**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： 8209240208

姓 名： 何承祖

**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）

#include<iostream>

using namespace std;

int max\_n(int m, int n, int &d) {

for (int i = d; ; d--) {

if (m % d == 0 && n % d == 0)

break;

}

return d;

}

int min\_n(int a, int b, int c) {

return (a \* b) / c;

}

int main() {

int a, b;

cout << "请输入a,b" << endl;

cin >> a >> b;

int c = (a > b) ? b : a;

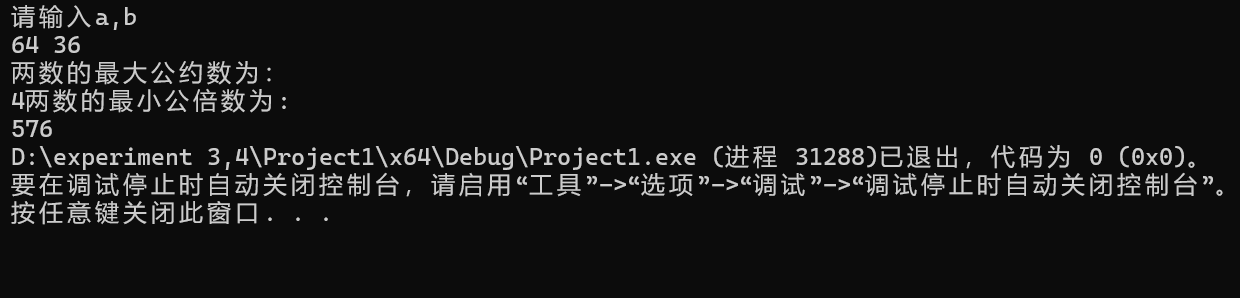
cout<<"两数的最大公约数为：" << endl;

cout << max\_n(a,b,c);

cout<<"两数的最小公倍数为:" << endl;

cout << min\_n(a,b,c );

return 0;



}

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

#include<iostream>

using namespace std;

bool is\_prime(int num) {

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

if (num % i == 0)

{

return false;

}

else {

return true;

}

}

}

int main()

{

int count = 0;

for (int i = 2; i <=200; i++) {

if (bool b = is\_prime(i)) {

cout << i << " ";

count++;

if (count % 10 == 0)

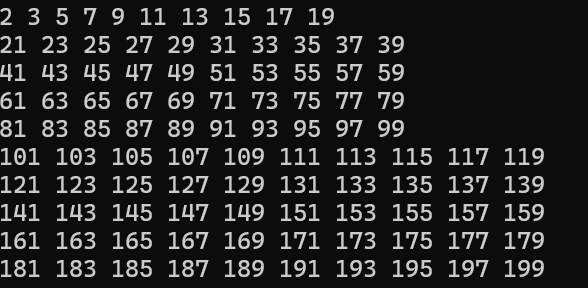
cout << endl;

}

}

return 0;

}



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）

#pragma once

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

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

#include<cmath>

#include"mytriangle.h"

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

if (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1) {

return true;

}

else return false;

}

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

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1)\*(s - side2)\*(s - side3));

}

#include<iostream>

#include"mytriangle.h"

using namespace std;

int main() {

double side1, side2, side3;

cout << "请输入三边长：" << endl;

cin >> side1 >> side2 >> side3;

bool a = is\_valid(side1,side2,side3);

if (a == true) {

cout << "三角形三边长为:" << side1 << ' ' << side2 << ' ' << side3 << endl;;

cout << "三角形的面积为" << \_area(side1, side2, side3);

}

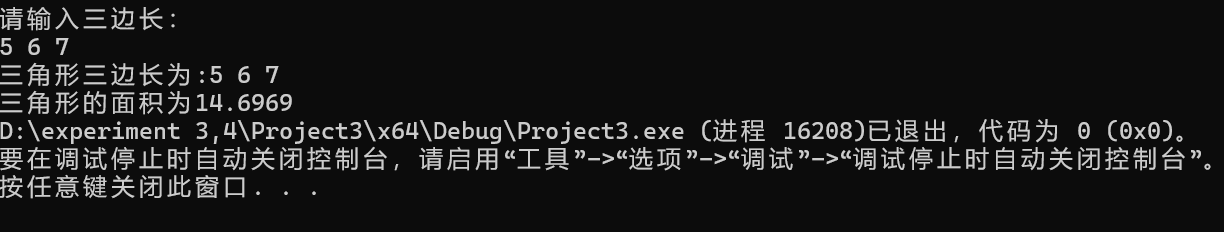
else {

cout << "不构成三角形" << endl;

