@阿西拜-南昌

字符串、向量和数组

C++ Primer第五版　　第3章

目录

命名空间的using声明：using namespace::name;


```cpp
#include <iostream>
int main()
{
    std::cout << "Enter two numbers:" << std::endl;

    int v1, v2;
    std::cin >> v1 >> v2;

    std::cout << "The sum of " << v1 << " and " << v2
        << " is " << v1 + v2 << std::endl;

    return 0;
}
```

```cpp
#include <iostream>

// using declarations for names from the standard library
using std::cin;
using std::cout; using std::endl;

int main()
{
    cout << "Enter two numbers:" << endl;

    int v1, v2;
    cin >> v1 >> v2;

    cout << "The sum of " << v1 << " and " << v2
        << " is " << v1 + v2 << endl;

    return 0;
}
```

头文件不应该包含using声明，因为会被拷贝到引用该头文件的文件中。

标准库类型string：长度可变的字符序列，需要包含string头文件

- **定义和初始：**

```
#include <iostream>
#include <string>
using std::string;
int main()
{
	string s1; //空字符串
	string s2 = s1; //副本，等价于 s2(s1)
	string s3 = "hiya"; //副本，等价于 s3("hiya")
	string s4(10,'c'); //ccccccccccc

	return 0;
}
```

- **string对象上的操作**

string的操作	
os<<s	将s写到输出流os当中，返回os
is>>s	从is中读取字符串赋给s，字符串以空白分隔，返回is
getline(is,s)	从is中读取一行赋给s，返回is
s.empty()	s为空返回true，否则返回false
s.size()	返回s中字符的个数，返回值为string::size_type类型
s[n]	返回s中第n个字符的引用，位置n从0记起
s1+s2	返回s1和s2连接后的结果
s1=s2	用s2的副本替代s1中原来的字符
s1==s2	判断s1和s2是否完全一样
s1!=s2	判断s1和s2是否不完全一样
<,<=,>,>=	利用字符在字典中的顺序进行比较

```
string s1, s2;

cin >> s1 >> s2; // read first input into s1, second into s2
cout << s1 << s2 << endl; // write both strings
```

C:\Windows\system32\cmd.exe

```
hello world
helloworld
```

```
int main(){
	string word;
	while(cin >> word)
		cout<<word<<endl;
	return 0;
}
```

C:\Windows\system32\cmd.exe

```
hello the world
hello
the
world
^Z
```

C:\Windows\system32\cmd.exe

```
hello
hello
the
the
world
world
^Z
```

```cpp
int main()
{
    string line;
    //每次读入一整行，包括空白，直到文件末尾
    while(getline(cin,line))
        cout<<line<<endl;

    return 0;
}
```

getline得到的string对象中并不包含换行符

```
C:\Windows\system32\cmd.exe
hello the world
hello the world
^Z
```

- **字面值和string对象相加**

```cpp
string s1 = "hello", s2 = "world";
string s3 = s1 + "," + "s2" + "\n";

string s4 = s1 + ",";
string s5 = "hello" + ","; //错误

string s6 = s1 + "," + "world";
string s7 = "hello" + "," + s2; //错误
string s8 = "hello" + ("," + s2);
```

C++语言中，字符串字面值并不是stirng对象。

C++将C语言标准库的内容，命名为cname（不含.h）

cctype头文件（ctype.h）中的函数	
isalnum(c)	如果c是字母或数字，该函数返回true
isalpha(c)	如果c是字母，该函数返回真
iscntrl(c)	如果c是控制字符，该函数返回true
isdigit(c)	如果c是数字（0～9），该函数返回true
isgraph(c)	如果c是除空格之外的打印字符，该函数返回true
islower(c)	如果c是小写字母，该函数返回true
isprint(c)	如果c是打印字符（包括空格），该函数返回true
ispunct(c)	如果c是标点符号，该函数返回true
isspace(c)	如果c是标准空白字符，如空格、进纸、换行符、回车、水平制表符或者垂直制表符，该函数返回true
isupper(c)	如果c是大写字母，该函数返回true
isxdigit(c)	如果c是十六进制的数字，即0～9、a~f、A~F，该函数返回true
tolower(c)	如果c是大写字符，则返回其小写，否则返回该参数
toupper(c)	如果c是小写字母，则返回其大写，否则返回该参数

```cpp
int main() {
	string s("Hello World!!!");
	decltype(s.size()) punct_cnt = 0;
	// 算一算有多算个标点符号
	for (auto c : s)		// for every char in s
		if (ispunct(c))		// if the character is punctuation
			++punct_cnt;	// increment the punctuation counter

	cout << punct_cnt << " punctuation characters in " << s << endl;
	// 转换为大写
	string orig = s;
	for (auto& c : s)
		c = toupper(c);
	cout << s << endl;
```

