int parseHex(const char \*const hexString);

如：调用函数 parseHex(“A5”);返回165

1. 主程序中建立一动态数组（使用new），数组元素及元素个数由键盘输入，动态调试观察指针及指针指向的内容；设计一个函数对数组由小到大排序；主程序中用指针方式输出数组元素；最后释放数组内存（delete）。

【完成实验报告】

**实验报告只要求写程序设计部分**

1. **算法分析，程序结果**
2. **数组：**
3. **代码如下:**

#include <iostream>

#include <set>

using namespace std;

int main() {

set<int> uniqueSet;

int num;

cout << "Enter ten numbers:" << endl;

for (int i = 0; i < 10; ++i) {

cin >> num;

uniqueSet.insert(num); // 去重

}

cout << "The distict numbers are:" << endl;

for (const auto& n : uniqueSet) {

cout << n << " ";

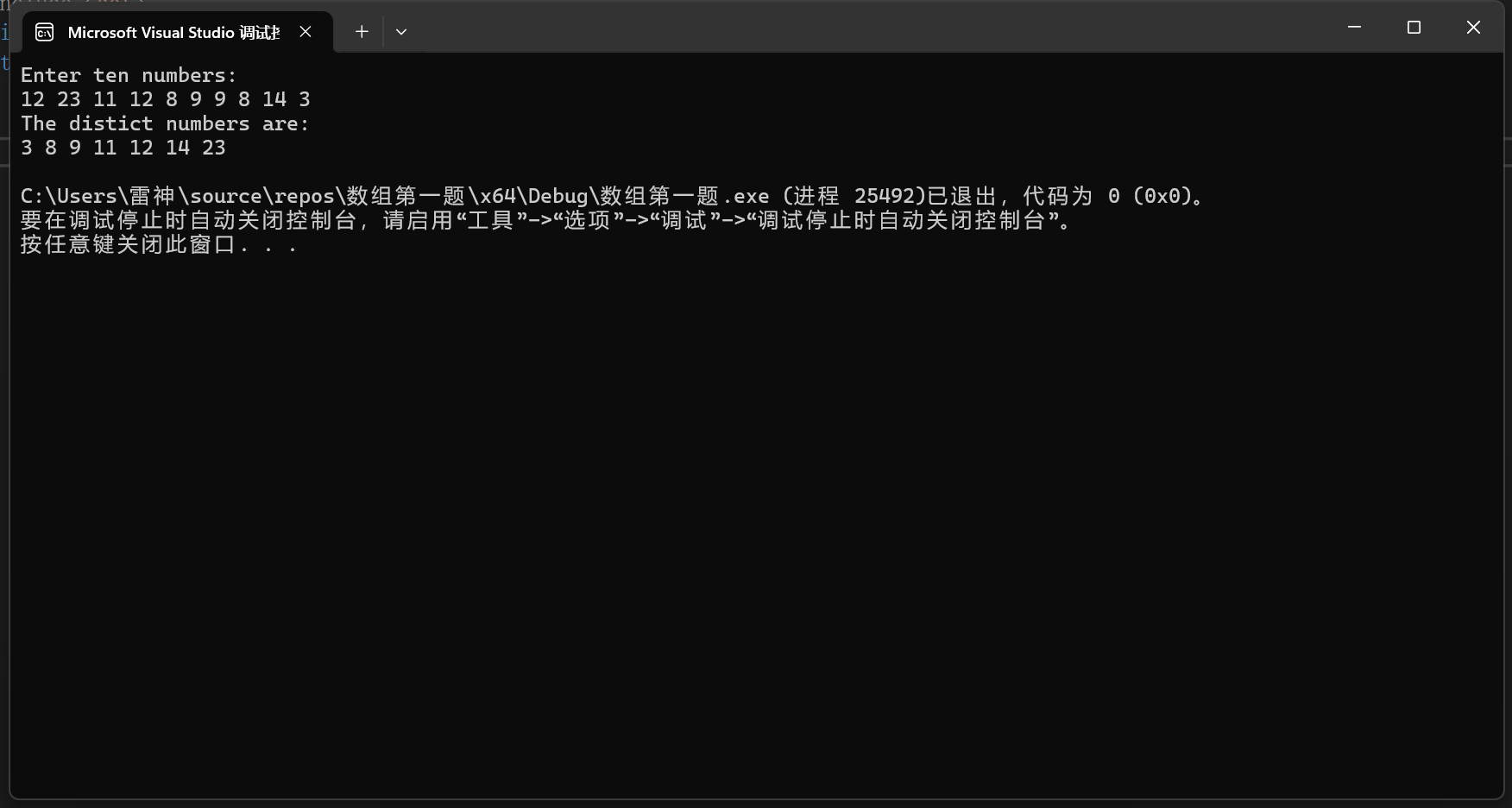
}

cout << endl;

return 0;

}

运行结果：



1. 代码如下：

#include <iostream>

using namespace std;

// 交换两个元素的值的函数

void swap(double& a, double& b) {

double t = a;

a = b;

b = t;

}

// 起泡排序函数

void bubbleSort(double list[], int listSize) {

bool changed = true;

do {

changed = false;

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

if (list[j] > list[j + 1]) {

swap(list[j], list[j + 1]);

changed = true;

}

}

} while (changed);

}

int main() {

const int arraySize = 10;

double numbers[arraySize];

// 读入10个双精度数字

cout << "请输入10个双精度数字：" << endl;

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

cin >> numbers[i];

}

// 调用起泡排序函数

bubbleSort(numbers, arraySize);

// 显示排列后的数组

cout << "排序后的数组：" << endl;

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

cout << numbers[i] << " ";