}

return 0;

}



**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）。

#include<iostream>

using namespace std;

int fac(int a) {

int t;

if (a ==10) {

t = 1;

}

else {

t = (fac(a+1)+1)\*2;

}

return t;

}

int main() {

int b;

cout << "请输入天数" << endl;

cin >> b;

if (b >= 1 && b <= 10)

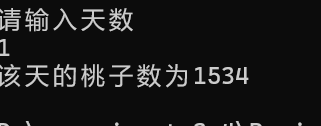
cout << "该天的桃子数为" << fac(b) << endl;

else

cout << "default";

return 0;

}



**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？

1. ****函数类型的定义和作用****
   1. 在 C++ 中，函数类型是在函数声明或定义时指定的返回值类型。例如，int add(int a, int b)这个函数声明中，int就是函数类型，表示这个函数执行完后会返回一个整数类型的值。函数类型的主要作用是让编译器知道函数调用后应该期待返回什么样的数据类型，这样编译器可以进行正确的类型检查和内存分配等操作。
2. ****返回值与函数类型一致的原因****
   1. ****编译器要求和类型安全****
      1. 编译器会根据函数类型来检查函数体中返回的值是否符合要求。如果函数类型是int，但在函数体中返回了一个double类型的值，编译器会发出警告或者错误信息。这是为了保证类型安全。例如：

收起

cpp

复制

int getValue() {

double d = 3.14;

return d; // 这里会产生警告或者错误，因为试图将double类型的值返回给int类型的函数}

* ****内存管理和数据表示的一致性****
  + 当函数返回一个值时，调用函数的地方（通常是另一个函数或者主函数）会根据函数类型来接收和处理这个返回值。如果函数类型和返回值不一致，可能会导致内存中数据的错误解释。例如，假设一个函数被定义为返回一个char类型的值，调用者会按照char类型的大小（通常是 1 字节）和格式来接收这个返回值。如果函数内部错误地返回了一个int类型的值（通常是 4 字节），就会导致数据的错误读取和使用。
* ****函数调用约定和栈操作****
  + 在函数调用过程中，编译器会按照函数类型来安排栈帧的操作。当函数返回时，会根据函数类型将返回值正确地放置在调用者可以获取的位置。如果返回值类型和函数类型不一致，栈帧中的数据传递就会出现混乱。例如，对于返回int类型值的函数，编译器可能会将返回值放在特定的寄存器（如 EAX 寄存器）或者栈上的特定位置，而这个操作是基于函数类型来确定的。如果返回的是其他类型，就无法正确地遵循这个约定，导致程序运行错误。

2. 本实验中主函数调用函数时采用的是何种传递方式？

**概念：在 C++ 中，当主函数调用一个函数，参数采用值传递方式时，会将实参的值复制一份传递给被调用函数的形参。被调用函数对形参的修改不会影响主函数中的实参。**

**原理：在函数调用时，系统会为形参分配新的内存空间，然后将实参的值复制到形参的内存空间中。所以形参和实参在内存中是不同的位置，对形参的操作不会影响实参。**

**引用传递（Pass - by - Reference）**

**概念：引用是变量的别名，当主函数以引用方式传递参数给被调用函数时，形参成为实参的别名。对形参的操作实际上就是对实参的操作。**

**指针传递（Pass - by - Pointer）**

**概念：主函数可以将变量的地址作为参数传递给被调用函数，这就是指针传递。被调用函数通过指针可以间接访问和修改主函数中的变量。**

**四、算法分析，程序结果**

**五、遇到的问题与解决方法**

**1 在cpp文件中引用.h文件时老是显示无法提取到.h文件**

**后面查阅相关资料，明白引用头文件时要用冒号。**

**2 不会控制没输出十个数字进行换行处理。后面知道可以引入一个变量，每输出一个数，变量加一，变量可以被10整除，进行换行处理。**

1. **体会**

**一、对递归思维的理解加深**

**问题分解：**

**递归要求我们能够将一个复杂的大问题逐步分解成与原问题相似但规模更小的子问题。在猴子吃桃问题中，我们把计算第一天摘的桃子数这个大目标，转化为根据后一天剩下的桃子数来推算前一天桃子数的一系列子问题。例如，要知道第一天的桃子数，就得先根据第二天的情况来推，而要知道第二天的又得基于第三天的，以此类推，直到已知的第十天剩下 1 个桃子这个基础情况。**