Microsoft Visual Studio 调试控制台

```
3 punctuation characters in Hello World!!!
HELLO WORLD!!!
```

```cpp
	s = orig;
	decltype(s.size()) index = 0;

	while (index != s.size() && !isspace(s[index])) {
		s[index] = toupper(s[index]);
		++index;
	}
	cout << s << endl;
	return 0;
}
```

```cpp
int main(){
	const string hexdigits = "0123456789ABCDEF";  // possible hex digits

	cout << "Enter a series of numbers between 0 and 15\n"
	<< " separated by spaces.  Hit ENTER when finished: "<< endl;
	string result;		// will hold the resulting hexify'd string

	string::size_type n;  // hold numbers from the input
	while (cin >> n)
		if (n < hexdigits.size())    // ignore invalid input
			result += hexdigits[n];  // fetch the indicated hex digit

	cout << "Your hex number is: " << result << endl;
	return 0;
}
```

C:\Windows\system32\cmd.exe

```
Enter a series of numbers between 0 and 15
 separated by spaces.  Hit ENTER when finished:
1 2 3 4 5 10 11 15
^Z
```

标准库类型vector：对象的集合

- vector是模板，由vector生成的模板必须包含vector中元素的类型
  - 例如vector<int>、vector<vector<int> >
- 引用不是对象，所以不存在包含引用的vector

```
#include <vector>
using std::vector;
```

初始化vector对象的方法	
vector<T> v1	v1是一个空vector，它潜在的元素是T类型的，指向默认初始化
vector<T> v2(v1)	v2中包含有v1所有元素的副本
vector<T> v2 = v1	等价于v2(v1)
vector<T> v3(n,val)	v3包含n个重复个元素，每个元素的值都是val
vector<T> v4(n)	v4包含了n个重复地执行了值初始化的对象
vector<T> v5{a,b,c...}	v5包含了初始值个数的元素，每个元素都被赋予相应的初始值
vector<T> v5={a,b,c...}	等价于vector<T> v5{a,b,c...}

需要使用for语句时，尽量使用范围for，以免溢出。

```
int main()
{
    vector<int> v{1,2,3,4,5,6,7,8,9};
    for(auto &i :v )
        i*=i; //求元素值的平方

    for(auto i:v)
        cout<< i << " ";

    cout << endl;
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
1 4 9 16 25 36 49 64 81
```

Vector支持的操作	
v.empty()	如果v不含有任何元素，返回阵：否则返回假
v.size()	返回v中元素的个数
v.push_back(t)	向v的尾端添加一个值为t的元素
v[n]	返回v中第n个位置上元素的引用
v1 = v2	用v2中元素的拷贝替换v1中的元素
v1 = {a,b,c...}	用列表中元素的拷贝替换v1中短时
v1 == v2	v1、v2相等当且仅当他们的元素数量相同且对应位置的元素值都相同
v1 != v2	
<,<=,>,>=	以字典顺序进行比较

```cpp
#include <string>
using std::string;
#include <vector>
using std::vector;
#include <iostream>
using std::cin; using std::cout; using std::endl;

int main(){
	vector<unsigned> grades;// hold the grades we read from the standard input

	// count the number of grades by clusters of ten:
	// 0--9, 10--19, . ..  90--99, 100
	vector<unsigned> scores(11, 0); // 11 buckets, all initially 0
	unsigned grade;
	while (cin >> grade) {     // read the grades
		if (grade <= 100)      // handle only valid grades
			grades.push_back(grade);
		++scores[grade/10]; // increment the counter for the current cluster
	}
	cout << "grades.size = " << grades.size() << endl;
	for (auto it : grades)
		cout << it << " " ;
	cout << endl;

	cout << "scores.size = " << scores.size() << endl;
	for (auto it : scores)
		cout << it << " " ;
	cout << endl;
}
```

```
Microsoft Visual Studio 调试控制台
42 65 95 100 39 67 95 76 88 76 83 92 76 93
^Z
grades.size = 14
42 65 95 100 39 67 95 76 88 76 83 92 76 93
scores.size = 11
0 0 0 1 1 0 2 3 2 4 1
```