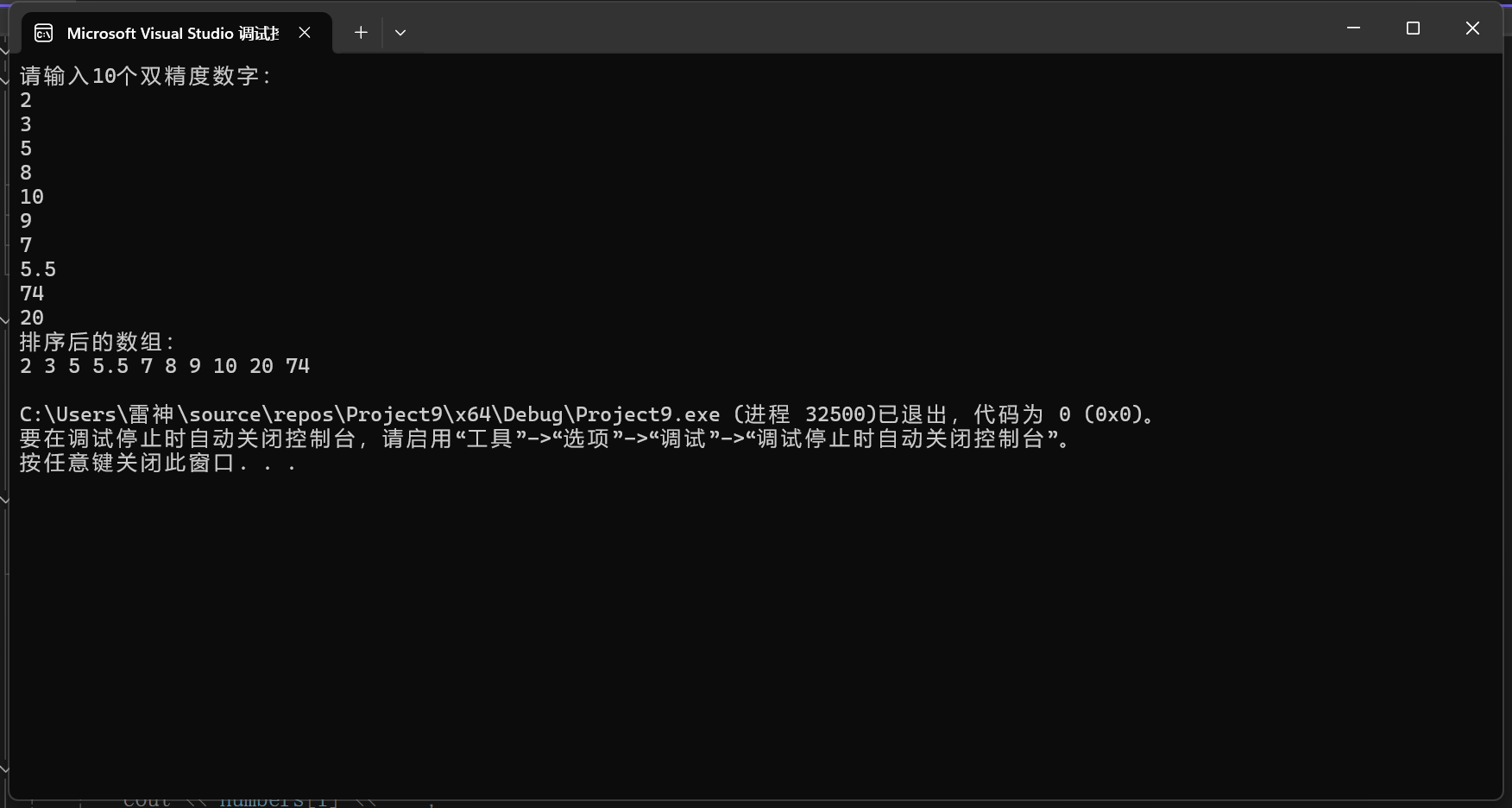
}

cout << endl;

return 0;

}

运行结果：



1. 代码如下：

#include<iostream>

using namespace std;

int main()

{

//设置一个布尔类型数组判断储物柜的开闭情况

bool arr[100] = { false };//这里用于固定储物柜为关闭状态

for (int student = 1; student <= 100; ++student) {

for (int i = student - 1; i < 100; i += student) {

arr[i] = !arr[i];

}

}

for (int i = 0; i < 100; ++i) {

if (arr[i]) {

cout << (i + 1) << " ";//输出对应的编号

}

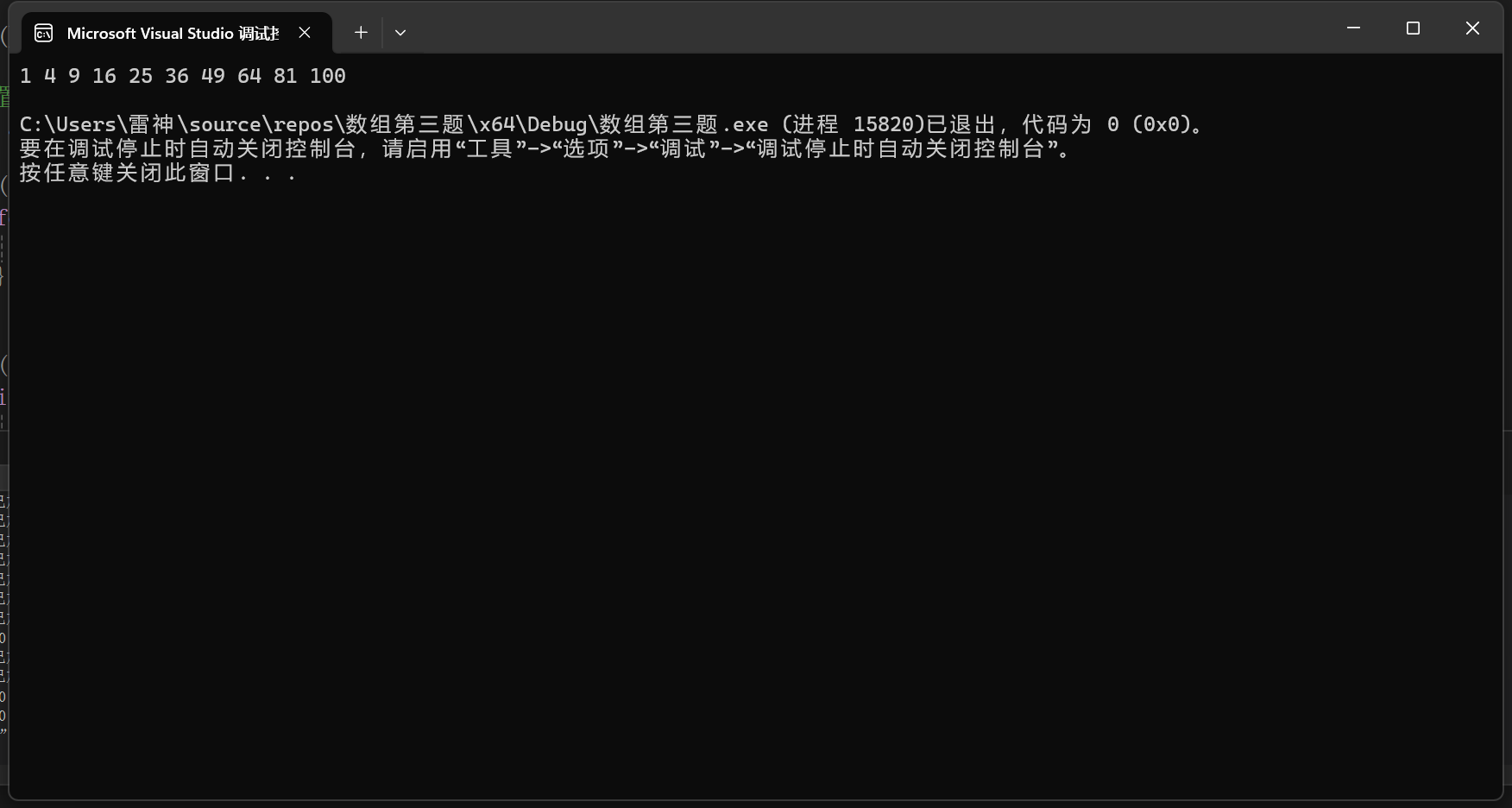
}

cout << endl;

return 0;

}

运行结果：



4.