**这种分解方式让我们明白，面对一些具有重复性规律且可以从后往前或从前往后逐步推导的问题时，递归是一种很有效的思考方向。**

**递归调用机制：**

**通过编写代码实现递归函数，能更清晰地感受到递归调用的过程。函数会不断地调用自身，每次调用时传入的参数会逐渐逼近终止条件。在猴子吃桃问题的代码中，每次递归调用 monkeyPeach(n + 1) 就是在深入下一层去获取后续某一天的桃子信息，然后再根据这个信息返回来计算当前天的桃子数。**

**理解递归调用时的栈机制也很重要，每一次递归调用都会在栈上开辟新的空间来保存当前函数调用的状态（包括局部变量、参数等），当达到终止条件后再依次出栈返回结果，这个过程虽然在代码表面看起来简洁，但背后的执行逻辑需要深入理解才能更好地把握递归的本质。**

**二、对问题规律的把握**

**发现规律并数学建模：**

**解决这个问题首先要从题目描述中准确提炼出桃子数量变化的规律。即每天都是先吃掉剩下桃子的一半然后再多吃一个，那么反过来，要计算前一天的桃子数，就需要对后一天剩下的桃子数进行先加 1 再乘以 2 的操作。**

**将这种实际问题中的规律用数学表达式准确地表示出来，并转化为代码中的计算逻辑，是解决这类问题的关键步骤。这让我们意识到在面对实际编程场景中的问题时，能够将其抽象成数学模型或者找到明确的计算规则是多么重要，只有这样才能写出正确有效的代码。**

**利用规律进行逆向推导：**

**猴子吃桃问题是一个典型的逆向推导问题，我们已知的是最后一天的情况（第十天剩 1 个桃子），然后要通过规律逐步往前推导出第一天的桃子数。这种逆向思维在很多编程问题中都有应用，它要求我们能够根据最终的结果状态和中间的变化规律，反推出初始的状态或者条件。**

**三、代码实现层面的收获**

**简洁性与可读性：**

**递归代码在形式上往往比较简洁，相比于用循环等其他方式来实现同样的功能（比如用循环从第一天开始正向模拟猴子每天吃桃的过程来计算第一天摘的桃子数），递归代码能够用较少的代码行数就表达出复杂的问题求解过程。在猴子吃桃问题中，递归函数的主体部分就几句代码就完成了整个计算逻辑。**

**然而，递归代码的可读性相对来说可能有一定挑战，尤其是对于不熟悉递归思维的人。因为它的执行过程是隐式的，需要在脑海中去模拟函数不断调用自身的过程才能完全理解代码的意图，所以在编写递归代码时，适当添加注释来解释递归的目的、终止条件以及每次递归调用所做的事情是非常有必要的，以提高代码的可读性和可维护性。**

**终止条件的重要性：**

**在递归函数中，终止条件是至关重要的。如果没有正确设置终止条件，递归函数就会无限地调用自身，导致程序出现栈溢出等错误。在猴子吃桃问题中，当 n == 10 时返回 1 就是明确的终止条件，它界定了递归的边界，使得函数能够在合适的时候停止递归调用并开始返回结果，逐步计算出最终我们想要的第一天摘的桃子数。**

**确定终止条件需要对问题有深入的理解，要找到那个能够让我们从递归的循环中 “跳出来” 的关键情况，这也是编写递归代码时需要重点关注和仔细思考的部分。**

**通过解决猴子吃桃问题，我们在递归思维、问题规律把握以及代码实现等方面都能获得不少的收获，这些收获对于今后解决更复杂的编程问题都有着重要的启示和帮助**

**一、模块化编程的优势**

**代码结构清晰：**

**通过将函数声明放在头文件 mytriangle.h，函数定义放在单独的 mytriangle.cpp 文件，以及主测试程序放在 main.cpp 文件，实现了清晰的模块化划分。这样一来，每个文件各司其职，头文件明确给出了对外提供的功能接口（函数声明），函数定义文件具体实现这些功能，而主程序则负责调用这些功能进行实际的测试操作。这种结构使得整个代码的组织一目了然，易于理解和维护，即使在大型项目中，也能方便地定位和修改各个功能模块。**

