字符串、向量和数组

2021年10月25日 星期一 14:55

1、在执行读取操作时，string对象会自动忽略开头的空白（即空格符、换行符、制表符等），并从第一个真正的字符开始读起，直到遇见下一个空白为止。

2、getline函数一次读取一整行，参数是一个输入流和一个string对象。

getline(cin,line);

3、字符串字面值（如"hello"）和string是不同的类型。

4、一般来说，C++程序更应该使用名为cname的头文件而不使用name.h的形式（方便区分哪些是从C继承过来的，哪些是C++独有的）。

5、

```
for (declaration : expression)
    statement
```

其中，*expression* 部分是一个对象，用于表示一个序列。*declaration* 部分负责定义一个变量，该变量将被用于访问序列中的基础元素。每次迭代，*declaration* 部分的变量会被初始化为 *expression* 部分的下一个元素值。

for(auto& c : s)

{ c=toupper(c); } //c是一个引用，因此赋值语句将改变s中字符的值

6、引用不能成为vector的元素，因为其不是对象。

7、使用小括号()和花括号{}初始化vector的区别。（百度）

8、如果循环体内部包含有向vector对象添加元素的语句，则不能使用范围for循环。

范围for循环语句体内不应改变其所遍历序列的大小。

9、如果vector对象或string对象是一个常量，只能使用const_iterator

10、谨记，但凡是使用了迭代器的循环体，都不要像迭代器所属的容器添加元素。

11、可以令迭代器和一个整数值相加或相减，其返回值是向前或向后移动了若干个位置的迭代器。

12、end迭代器指向的是最后一个元素的下一位置。

13、二分查找要用mid=beg+(end-beg)/2;而不是mid=(beg+end)/2;

原因：迭代器只能相减，代表它们之间的距离。没有定义迭代器之间的加法运算。

14、

```
int *ptrs[10];              // ptrs 是含有 10 个整型指针的数组
int &refs[10] = /* ? */;    // 错误：不存在引用的数组
int (*Parray)[10] = &arr;   // Parray 指向一个含有 10 个整数的数组
int (&arrRef)[10] = arr;    // arrRef 引用一个含有 10 个整数的数组
int *(&arry)[10] = ptrs; // arry 是数组的引用，该数组含有 10 个指针
```

要想理解数组声明的含义，最好的办法是从数组的名字开始按照由内向外的顺序阅读。

15、在使用数组下标的时候，通常将其定义为size_t类型。size_t是一种机器相关的无符号整数类型，它被设计得足够大，能够表示任意数组的大小。

16、

尽管能计算得到尾后指针，但这种用法极易出错。为了让指针的使用更简单、更安全，C++11 新标准引入了两个名为 begin 和 end 的函数。这两个函数与容器中的两个同名成员（参见 3.4.1 节，第 95 页）功能类似，不过数组毕竟不是类类型，因此这两个函数不是成员函数。正确的使用形式是将数组作为它们的参数：

```
int ia[] = {0,1,2,3,4,5,6,7,8,9}; // ia 是一个含有 10 个整数的数组
int *beg = begin(ia);        // 指向 ia 首元素的指针
int *last = end(ia);         // 指向 arr 尾元素的下一位置的指针
```

begin 函数返回指向 ia 首元素的指针，end 函数返回指向 ia 尾元素下一位置的指针，这两个函数定义在 iterator 头文件中。

使用 begin 和 end 可以很容易地写出一个循环并处理数组中的元素。例如，假设 arr

difference_type 由 string 和 vector 定义的一种带符号整数类型，表示两个迭代器之间的距离。

size_type 是 string 和 vector 定义的类型的名字，能存放下任意 string 对象或 vector 对象的大小。在标准库中，size_type 被定义为无符号类型。

prtdiff_t 是 cstddef 头文件中定义的一种与机器实现有关的带符号整数类型，它的空间足够大，能够表示数组中任意两个指针之间的距离。

size_t 是 cstddef 头文件中定义的一种与机器实现有关的无符号整数类型，它的空间足够大，能够表示任意数组的大小。

17、difference_type是一种由string和vector定义的一种带符号整数类型，表示两个迭代器之间的距离。

18、指针也可以像迭代器那样进行运算，注意两个指针之间只能相减不能相加。

两个指针相减的结果的类型是一种名为ptrdiff_t的标准库类型，和size_t一样，ptrdiff_t也是一种定义在cstddef头文件中的机器相关的类型。

因为差值可能是负的，所以ptrdiff_t是一种带符号类型。

19、内置的下标运算符所用的索引值不是无符号类型，这一点与vector和string不一样。

也就是说例如数组的下标可以出现负数，如a[-2]，只要这块内存已经被定义了就可以这么用。

20、严格来说，C++语言中没有多维数组，通常所说的多维数组其实是数组的数组。

21、

```
for (const auto &row : ia)    // 对于外层数组的每一个元素
    for (auto col : row)      // 对于内层数组的每一个元素
        cout << col << endl;
```