代码如下：

#include <iostream>

using namespace std;

// 合并两个已排序数组的函数

void merge(const int list1[], int size1, const int list2[], int size2, int list3[]) {

int i = 0, j = 0, k = 0;

// 比较两个数组的元素并将较小的放入新数组list3中

while (i < size1 && j < size2) {

if (list1[i] < list2[j]) {

list3[k++] = list1[i++];

}

else {

list3[k++] = list2[j++];

}

}

// 如果list1还有剩余元素，将它们添加到list3

while (i < size1) {

list3[k++] = list1[i++];

}

// 如果list2还有剩余元素，将它们添加到list3

while (j < size2) {

list3[k++] = list2[j++];

}

}

int main() {

const int MAX\_SIZE = 80;

int list1[MAX\_SIZE], list2[MAX\_SIZE], list3[MAX\_SIZE \* 2];

int size1, size2;

// 输入第一个已排序数组

cout << "Enter the size of the first sorted array: ";

cin >> size1;

cout << "Enter the first sorted array: ";

for (int i = 0; i < size1; ++i) {

cin >> list1[i];

}

// 输入第二个已排序数组

cout << "Enter the size of the second sorted array: ";

cin >> size2;

cout << "Enter the second sorted array: ";

for (int i = 0; i < size2; ++i) {

cin >> list2[i];

}

// 合并两个数组

merge(list1, size1, list2, size2, list3);

// 输出合并后的数组

cout << "The merged array is: ";

for (int i = 0; i < size1 + size2; ++i) {

cout << list3[i] << " ";

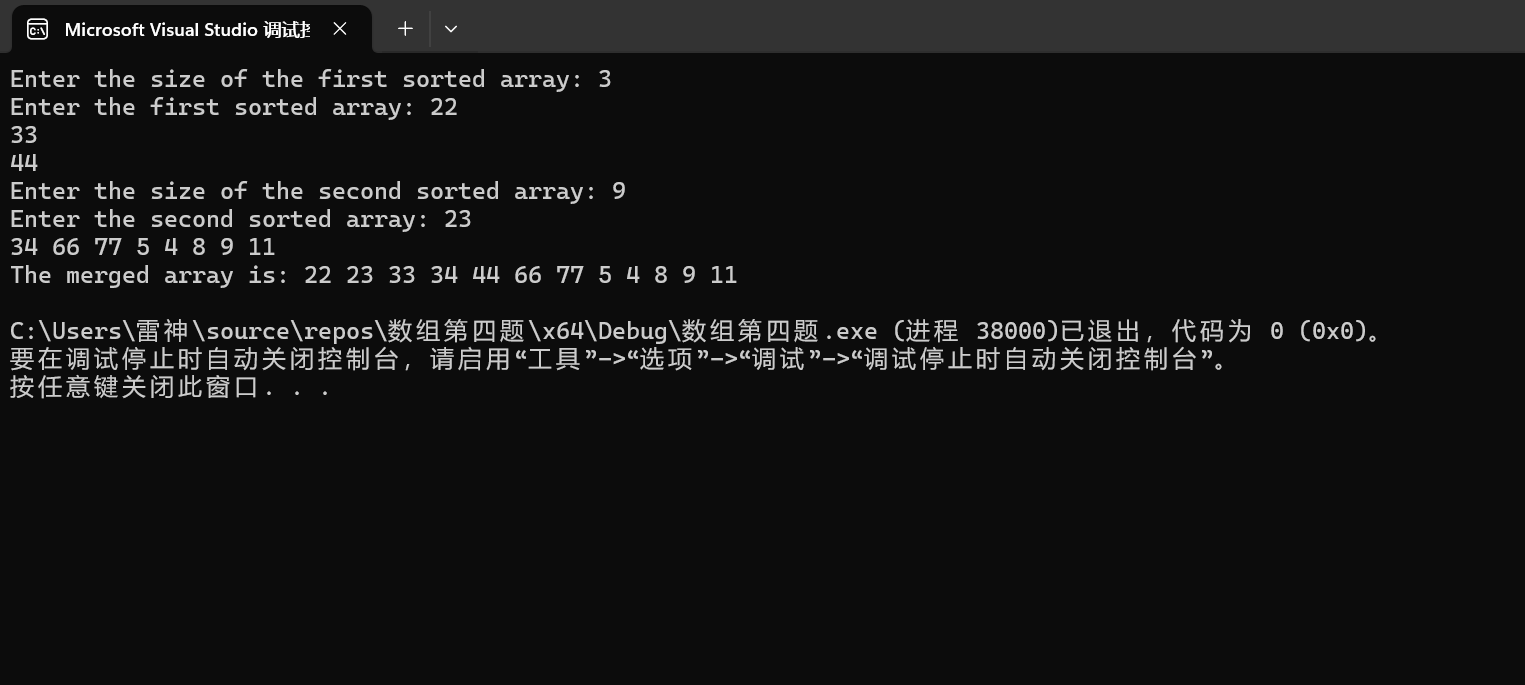
}

cout << endl;

return 0;

}

运行结果:



5.代码如下：

#include <iostream>

#include <string>

using namespace std;

// 函数用于检验s1是否为s2的子串

int indexOf(const string& s1, const string& s2)

{

size\_t pos = s2.find(s1);

if (pos != string::npos) {

return static\_cast<int>(pos);

}

else {

return -1;

}}

int main()

{

string s1, s2;

cout << "Enter the first string";

cin >> s1;

cout << "Enter the second string";

cin >> s2;

int index = indexOf(s1, s2);

// 输出结果

if (index != -1) {

cout << "indexOf(\"" << s1 << "\",\"" << s2 << "\") is " << index << endl;

