C++基础

Peking University
北京大学
ShuwenHe
何书文

2023 年 6 月 19 日

目录

Chapter

环境搭建

1 linux 命令

1.1 linux 文件文件夹命令

创建文件夹
mkdir cpp 创建文件夹
mkdir cpp
进入文件夹
cd cpp
vi richard.cpp
i 进入 insert 模式
o 进入下一行
int 整型
main() 主函数 function
// 单行注释
“ ” 输入字符串
endl
; 一行语句结束需要用分号结束
return 0 一个函数正确执行完成之后需要用 return 0 来结束
esc 推出 vi 编辑器
:wq 保存并推出 write quit
ls 查看当前文件夹有什么文件 ls - list directory contents

2 文本编辑器

3 C++编译器

命令行使用下面的命令来检查您的系统上是否安装了 gcc
g++ -v g++ 编译器 C++
使用 -o 选项指定可执行程序的文件名
g++ hello.cpp -o hello
./richard 执行编译器编译产生的二进制文件
指定使用 C++14 来编译
g++ -std=c++14 cpp.cpp -o cpp

3.1 g++常用命令选项

-o
file 生成指定的输出文件, 用在生成可执行文件时。
C++ 中的分号&语句块
编辑编译执行 C++程序

Chapter

C++基本语法

1

```
#include <iostream>
using namespace std;
int main() {
        cout<<"hello"<<endl;
        return 0;
}
```

C++头文件<iostream>
using namespace std; 告诉编译器使用 std 命名空间。
int main() 是主函数，程序从这里开始执行。
cout«"Hello World"; 会在屏幕上显示消息 "Hello World"。
下一行 return 0; 终止 main( ) 函数，并向调用进程返回值 0。
在 C++ 中，分号是语句结束符。也就是说，每个语句必须以分号结束。它表明一个逻辑实体的结束。

2 C++ 标识符

3 C++ 关键字

Chapter

C++ 注释

1

C++ 支持单行注释和多行注释。注释中的所有字符会被 C++ 编译器忽略。
// - 一般用于单行注释。
/* ... */ - 一般用于多行注释。

Chapter

C++ 数据类型

使用编程语言进行编程时，需要用到各种变量来存储各种信息。变量保留的是它所存储的值的内存位置。这意味着，当您创建一个变量时，就会在内存中保留一些空间。
您可能需要存储各种数据类型（比如字符型、宽字符型、整型、浮点型、双浮点型、布尔型等）的信息，操作系统会根据变量的数据类型，来分配内存和决定在保留内存中存储什么。

1 基本的内置类型

表 .4.1: 几种基本的 C++ 数据类型

类型	关键字
布尔型	bool
字符型	char
整型	int
浮点型	float
双浮点型	double
无类型	void
宽字符型	wchar_t

一些基本类型可以使用一个或多个类型修饰符进行修饰：
signed
unsigned
short
long
下面实例会输出电脑上各种数据类型的大小。

```
\begin{minted}[xleftmargin=2em]{c++}
    #include<iostream>
```

```
#include <limits>

using namespace std;

int main()  {
    cout << "type:␣\t\t" << "***********size*************"<< endl;
    cout << "bool:␣\t\t" << "所占字节数：" << sizeof(bool);
    cout << "\t最大值：" << (numeric_limits<bool>::max)();
    cout << "\t\t最小值：" << (numeric_limits<bool>::min)() << endl;
    cout << "char:␣\t\t" << "所占字节数：" << sizeof(char);
    cout << "\t最大值：" << (numeric_limits<char>::max)();
    cout << "\t\t最小值：" << (numeric_limits<char>::min)() << endl;
    cout << "signed␣char:␣\t" << "所占字节数：" << sizeof(signed char);
    cout << "\t最大值：" << (numeric_limits<signed char>::max)();
    cout << "\t\t最小值：" << (numeric_limits<signed char>::min)() << e
    cout << "unsigned␣char:␣\t" << "所占字节数：" << sizeof(unsigned ch
    cout << "\t最大值：" << (numeric_limits<unsigned char>::max)();
    cout << "\t\t最小值：" << (numeric_limits<unsigned char>::min)() <<
    cout << "wchar_t:␣\t" << "所占字节数：" << sizeof(wchar_t);
    cout << "\t最大值：" << (numeric_limits<wchar_t>::max)();
    cout << "\t\t最小值：" << (numeric_limits<wchar_t>::min)() << endl;
    cout << "short:␣\t\t" << "所占字节数：" << sizeof(short);
    cout << "\t最大值：" << (numeric_limits<short>::max)();
    cout << "\t\t最小值：" << (numeric_limits<short>::min)() << endl;
    cout << "int:␣\t\t" << "所占字节数：" << sizeof(int);
    cout << "\t最大值：" << (numeric_limits<int>::max)();
    cout << "\t最小值：" << (numeric_limits<int>::min)() << endl;
    cout << "unsigned:␣\t" << "所占字节数：" << sizeof(unsigned);
    cout << "\t最大值：" << (numeric_limits<unsigned>::max)();
    cout << "\t最小值：" << (numeric_limits<unsigned>::min)() << endl;
    cout << "long:␣\t\t" << "所占字节数：" << sizeof(long);
    cout << "\t最大值：" << (numeric_limits<long>::max)();
    cout << "\t最小值：" << (numeric_limits<long>::min)() << endl;
    cout << "unsigned␣long:␣\t" << "所占字节数：" << sizeof(unsigned lo
    cout << "\t最大值：" << (numeric_limits<unsigned long>::max)();
    cout << "\t最小值：" << (numeric_limits<unsigned long>::min)() << e
    cout << "double:␣\t" << "所占字节数：" << sizeof(double);
```

```
        cout << "\t最大值：" << (numeric_limits<double>::max)();
        cout << "\t最小值：" << (numeric_limits<double>::min)() << endl;
        cout << "long␣double:␣\t" << "所占字节数：" << sizeof(long double);
        cout << "\t最大值：" << (numeric_limits<long double>::max)();
        cout << "\t最小值：" << (numeric_limits<long double>::min)() << end
        cout << "float:␣\t\t" << "所占字节数：" << sizeof(float);
        cout << "\t最大值：" << (numeric_limits<float>::max)();
        cout << "\t最小值：" << (numeric_limits<float>::min)() << endl;
        cout << "size_t:␣\t" << "所占字节数：" << sizeof(size_t);
        cout << "\t最大值：" << (numeric_limits<size_t>::max)();
        cout << "\t最小值：" << (numeric_limits<size_t>::min)() << endl;
        cout << "string:␣\t" << "所占字节数：" << sizeof(string) << endl;
        // << "\t最大值：" << (numeric_limits<string>::max)() << "\t最小值:
        cout << "type:␣\t\t" << "************size**************"<< endl;
        return 0;
    }
    \end{minted}
```