**例如，当需要对三角形面积计算的逻辑进行修改时，我们可以直接在 mytriangle.cpp 文件中找到 area 函数进行调整，而不用担心会影响到其他部分的代码，因为其与主程序和其他功能的交互是通过明确的函数接口进行的。**

**代码复用性增强：**

**把头文件和函数定义文件独立出来后，这些代码模块可以很容易地在其他项目或程序中被复用。如果在另一个程序中也需要判断三角形是否有效以及计算其面积的功能，只需将 mytriangle.h 和 mytriangle.cpp 文件引入到新的项目中，并按照正确的方式进行编译链接即可使用。这大大节省了开发时间，避免了重复编写相同功能代码的繁琐过程，提高了代码的整体效率。**

**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：

#include<iostream>

using namespace std;

int main() {

int num[10];

int distinct[10];

int count = 0;

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

cin >> num[i];

int found = 0;

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

if (num[i] == distinct[j]) {

found = 1;

break;

}

}

if (!found) {

distinct[count++] = num[i];

}

}

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

cout << distinct[i] << " ";

}

return 0;

}



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

The distinct numbers are: 1 2 3 6 4 5

Enter

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。

#include<iostream>

using namespace std;

void swap(double& a, double& b)

{

double temp = a;

a = b;

b = temp;

}

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() {

double list[10];

cout << "please input 10 numbers";

for (int i = 0; i < 10; i++) { cin >> list[i];

}

bubblesort(list, 10);

cout << "after bubblesort" << endl;

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

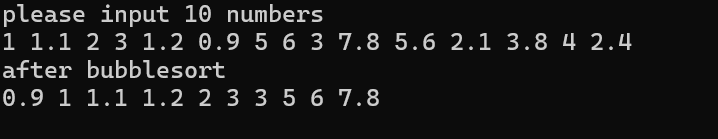
cout << list[i] << " ";

}

cout << endl;

return 0;

}



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。

#include<iostream>

using namespace std;

int main() {

bool lockers[100] = { false };

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

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

lockers[i] = !lockers[i];

}

}

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

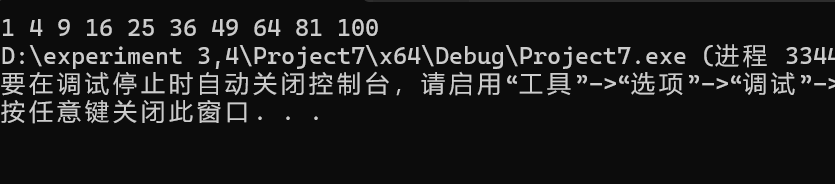
if (lockers[i]) {

cout << i + 1 << " ";

}

}

}



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

#include<iostream>

using namespace std;

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

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

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

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

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

}

else {

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

}

}

while (i < size1) { list3[k++] = list1[i++];

}

while (j < size2) {

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

}

}

int main() {

int list1[80], list2[80], list3[160];

int size1, size2;

cout << "enter list1";

cin >> size1;

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

cin >> list1[i];

}

cout << "enter list2";

cin >> size2;

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

cin >> list2[i];

}

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

cout << "the merged list is";

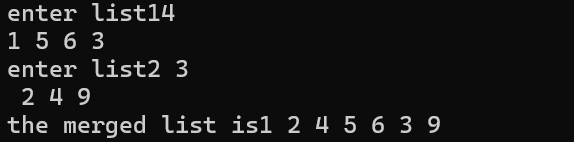
for (int i = 0; i < size1 + size2; i++) {

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

}

return 0;

}



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!

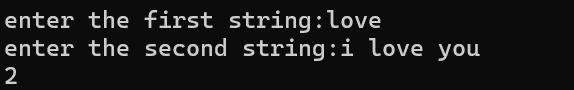
indexOf(“welcome”, “We welcome you!”) is 3

Enter

Enter the first string: welcome

Enter the second string: We invite you!

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



#include<iostream>

#include<string>

using namespace std;

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

int len1 = s1.size();

int len2 = s2.size();

for (int i = 0; i <= len2 - len1; i++)

{

int j;

for (j = 0; j < len1; j++) {

if (s2[i + j] != s1[j])

{

break;

}

}

if (j==len1) {

return i;

}

}

return -1;

}

int main() {

string s1,s2;

cout << "enter the first string:";

getline(cin, s1);

cout << "enter the second string:";

getline(cin,s2);

int result = indexof(s1, s2);

cout << result;

return 0;