}

else {

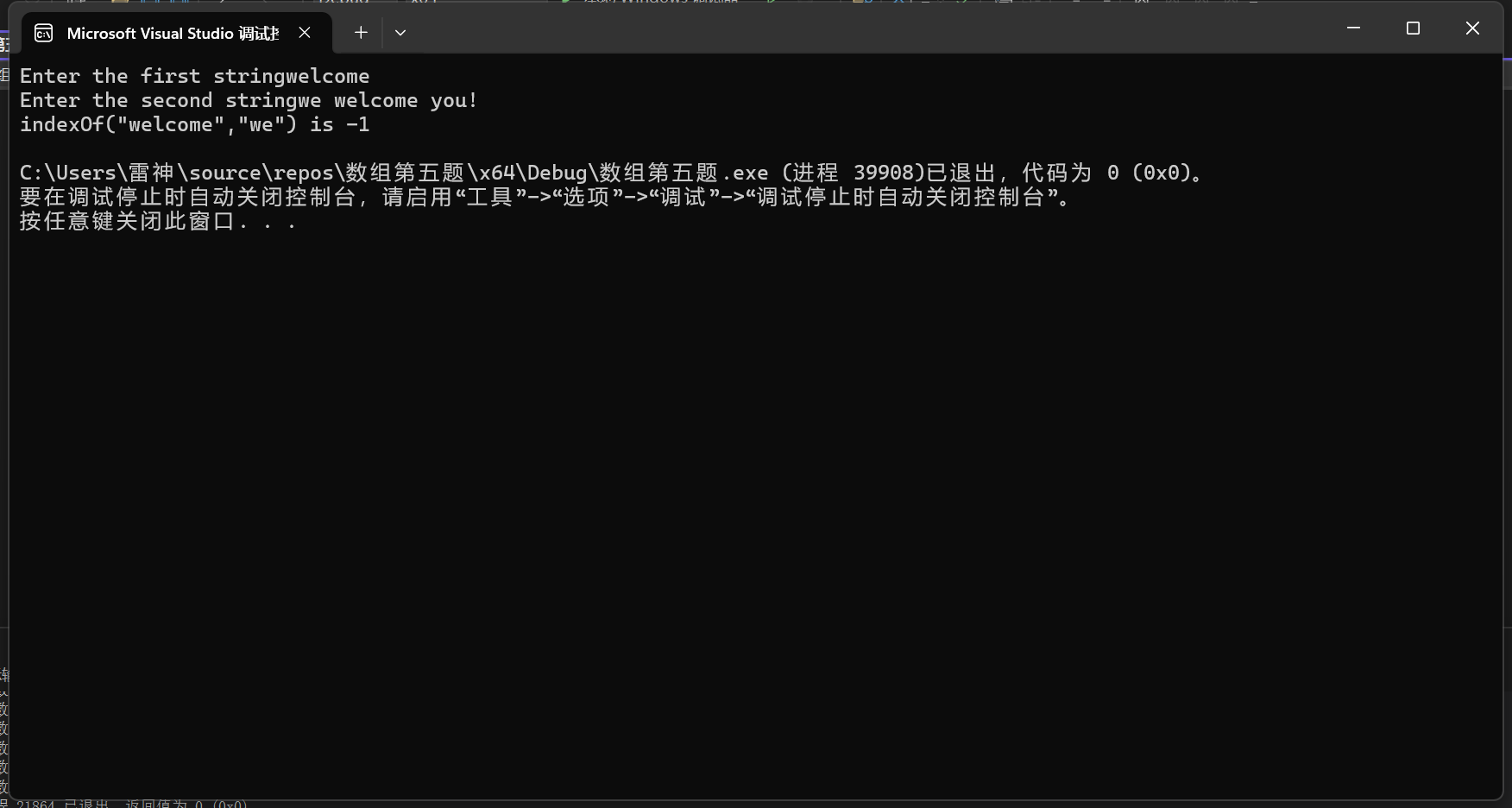
cout << "indexOf(\"" << s1 << "\",\"" << s2 << "\") is -1" << endl;

}

return 0;

}

运行结果：



1. **代码如下：**

#include <iostream>

#include <cctype>

// 函数定义，用于统计字符串中每个字母出现的次数

void count(const char s[], int counts[]) {

// 初始化counts数组元素为0

for (int i = 0; i < 26; ++i) {

counts[i] = 0;

}

int i = 0;

while (s[i] != '\0') {

char ch = std::tolower(s[i]);

if (ch >= 'a' && ch <= 'z') {

// 统计对应字母出现的次数

counts[ch - 'a']++;

}

++i;

}

}

int main() {

char inputString[1000]; // 假设输入字符串长度不超过1000

int counts[26];

std::cout << "请输入一串字符串：";

std::cin.getline(inputString, sizeof(inputString));

count(inputString, counts);

// 输出每个字母出现的次数

for (int i = 0; i < 26; ++i) {

if (counts[i] > 0) {

std::cout << "字母 " << static\_cast<char>('a' + i) << " 出现的次数：" << counts[i] << std::endl;

}

}

return 0;

}

**运行结果：**

1. **指针**

**（1）**

1. **代码如下：**

#include<iostream>

using namespace std;

void main()

{

int i, j, \* pi, \* pj; //此处的\*表示定义指针变量，而非间接运算符

pi = &i;

pj = &j;

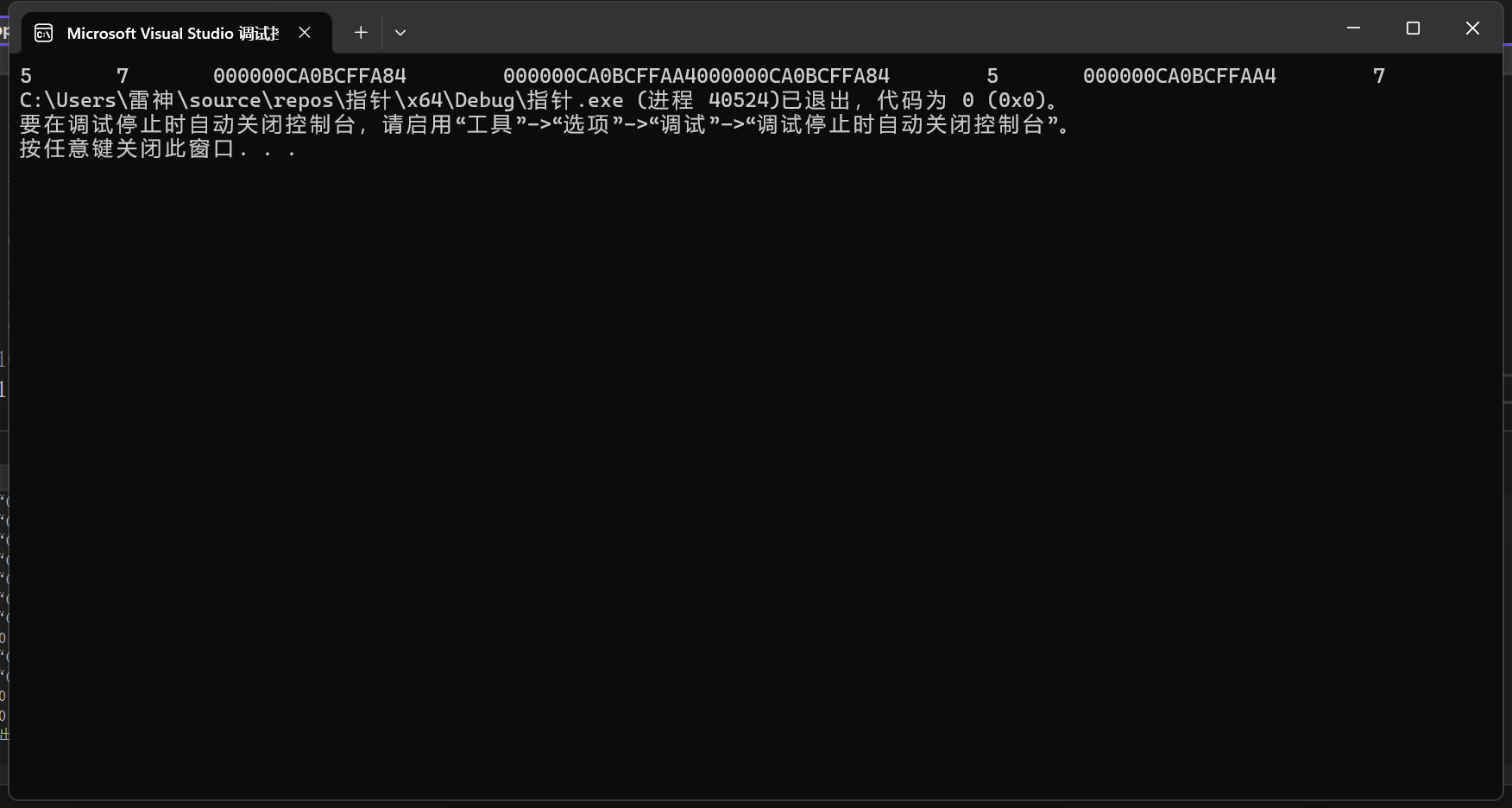
i = 5; j = 7;

cout << i <<'\t' << j << '\t' << pi << '\t' << pj;

cout << &i << '\t' << \*&i << '\t' << &j << '\t'<< \*&j;