Chapter

C++ 变量类型

Chapter

C++ 变量作用域

一般来说有三个地方可以定义变量：
在函数或一个代码块内部声明的变量，称为局部变量。
在函数参数的定义中声明的变量，称为形式参数。
在所有函数外部声明的变量，称为全局变量。
作用域是程序的一个区域，变量的作用域可以分为以下几种：
局部作用域：在函数内部声明的变量具有局部作用域，它们只能在函数内部访问。局部变量在函数每次被调用时被创建，在函数执行完后被销毁。
全局作用域：在所有函数和代码块之外声明的变量具有全局作用域，它们可以被程序中的任何函数访问。全局变量在程序开始时被创建，在程序结束时被销毁。
块作用域：在代码块内部声明的变量具有块作用域，它们只能在代码块内部访问。块作用域变量在代码块每次被执行时被创建，在代码块执行完后被销毁。
类作用域：在类内部声明的变量具有类作用域，它们可以被类的所有成员函数访问。类作用域变量的生命周期与类的生命周期相同。

1 局部变量

在函数或一个代码块内部声明的变量，称为局部变量。它们只能被函数内部或者代码块内部的语句使用。下面的实例使用了局部变量：

```
// 局部变量
int sum(){
        int a,b,sum; // 局部变量声明
        a = 1,b = 2; // 实际初始化
        sum = a + b;
        cout<<"sum␣=␣"<<sum<<endl;
        return 0;
```

```
}
```

Chapter

控制语句

1 for 循环

// 高斯求和公式求和 1+2+3+...+100

```
#include <iostream>
using namespace std;
int main() {
    int sum = 0;
    sum = (1+100)*100/2;
    cout << "gauss sum=" << sum << endl;
    return 0;
}
```

2

for 循环求和 1+2+3+...+100

```
int forSum(){
    int sum = 0;
    for (int i = 1; i <= 100; i++) {
        sum += i;
    }
    cout << "for sum=" << sum << endl;
    return 0;
}
```

3 2014NOIP 普及 T1 珠心算测验

【题目描述】

珠心算是一种通过在脑中模拟算盘变化来完成快速运算的一种计算技术。珠心算训练，既能够开发智力，又能够为日常生活带来很多便利，因而在很多学校得到普及。某学校的珠心算老师采用一种快速考察珠心算加法能力的测验方法。他随机生成一个正整数集合，集合中的数各不相同，然后要求学生回答：其中有多少个数，恰好等于集合中另外两个（不同的）数之和？最近老师出了一些测验题，请你帮忙求出答案。

输入格式

共两行，第一行包含一个整数 n，表示测试题中给出的正整数个数。第二行有 n 个正整数，每两个正整数之间用一个空格隔开，表示测试题中给出的正整数。输出格式一个整数，表示测验题答案。【输入输出样例】

输入

复制

4

1 2 3 4

输出 1

复制

2

【样例解释】

由

1+2=3,1+3=4，故满足测试要求的答案为 2。

注意，加数和被加数必须是集合中的两个不同的数。

数据范围

3 n 100，测验题给出的正整数大小不超过

假设集合中有 n 个数，我们可以使用双重循环枚举集合中的每一对数，判断它们的和是否也在集合中出现过。这个方法的时间复杂度为 $O(n^2)$，对于较大的 n 可能会超时。更高效的方法是，先将集合中的数按从小到大排序，然后对于每个数 x，使用双指针法在剩余的数中寻找两个数，使它们的和等于 x。具体地，我们可以将左指针指向 x 的下一个数，将右指针指向集合中最大的数，然后不断地将左指针右移或右指针左移，直到两个指针相遇为止。如果两个指针指向的数的和等于 x，则找到了一组符合条件的数对。这个方法的时间复杂度为 $O(n \log n)$（排序的时间复杂度为 $O(n \log n)$，每个数最多被判断一次，因此双指针法的时间复杂度为 $O(n)$），可以通过本题。以下是使用双指针法实现题目的 C++ 代码：

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    sort(a.begin(), a.end());
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        int left = i + 1, right = n - 1;
        while (left < right) {
            int sum = a[left] + a[right];
            if (sum == a[i]) {
                cnt++;
                left++;
                right--;
            } else if (sum < a[i]) {
                left++;
            } else {
                right--;
            }
        }
    }
    cout << cnt << endl;
    return 0;
}
```

代码中使用了 STL 的 ‘vector‘ 和 ‘sort‘，可以方便地实现数组的排序。双指针法的部分使用了 ‘while‘ 循环和条件语句进行实现。

Chapter

函数

1　CSP-J（普及组）2022 年 T1 乘方 (pow)

Chapter

数组

Chapter

STL