迭代器（iterator）介绍：类似指针

- 使用迭代器可以访问某个元素，迭代器也能从一个元素移动到另一个元素。
- 有迭代器的类型都拥有begin和end成员
  - begin：返回指向第一个元素（或字符）的迭代器
  - end：尾后迭代器，即尾元素的下一个位置（一个本不存在的元素）

```
auto b = v.begin(), e = v.end(); //b和e的类型相同
```

如果容器为空，则begin和end返回的是同一个迭代器，都是尾后迭代器。

标准容器迭代器的运算符	
*iter	返回迭代器iter所指元素的引用
iter->mem	解引用iter并获取该元素的名为mem的成员，等价于(*iter).mem
++iter	令iter指示容器中的下一个元素(尾后迭代器除外)
--iter	令iter指示容器中的上一个元素
iter1 == iter2	如果两个迭代器指示的是同一个元素则相等，否则不等。
iter1 != iter2	

尾后迭代器 并不实际指示某一个元素，所以不能对其进行递增或解引用

```
string s("some string");
//将s首字母大写，变为Some string
if(s.begin()!=s.end()){
        auto it = s.begin();
        *it = toupper(*it);
}
//将s首单词大写，变为SOME string
for(auto it = s.begin();it != s.end()&&!isspace(*it); ++it)
                *it=toupper(*it);
```

迭代器类型：

- 拥有迭代器的标准类型使用iterator和const_iterator（和常量指针差不多）

```
vector<int>::iterator it; //it能读写vector<int>元素
string::iterator it2; //it2能读写string对象中的字符

vector<int>::const_iterator it3; //it3只能读元素，不能写元素
string::const_iterator it4; //it4只能读元素，不能写元素
```

- 如果对象是常量，begin和end返回const_iterator,否则返回iterator：

```
vector<int> v;
const vector<int> cv;
auto it1 = v.begin(); //it1的类型是vector<int>::iterator
auto it2 = cv.begin(); //it2的类型是vector<int>::const_iterator
```

- 有时候我们希望即使对象不是常量，我们也要使用const_iterator：
  - C++11新标准引入了cbegin和cend：

```
auto it3 = v.cbegin(); //it3的类型是vector<int>::const_iterator
```

结合解引用的成员访问：

```
vector<string> v;
auto it = v.begin();

(*it).empty();
*it.empty();  //错误：试图访问it的名为empty的成员，但it是迭代器
it->empty(); //箭头运算符：把解引用和成员访问两个操作合在一起
```

```
int main() {
	vector<string> text(3, "hello the world!!");

	for (auto it = text.cbegin(); it != text.cend() && !it->empty(); ++it)
		cout << *it << endl;
} //将输出三行hello the world!!
```

任何一种可能改变vector对象容量的操作，都会使得对于的迭代器失效

迭代器运算：

vector和string的迭代器提供了更多额外的运算符：

- **iter+n**
- **iter-n**
- **iter1 + = n**
- **iter1 – = n**
- **iter1 – iter2**
- **> 、>= 、< 、<=**

可以令迭代器和一个整数相加（或相减），其返回值是向前（或向后）移动了若干位置的迭代器。

```
int main() {
	vector<int> text = { 1,2,3,4,5 };
	auto sought = 2;
	auto beg = text.begin(), end = text.end();
	auto mid = beg + (end-beg) / 2; //初始状态的中间点

	while (mid != end && *mid != sought) {
		if (sought < *mid)//在前半部分吗？
			end = mid;
		else
			beg = mid + 1;
		mid = beg + (end - beg) / 2;//新的中间点
	}
	if (mid != text.end())
		cout << "找到了！" << *mid <<endl;
	else
		cout << "没有找到！" << endl;

	return 0;
}
```

数组：复合类型（声明形如：a[d]）

- a：数组名称
- d：元素个数（必须是常量表达式）

```
unsigned cnt = 42;
constexpr unsigned sz = 42; //常量表达式
int arr[10];
int *parr[sz];
string bad[cnt]; //错误：cnt不是常量表达式
string strs[get_size()]; //当get_size是constexpr时正确；否则错误
```

const只限制只读，并不要求在编译时就确定，可以在运行时确定。只有编译时const才可以表示数组大小

- 不存在引用数组
- 可以使用列表初始化，但必须指定数组类型，不允许使用auto

```
const unsigned sz = 3;
int ial[sz] = {0,1,2};
int  a2[]  =  {0,1,2};  //自动推断元素个数为3
int a3[5] = {0,1,2}; //等价于a3[] = {0,1,2,0,0}
string a4[3] = {"hi","bye"}; //等价于a4[]={"hi","bye",""}
int a5[2] = {0,1,2}; //错误：初始值过多
```