}

**运行结果：**



**3.**

**补充后的完整代码：**#include <iostream>

#include <cstring> // 用于处理字符串相关函数，这里虽然示例代码中未明显用到，但实际更完善的处理字符串可能会用到

// 修正函数定义，添加正确的头文件引用和命名空间使用

void f(char\* st, int i)

{

st[i] = '\0';

std::cout << st << std::endl;

if (i > 1) f(st, i - 1);

}

int main()

{

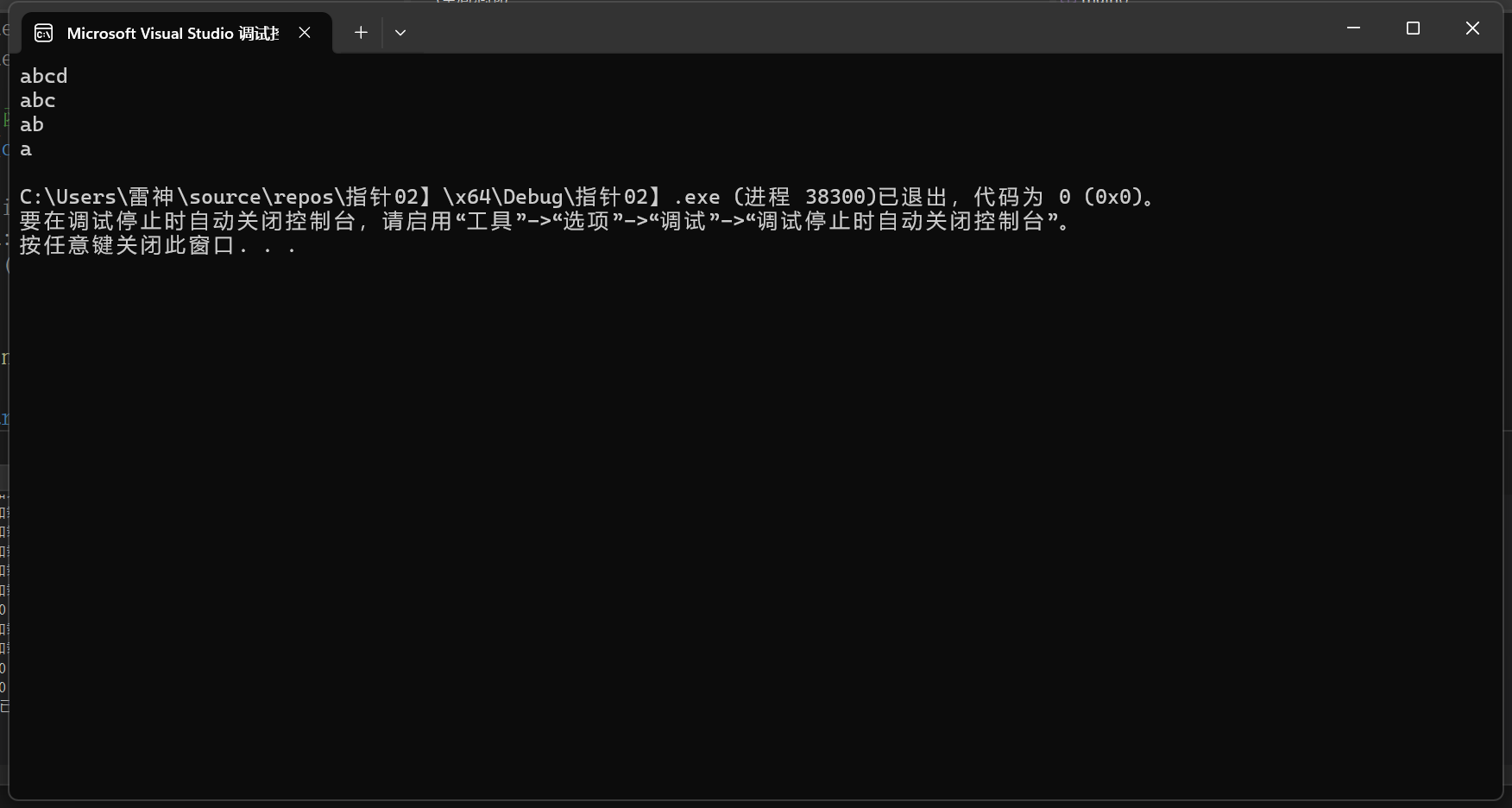
char st[] = "abcd";

f(st, 4);

return 0;

}

**运行结果：**

****

1. **答案：不能够正常输出**

### **存在的问题**

在原程序中，不能保证 p[0] 输出 1，p[1] 输出 2。原因在于函数 f 中返回了一个局部数组 list 的地址。

当函数 f 执行完毕后，其内部定义的局部变量（包括数组 list）所占用的内存空间会被自动释放，此时返回的指针就变成了指向已经被释放的内存区域的指针，也就是所谓的 “悬空指针”。当在 main 函数中通过这个悬空指针去访问数组元素时，会导致未定义行为，输出的结果是不确定的，可能会输出一些随机的值，甚至可能导致程序崩溃

更改后的代码如下：

#include<iostream>

using namespace std;

int\* f()

{

// 使用new动态分配数组内存

int\* list = new int[4] {1, 2, 3, 4};

return list;

}

int main()

{

int\* p = f();

cout << p[0] << endl;

cout << p[1] << endl;

// 使用完动态分配的内存后，要记得释放

delete[] p;

return 0;

}

**（2）**

1. **代码如下：**

#include <iostream>

#include <string>

using namespace std;

// 函数声明

int indexof(const string& s1, const string& s2) {

int len1 = s1.length(); // 获取 s1 的长度

int len2 = s2.length(); // 获取 s2 的长度

if (len1 > len2) {

return -1; // 如果 s1 长度大于 s2，必定不是子

}

}

int main()

{

string s1, s2; // 定义两个字符串

cout << "请输入字符串s1: ";

cin >> s1;

cout << "请输入字符串s2: ";

cin >> s2;

int index = indexof(s1, s2);

if (index != -1) {

cout << "字符串 \"" << s1 << "\" 是字符串 \"" << s2 << "\" 的子串，第一次出现的位置是: " << index << endl;