}

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

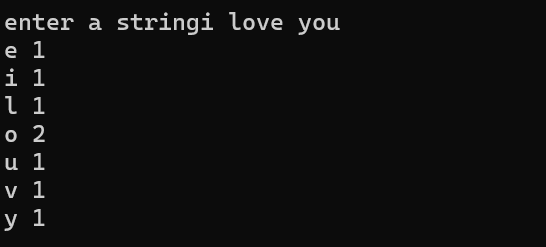
o: 3 times

r: 1 times

t: 1 times

w: 2 times

y: 1 times



#include<iostream>

#include<cctype>

#include<string>

using namespace std;

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

for (char c : s) {

if (isalpha(c)) {

char lowerchar = tolower(c);

counts[lowerchar - 'a']++;

}

}

}

int main()

{

string str;

int counts[26] = {0};

cout << "enter a string";

getline(cin, str);

count(str, counts);

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

if (counts[j]!= 0)

cout << static\_cast<char>('a' + j) <<' '<< counts[j] << endl;

}

return 0;

}

**（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;

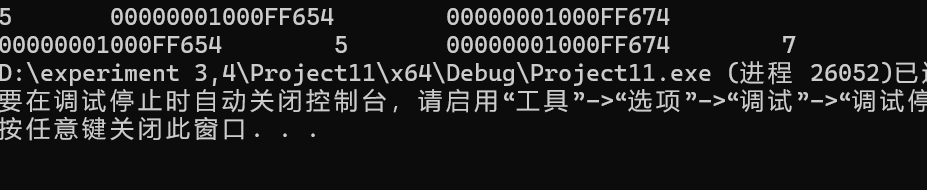
i=5;j=7;

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

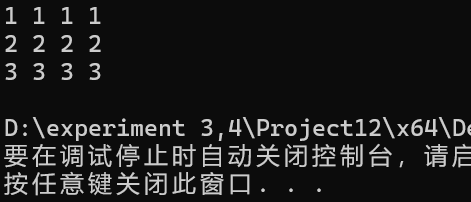
3,3,3,3

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()

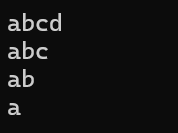
{

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;

}

#include<iostream>

using namespace std;

int\* f()

{

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

return list;

}

void main()

{

int\* p = f();

cout << p[0] << endl;

cout << p[1] << endl;

delete[]p;

}

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);

#include<iostream>

#include<string>

using namespace std;

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

int len1 = s1.size();

int len2 = s2.size();

for (int i = 0; i <= len2 - len1; i++)

{

int j;

for (j = 0; j < len1; j++) {

if (s2[i + j] != s1[j])

{

break;

}

}

if (j == len1) {

return i;

}

}

return -1;

}

int main() {

string s1, s2;

cout << "enter the first string:";

getline(cin, s1);

cout << "enter the second string:";

getline(cin, s2);

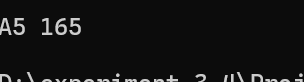
int result = indexof(s1, s2);

cout << result;

return 0;

(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



#include<iostream>

#include<string>

using namespace std;

int parseHex(const char\* const hexString) {

int len = strlen(hexString);

int decimalnumber = 0;

int power = 1;

for (int i = len - 1; i >= 0; i--) {

int digit;

if (hexString[i] >= '0' && hexString[i] <= '9') {

digit = hexString[i] - '0';

}

else if(hexString[i] >= 'A'&&hexString[i]<= 'F'){

digit = hexString[i] - 'A' + 10;

}

else if (hexString[i] >= 'a' && hexString[i] <= 'f') {

digit = hexString[i] - 'a' + 10;

}

else {

cout << "default";

}

decimalnumber += digit \* power;

power \*= 16;

}

return decimalnumber;

}

int main() {

const char\*hex="A5";

int decimal = parseHex(hex);

cout << hex << " " << decimal << endl;

return 0;

}

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）。

#include<iostream>

using namespace std;

void bubblesort(int\* arr, int n) {

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

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

if (arr[i] < arr[j]) {

int temp;

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

int main() {

int n;

cout << "input the numbers of arr:";

cin >> n;

int\* arr = new int[n];

cout << "please input n numbers:" << endl;

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

cin >> arr[i];

}

bubblesort(arr, n);

cout << "after what you do" << endl;

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

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

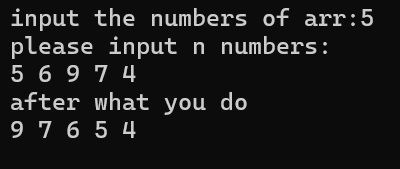
}

cout << endl;

delete[]arr;

return 0;