- 字符数组的特殊性：字符串字面值的结尾处还有一个空字符

```
char a1[] = {'C','+','+'}; //列表初始化，没有空字符
char a2[] = {'C','+','+','\0'}; //列表初始化，含有显示的空字符
char a3[] = "C++"; //含有空字符
const char a4[6] = "Danial"; //错误，没有空间存放空字符！
```

- 不允许拷贝和赋值

```
int a[] = {0,1,2};
int a2[] = a; //初始化时拷贝错误
a2 = a; //赋值错误
```

- 理解复杂的数组声明

```
int  *ptrs[10];  //ptrs是含有10个元素（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 (&Parray)[10] = arr; //Parray引用一个含有10个整数的数组
int *(&arry)[10] = ptrs; //arry是数组的引用，该数组包含10个指针
```

```
Microsoft Visual Studio 调试控制台
42 65 95 100 39 67 95 76 88 76 83 92 76 93
^Z
grades.size = 14
42 65 95 100 39 67 95 76 88 76 83 92 76 93
0 0 0 1 1 0 2 3 2 4 1
```

```
int main()
{
	vector<unsigned> grades;
	// count the number of grades by clusters of ten:
	// 0--9, 10--19, . . . 90--99, 100
	unsigned scores[11] = {};  // 11 buckets, all value initialized to 0
	unsigned grade;
	while (cin >> grade) {
		if (grade <= 100)
			++scores[grade / 10];

		grades.push_back(grade);
	}
	cout << "grades.size = " << grades.size() << endl;

	for (auto g : grades)  // for every element in grades
		cout << g << " ";
	cout << endl;

	for (auto i : scores)	// for each counter in scores
		cout << i << " ";	// print the value of that counter
	cout << endl;
}
```

指针和数组：

使用数组的时候，编译器一般会把它转换成指针

```
string nums[] = {"one","two","three"}; //数组的元素是string对象
string *p = &nums[0]; //p指向nums的第一个元素
string *p2 = nums; //等价于p2 = &nums[0]

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
auto ia2(ia); //ia2是一个整型指针，指向ia的第一个元素,等价于ia2(&ia[0]);
ia2 =  42;  //错误：ia2是一个指针

//当使用decltype时，上述转换不会发生
decltype(ia) ia3 = {0,1,2,3,4,5,6,7,8,9}; //ia3是数组
ia3 =  p;  //错误：不能用整型指针给数组赋值
ia3[4] = 42; //正确
```

- **指针也是迭代器**

```
int arr[] = {0,1,2,3,4,5,6,7,8,9};
int *p = arr; //p指向arr的第一个元素
++p; //p指向arr[1]
int *e = &arr[10]; //指向arr尾元素的下一个位置的指针
for(int *b = arr; b!=e; ++b)
    cout<<*b<<endl; //输出arr的元素
```

尽管能计算得到尾后指针，但这种用法极易出错。

```
int main()
{
    //begin和end函数定义在iteator头文件中
    int ia[] = { 0,1,2,3,4,5,6,7,8,9 };
    int *beg = std::begin(ia);
    int *end = std::end(ia);

    //寻找第一个负数
    while (beg != end && *beg >= 0)
        ++beg;

    if (beg != end)
        cout << *beg << endl;
    else
        cout << "没找到！"  <<  endl;
}
```

指针运算：

```
constexpr std::size_t sz = 5;//#include <cstddef>
int arr[sz] = { 1,2,3,4,5 };
int *ip = arr; //等价于int *ip=&arr[0]
int *ip2 = ip + 4; //ip2指向arr的尾元素arr[4]

int *p3 = arr + sz; //正确，但不要解引用
int* ip4 = arr + 10; //错误：arr只有5个元素

auto n = std::end(arr) - std::begin(arr); //n=5

int *b = std::begin(arr), *e = std::end(arr);
while  (b<e)//只要两个指针指向同一个数组，或该数组的尾后元素就可以比较
{
      //使用*b
      ++b;
}
```

解引用和指针运算交互：

```
int  ia[]  =  {  0,2,4,6,8  };  //含有5个整数的数组
int last = *(ia + 4); //正确：last = 8
last = *ia + 4; //正确：last = 0 + 4 = 4
```

下标和指针：

```
int ia[] = { 0,2,4,6,8 };
int i = ia[2];//与下面两条等价

int *p = ia;
i = *(p + 2);

int *p = &ia[2];
int j = p[1]; //6
int k = p[-2]; //0，string，vector的下标不可以为负
```