}

else {

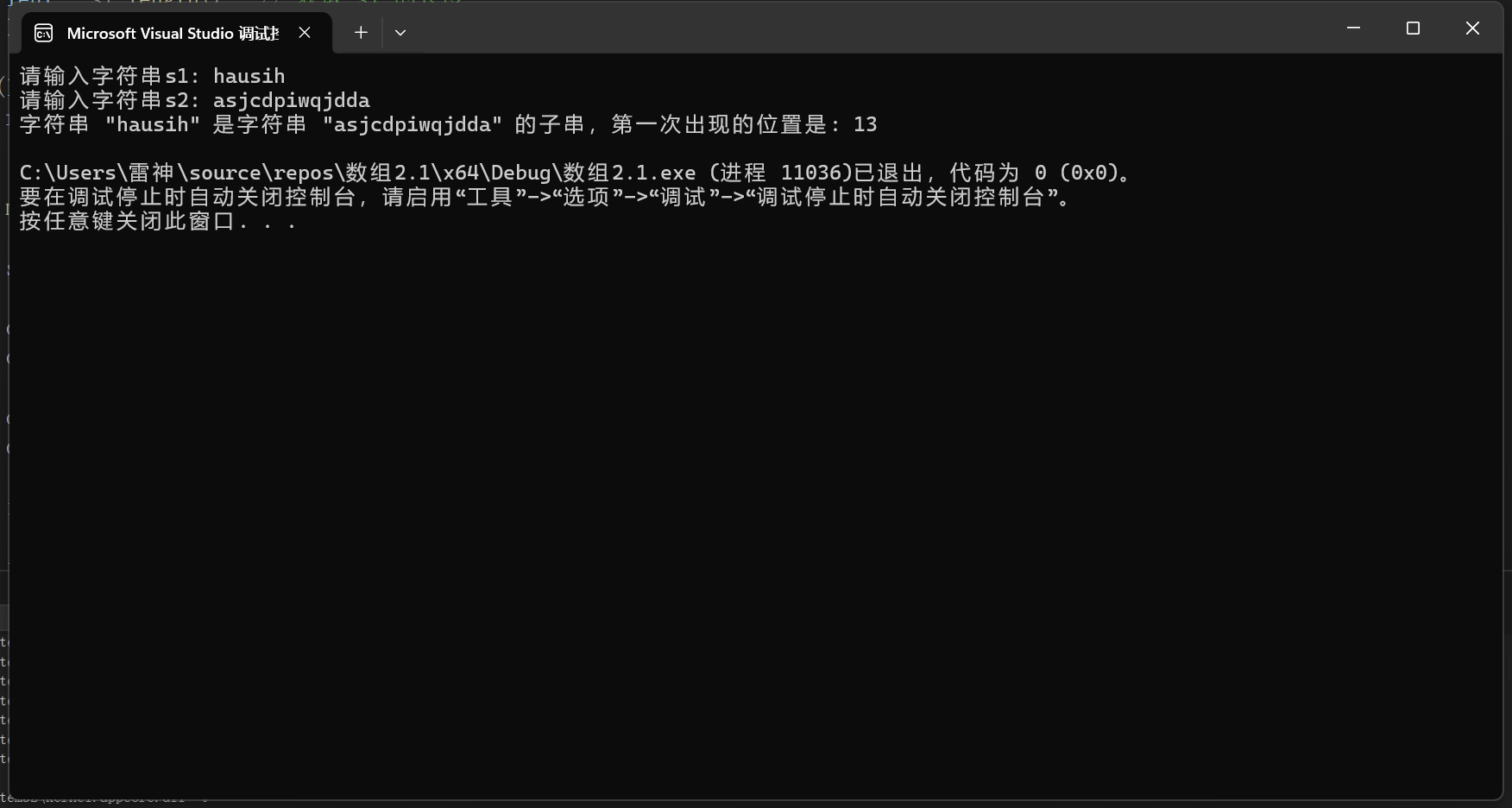
cout << "字符串 \"" << s1 << "\" 不是字符串 \"" << s2 << "\" 的子串。" << endl;

}

return 0;

}

运行结果：

1. ****代码如下：

#include <iostream>

#include <cstring> // 使用 strlen 函数

using namespace std;

int parseHex(const char\* const hexString) {

int length = strlen(hexString); // 获取十六进制字符串的长度

int result = 0; // 用于存储转换后的十进制数

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

char currentChar = hexString[i];

int currentDigit;

// 判断字符是0-9

if (currentChar >= '0' && currentChar <= '9') {

currentDigit = currentChar - '0'; // 字符0-9转换为整数0-9

}

// 判断字符是A-F

else if (currentChar >= 'A' && currentChar <= 'F') {

currentDigit = currentChar - 'A' + 10; // 字符A-F转换为整数10-15

}

// 判断字符是a-f

else if (currentChar >= 'a' && currentChar <= 'f') {

currentDigit = currentChar - 'a' + 10; // 字符a-f转换为整数10-15

}

// 如果是非法字符

else {

return -1; // 如果包含非合法十六进制字符，则返回错误值 -1

}

result = result \* 16 + currentDigit; // 按十六进制规则累加

}

return result;

}

int main() {

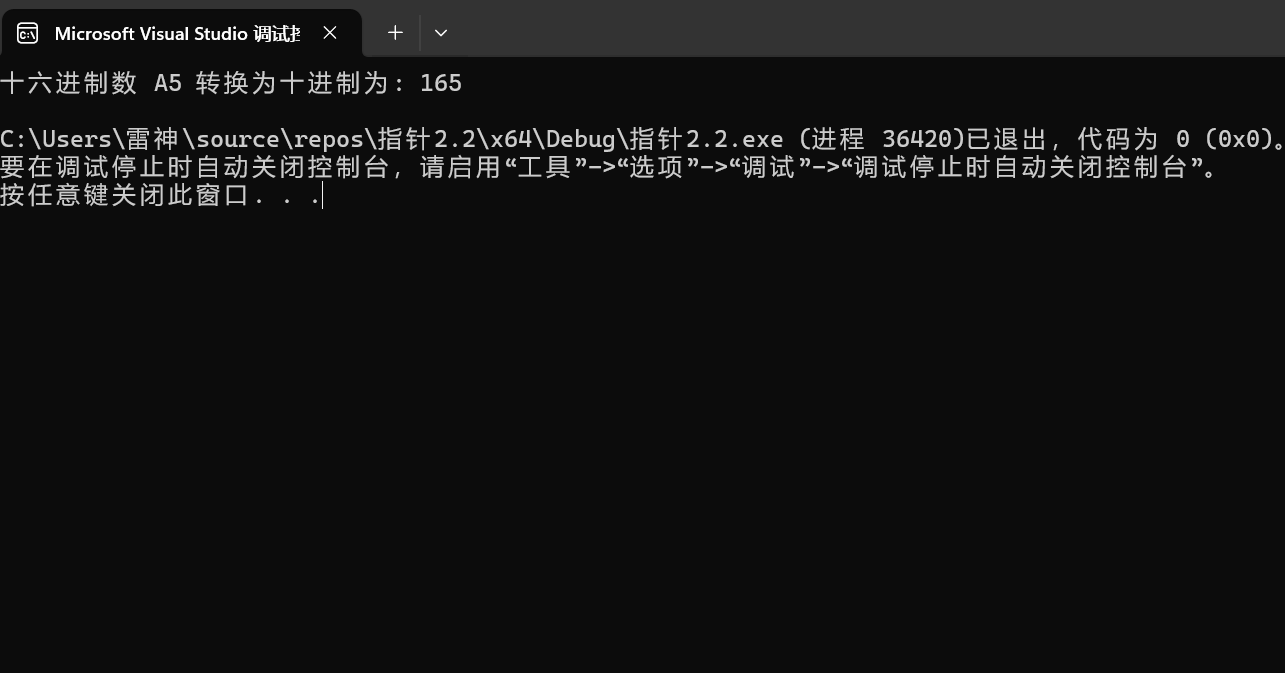
const char\* hex1 = "A5"; // 示例1

cout << "十六进制数 " << hex1 << " 转换为十进制为: " << parseHex(hex1) << endl;

return 0;