}

****

【完成实验报告】

**实验报告只要求写程序设计部分**

**三、算法分析，程序结果**

**四、遇到的问题与解决方法**

**1 不知道如何从数组元素中提取出不同的数**

**可以通过引入bool变量来进行同与不同的判断。利用循环逐一比较。**

**2 在合并数组时忘记思考多余元素的处理**

**可以通过循环将多余元素储存。**

**3使用cin不能读取空格，后面明白可以用cin.getline()读取空格。**

**4，需要将字符串中的字母不分大小写地进行统计，这就涉及到对字符大小写的统一处理。通过使用 tolower 函数，能够很方便地将输入字符串中的每个字符都转换为小写字母，这样就可以将大写字母和小写字母视为相同的字母进行统计。**

**五、体会**

去重逻辑的设计：

本题的核心目标是实现对输入数字的去重，为此采用了一种较为直观的方法。通过一个数组来存储已经确定为不同的数，然后对于每一个新读入的数，在已存储的数的数组中进行遍历查找。这种思路简单直接，将去重问题转化为了逐个比较和判断的过程，易于理解和实现。

例如，当读入一个新数时，通过内层循环遍历已有的不同数数组，只要找到与之相同的数，就说明它不是新的不同数，从而可以丢弃；只有当遍历完整个已存储数组都未找到相同数时，才确定它是新的不同数并进行存储。这让我深刻体会到，面对一个具体的编程问题，将其分解为一系列可操作的步骤和逻辑判断是解决问题的关键。

循环结构的运用：

程序中大量运用了循环结构，外层循环用于控制读入 10 个数的过程，确保能够完整地接收所有输入数据。而内层循环则用于在已存储不同数的数组中进行查找比较，以判断新读入的数是否为新的不同数。

这种内外层循环的嵌套配合是实现去重逻辑的重要手段。通过合理设置循环条件和循环体中的操作，能够有条不紊地完成对每个输入数的处理。同时，也让我更加熟悉了循环结构在处理重复性任务时的强大作用，以及如何根据具体问题来灵活运用不同类型的循环（本题中使用的是 for 循环）来实现所需的功能。

合并算法思路：

本题采用的合并算法思路清晰且直观。通过同时遍历两个已排序的数组，比较它们当前位置的元素大小，将较小的元素依次放入新的数组中。这个过程就像是在整理两堆已经按照某种顺序排好的物品，每次挑选出较小（或较大，取决于排序规则）的那个放入新的一堆，从而实现新数组的有序排列。

例如，在 merge 函数中，通过 while 循环不断比较 list1[i] 和 list2[j] 的大小，根据比较结果将较小的元素放入 list3 数组中，这让我深刻体会到，对于这种合并有序数据的问题，利用已有的排序特性，采用逐个比较并放置的方法是一种很有效的策略。

指针的运用：

在 merge 函数中，巧妙地运用了三个指针 i、j 和 k 来分别跟踪两个输入数组 list1、list2 的遍历进度以及新数组 list3 的填充位置。这种指针的使用方式使得对数组元素的访问和操作更加灵活和高效。

比如，通过移动指针 i、j 来依次比较两个输入数组中的元素，而指针 k 则随着每次放入新元素而递增，准确地指示了新数组 list3 中下一个可填充的位置。这让我更加熟悉了指针在处理数组相关问题时的重要作用，以及如何通过合理设置和移动指针来实现特定的算法逻辑。

计数数组的作用：

counts 数组在整个程序中起到了关键的作用，它用于存储每个字母出现的次数。通过定义一个有 26 个元素的整数数组，正好可以对应 26 个英文字母（从 a 到 z），每个元素的索引就代表了相应字母在字母表中的位置，而元素的值则表示该字母出现的次数。

例如，counts[3] 就用于记录字母 d 出现的次数。这种利用数组来进行计数统计的方式简洁直观，便于后续对统计结果的处理和展示。

数组初始化的重要性：

在 count 函数开始时，首先通过一个循环将 counts 数组的所有元素初始化为 0。这一步骤非常重要，因为如果不进行初始化，数组中的元素值将是不确定的，可能会导致统计结果出现错误。

例如，若没有初始化，可能会出现原本没有出现过的字母在 counts 数组中的对应位置上有一个随机的非零值，这样在后续显示统计结果时就会产生误导。这让我认识到，在使用数组进行计数或其他类似用途时，一定要先对数组进行初始化，确保数据的准确性和可靠性。