标准库类型限定使用下标必须是无符号型，而内置的下标运算无此要求

C风格字符串：

- C风格字符串不是一种类型，而是为了表达和使用字符串而形成的一种约定俗成的写法。
- C风格字符串的处理函数定义在cstring头文件（string.h的C++版本）中。
  - strlen(p)
  - strcmp(p1,p2)
  - strcat(p1,p2)
  - strcpy(p1,p2)

作为参数的字符串，必须以空字符串结束

```
char ca[] = { 'C','+','+' };
cout << strlen(ca) << endl; //严重错误：ca没有以空字符串结束

string s1 = "A string example";
string s2 = "A different string";
if (s1 < s2) //false：s2小于s2
{
//do something
}

const char ca1[] = "A string example";
const char ca2[] = "A different string";
if  (ca1<ca2)  //未定义的：试图比较两个无关的指针
{
//do something
}

if (strcmp(ca1,ca2)<0)//和两个string对象的比较效果一样
{
//ca1小于ca2
}
```

与旧代码的接口：

```
string s("Hello World"); //s的内容是Hello World
char *str = s; //错误：不能用string对象直接初始化字符的指针。
const char *str=s.c_str(); //正确，c风格的string
```

如果后续的操作改变了s的值，c_str所返回的数组将失效。

```
//使用数组初始化vector对象
int int_arr[] = {0,1,2,3,4,5};
//ivec有6个元素，分别是int_arr中对应元素的副本
vector<int> ivec(std::begin(int_arr),std::end(int_arr));
//拷贝三个元素：int_arr[1]、int_arr[2]、int_arr[3]
vector<int> subVec(int_arr+1,int_arr+4);
```

多维数组：数组的数组

- 严格来说C++没有多维数组。

```
int  ia[3][4];  //大小为3的数组，每个元素是含有4个整数的数组
int arr[10][20][30] ={0}; //将所有元素初始化为0
```

- 初始化

```
//允许使用花括号初始化多维数组
int  ia2[3][4] = {
    {0,1,2,3},
    {4,5,6,7},
    {8,9,10,11}
};
int  ia3[3][4] ={ 0,1,2,3,4,5,6,7,8,9,10,11};
int  ia4[3][4] = {{0},{4},{8}};//{ 0,0,0,0,4,0,0,0,8,0,0,0};
int  ia5[3][4] = {0,3,6,9}; // {0,3,6,9,0,0,0,0,0,0,0,0};
```

- 下标引用

```
//用arr的首元素为ia最后一行的最后一个元素赋值
ia[2][3] = arr[0][0][0];
int(&row)[4] = ia[1];//把row绑定到ia的第二个4元素数组上

constexpr size_t rowCnt = 3, colCnt = 4;
int ia[rowCnt][colCnt]; //12个未初始化的元素
//对于每一行
for (size_t i = 0; i != rowCnt; ++i)
      //对于每一列
      for (size_t j = 0; j != colCnt; ++j)
            ia[i][j] = i * colCnt + j;//将位置索引作为值

size_t cnt = 0;
for (auto& row : ia)
      for (auto& col : row) {
            col = cnt;
            ++cnt;
      }

for  (auto  row :  ia)  //取出来的数组，会被编译器转换为指针
      for (auto col : row) //错误：int *row没有合适的begin函数
            ;

for (auto& row : ia) //row声明为引用，可以避免被自动转换为指针
      for (auto col : row)
            cout << col << endl;
```

- 指针和多维数组

```
int i[3][4];
int(*p)[4] = ia;
p = &ia[2]; //p指向ia的尾元素

//输出ia中每个元素的值，每个内层数组各占一行
//p指向含有4个整数的数组
for (auto p = ia; p != ia+3; ++p){
      //q指向4个整数数组的首元素，也就是说，q指向一个整数
      for (auto q = *p; q != *p + 4; ++q)
            cout << *q << ' ';
      cout << endl;
}

//p指向ia的第一个数组
for (auto p = std::begin(ia); p != std::end(ia); ++p)
      //q指向内层数组的首元素
      for (auto q = std::begin(*p); q != std::end(*p); ++q)
            cout << *q << ' '; //输出q所指向的整数值
      cout << endl;
```

- 类型别名简化多维数组的指针

```
using int_array = int[4];//typedef int int_array[4];

for (int_array *p = ia; p !=ia+3; ++p){
      for (int* q = *p; q != *p + 4; ++q)
            cout << *q << ' ';
      cout << endl;
}
```