}

运行结果：



1. 代码如下：

#include <iostream>

using namespace std;

// 函数声明：对数组进行升序排序

void sortArray(int\* arr, int size);

int main() {

int size;

// 动态输入数组大小

cout << "请输入数组元素的个数: ";

cin >> size;

// 动态分配数组

int\* arr = new int[size];

// 输入数组元素

cout << "请输入 " << size << " 个数组元素:" << endl;

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

cin >> arr[i];

}

// 调试：查看指针和指针指向内容

cout << "\n调试信息：" << endl;

cout << "数组首地址: " << arr << endl;

cout << "数组内容: ";

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

cout << \*(arr + i) << " ";

}

cout << endl;

// 对数组进行排序

sortArray(arr, size);

// 使用指针方式输出排序后的数组元素

cout << "\n排序后的数组元素: ";

for (int\* p = arr; p < arr + size; p++) {

cout << \*p << " ";

}

cout << endl;

// 释放动态分配的内存

delete[] arr;

return 0;

}

// 对数组进行由小到大的排序

void sortArray(int\* arr, int size) {

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

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

if (\*(arr + j) > \*(arr + j + 1)) {

// 交换两个元素

int temp = \*(arr + j);

\*(arr + j) = \*(arr + j + 1);

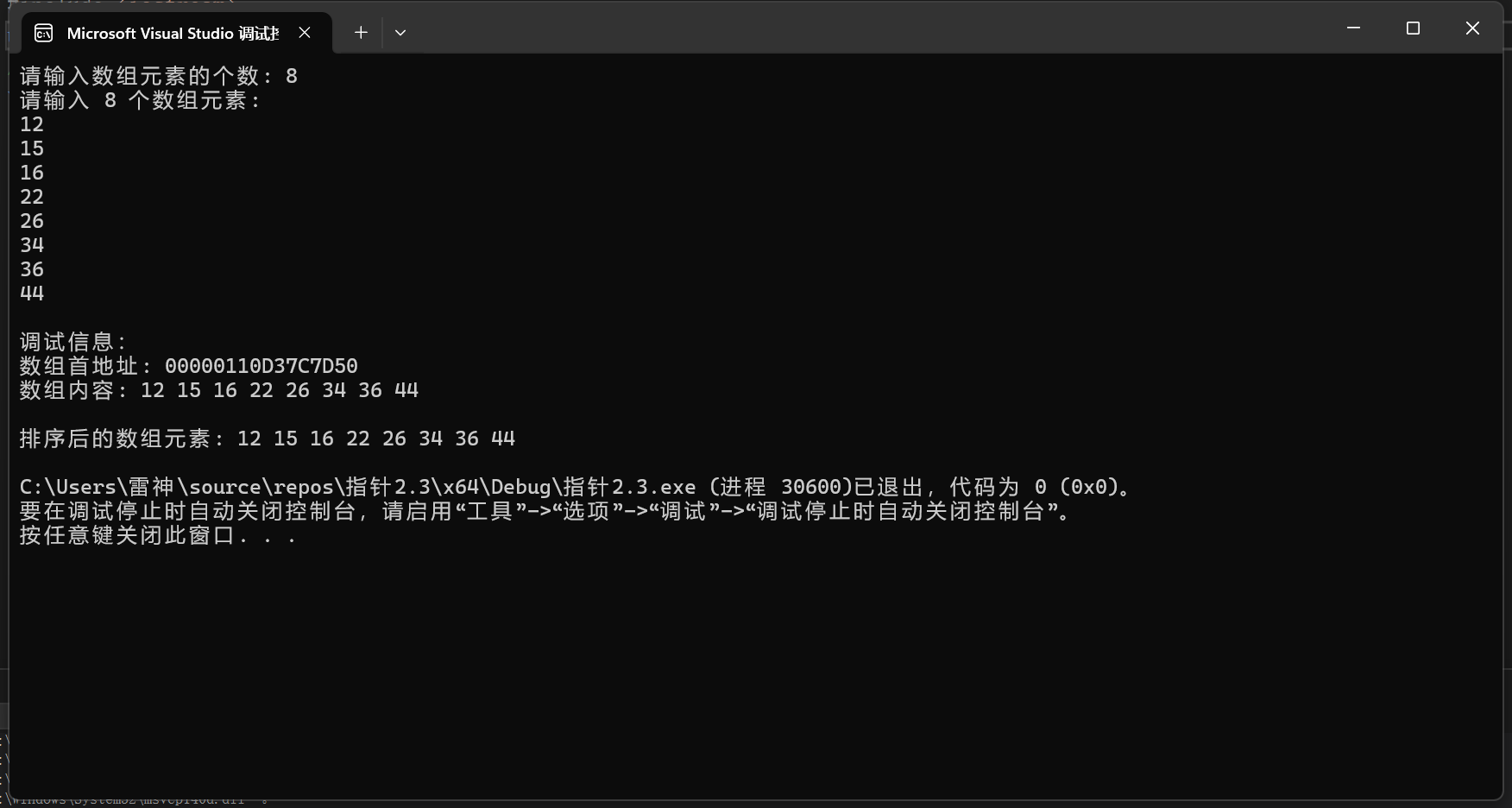
\*(arr + j + 1) = temp;

}

}

}}

运行结果：



1. **遇到的问题与解决方法**
2. **在完成数组的第一题时，对于如何对输入的数组当中重复的元素进行去重操作的时候没有思路，通过查询资料发现：**

**①需要使用到set的容器：**

**能够通过这种容器帮助我们做到快速去重**

**Set的定义：**

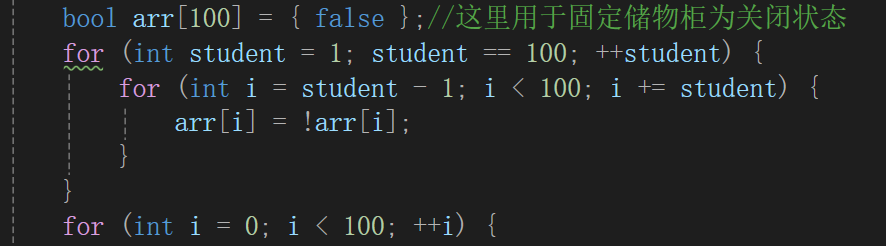
**set是STL中一种标准关联容器。它底层使用平衡的搜索树——红黑树实现，插入删除操作时仅仅需要指针操作节点即可完成，不涉及到内存移动和拷贝，所以效率比较高。set，顾名思义是“集合”的意思，在set中元素都是唯一的，而且默认情况下会对元素自动进行升序排列，支持集合的交(set\_intersection),差(set\_difference) 并(set\_union)，对称差(set\_symmetric\_difference) 等一些集合上的操作，如果需要集合中的元素允许重复那么可以使用multiset。（from csdn）**