这个循环中并没有任何写操作，可是我们还是将外层循环的控制变量声明成了引用类型，这是为了避免数组被自动转成指针（参见 3.5.3 节，第 105 页）。假设不用引用类型，则循环如下述形式：

```
for (auto row*: ia)
    for (auto col : row)
```

程序将无法通过编译。这是因为，像之前一样第一个循环遍历 ia 的所有元素，注意这些元素实际上是大小为 4 的数组。因为 row 不是引用类型，所以编译器初始化 row 时会自动将这些数组形式的元素（和其他类型的数组一样）转换成指向该数组内首元素的指针。这样得到的 row 的类型就是 int*，显然内层的循环就不合法了，编译器将试图在一个 int*内遍历，这显然和程序的初衷相去甚远。

Note 要使用范围 for 语句处理多维数组，除了最内层的循环外，其他所有循环的控制变量都应该是引用类型。

22、

练习 3.43：编写 3 个不同版本的程序，令其均能输出 ia 的元素。版本 1 使用范围 for 语句管理迭代过程；版本 2 和版本 3 都使用普通的 for 语句，其中版本 2 要求用下标运算符，版本 3 要求用指针。此外，在所有 3 个版本的程序中都要直接写出数据类型，而不能使用类型别名、auto 关键字或 decltype 关键字。

```
#include <iostream>
using namespace std;
int main()
{
    int ia[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
    cout << "利用范围 for 语句输出多维数组的内容：" << endl;
    for(int (&row)[4] : ia)
    {
        for(int &col : row)
            cout << col << " ";
        cout << endl;
    }

    cout << "利用普通 for 语句和下标运算符输出多维数组的内容：" << endl;
    for(int i = 0; i != 3; i++)
    {
        for(int j = 0; j != 4; j++)
            cout << ia[i][j] << " ";
        cout << endl;
    }

    cout << "利用普通 for 语句和指针输出多维数组的内容：" << endl;
    for(int (*p)[4] = ia; p != ia + 3; p++)
    {
        for(int *q = *p; q != *p + 4; q++)
            cout << *q << " ";
        cout << endl;
    }
    return 0;
}
```

23、

练习 3.44：改写上一个练习中的程序，使用类型别名来代替循环控制变量的类型。

```
#include <iostream>

using namespace std;
using int_array = int[4];

int main()
{
    int ia[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
    cout << "利用范围 for 语句输出多维数组的内容：" << endl;
    for(int_array &row : ia)
    {
        for(int &col : row)
            cout << col << " ";
        cout << endl;
    }

    cout << "利用普通 for 语句和下标运算符输出多维数组的内容：" << endl;
    for(int i = 0; i != 3; i++)
    {
        for(int j = 0; j != 4; j++)
            cout << ia[i][j] << " ";
        cout << endl;
    }

    cout << "利用普通 for 语句和指针输出多维数组的内容：" << endl;
    for(int_array *p = ia; p != ia + 3; p++)
    {
        for(int *q = *p; q != *p + 4; q++)
            cout << *q << " ";
        cout << endl;
    }
    return 0;
}
```

24、

练习 3.45：再一次改写程序，这次使用 auto 关键字。

```
#include <iostream>

using namespace std;

int main()
{
    int ia[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
    cout << "利用范围 for 语句输出多维数组的内容：" << endl;
    for(auto &row : ia)
    {
        for(auto &col : row)
            cout << col << " ";
        cout << endl;
    }

    cout << "利用普通 for 语句和下标运算符输出多维数组的内容：" << endl;
    for(auto i = 0; i != 3; i++)
    {
        for(auto j = 0; j != 4; j++)
            cout << ia[i][j] << " ";
        cout << endl;
    }

    cout << "利用普通 for 语句和指针输出多维数组的内容：" << endl;
    for(auto p = ia; p != ia + 3; p++)
    {
        for(auto q = *p; q != *p + 4; q++)
            cout << *q << " ";
        cout << endl;
    }
    return 0;
}
```