**②如何进行去重：**

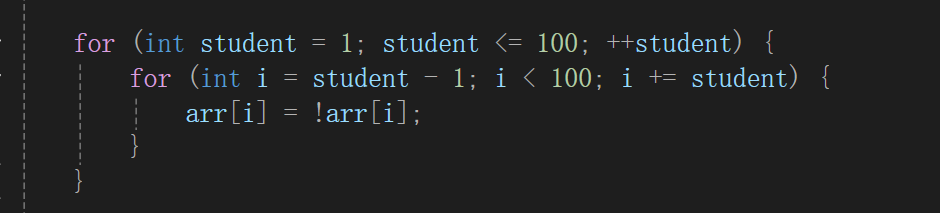
### ****方法：遍历判断法****

将每个输入值与已存储的不同值进行比较，如果该值不存在于已存储数组中，则将其加入数组，否则丢弃。

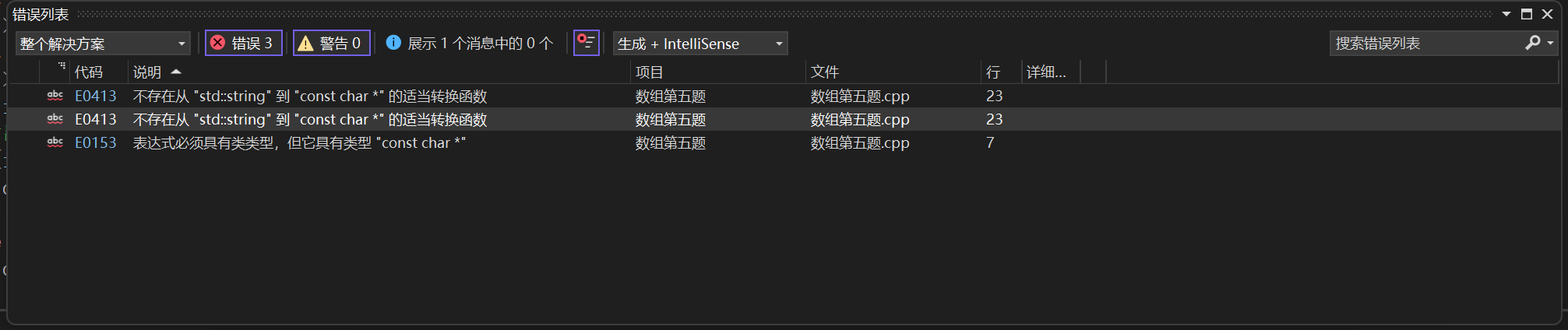
通过一个循环遍历已有的数组，判断新输入的值是否已经存在。

1. 在解决第三题时，对应的数据并没有正常输出：

这里是因为循环定义错误导致没有正常进入循环而产生无法输出的结果。

修改后：

这样能够避免产生未进入循环的问题

3.

如上图所示：

在 indexOf 函数中，代码试图使用 string 类的 find 方法来查找子串，但传入的参数是 const char[] 类型，而 find 方法是 string 类的成员函数，需要传入 string 类型的对象或者 const char\* 等与它重载形式匹配的参数。这里直接传入 const char[] 会导致编译错误，因为数组类型不能直接调用 string 类的成员函数

**解决办法：**

在 indexOf 函数里，将参数修改为 const string& 类型，这样就可以正确调用 string 类的 find 方法来查找子串了，并且在返回结果时将 size\_t 类型的结果转换为 int 类型（因为 size\_t 通常是无符号类型，而函数要求返回 int）。

1. 用指针解决动态数组问题的时候，对于new运算符的创建并没有思路

这里容易产生数组访问越界的报错导致程序不能够正常运行

解决办法：严格控制索引范围，使其在 [0, size-1] 之间。

for (int i = 0; i <= size; i++) { // 注意：如果 i <= size 会越界 cout << \*(arr + i) << " "; // 越界访问 }

1. **体会**

## **①数组**

数组在 C++ 中是一种基础且重要的数据结构。它允许我们在连续的内存空间中存储相同类型的数据元素，这使得数据的组织和访问变得较为高效。

在学习数组的过程中，我深刻体会到它的便利性与局限性。从便利性来讲，数组可以通过下标快速地访问元素，比如在处理一组成绩数据时，能够直接使用 scores[5] 来获取第 6 个学生的成绩，这种直接索引的方式简单直观，在循环遍历数组执行相同操作时也非常高效，如计算所有成绩的平均值，通过一个简单的 for 循环就能轻松实现。

然而，数组也有其局限性。其大小在定义时就必须确定且固定不变，这在实际应用中可能会带来困扰。例如，当我们事先不确定需要存储的数据量时，可能会因为数组定义得过小而导致数据溢出，或者定义得过大而浪费内存空间。而且，数组作为一种静态数据结构，缺乏灵活性，不能方便地进行插入和删除操作。例如，想要在一个已有的数组中间插入一个新元素，就需要移动其后所有元素的位置，这在数据量较大时会消耗大量的时间和资源。

## **②指针**

指针是 C++ 中极为强大但也颇具挑战性的概念。指针本质上是一个变量，它存储的是另一个变量的内存地址。

指针的强大之处在于它能够直接对内存进行操作，这使得程序可以更加灵活和高效。比如在函数间传递大型数据结构时，如果直接传递数据本身，会产生大量的复制开销，但传递指向该数据结构的指针则只传递了地址，大大提高了效率。同时，指针在动态内存分配方面也有着不可或缺的作用，通过 new 和 delete 运算符，可以在程序运行时根据实际需求动态地分配和释放内存，创建动态数组等灵活的数据结构，适应不同的程序场景，如创建一个大小由用户输入决定的数组。

但是，指针的使用也容易出错。由于指针直接操作内存地址，一旦出现错误的指针赋值或访问越界，就可能导致程序崩溃或产生难以察觉的错误结果。例如，指针未初始化就进行解引用操作，或者指针指向了已经释放的内存区域，这些错误可能在程序运行一段时间后才表现出来，给调试带来极大的困难。而且，指针的概念相对抽象，理解指针与变量之间的关系、指针的多级间接引用等需要花费较多的时间和精力去深入思考和实践。

总之，数组和指针在 C++ 编程中都有着独特的地位和作用。数组适合处理简单且规模相对固定的数据集合，而指针则为程序提供了更高的灵活性和对内存的精细控制，但同时也伴随着更多的风险和挑战。在学习和使用过程中，需要深入理解它们的特性，谨慎地运用，才能编写出高效且稳定的 C++ 程序。