第3章 字符串、向量和数组

——C++程序设计

对外经济贸易大学 雷擎
leiqing@uibe.edu.cn

内容

- 3.1 命名空间的using声明
- 3.2 标准库类型string
- 3.3 标准库类型vector
- 3.4 数组
- 3.5 多维数组

3.1 命名空间的using声明

- 命名空间是C++提供的一种解决符号名字冲突的方法。
- 一个命名空间是一个作用域，在不同命名空间中命名相同的符号代表不同的实体。
- 可以自己定义namespace
- using声明

  using namespace spacename;

  using spacename::name;

使用using声明一个名字

```
#include <iostream>
// using declaration; when we use the name cin, we get the one
    from the namespace std
using std::cin;

int main(){
    int i;
    cin >> i; // ok: cin is a synonym for std::cin
    cout << i; // error: no using declaration;
    std::cout << i; // ok: explicitly use cout from namepsace std
    return 0;
}
```

3.2 标准库string

- 标准库string表示可变长的字符序列。

- 程序中使用string，需要在程序开始说明：

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

3.2.1 定义和初始化string对象

- 初始化string对象的方式
  - 直接初始化
  - 拷贝初始化

`string s1`	Default initialization; `s1` is the empty string.
`string s2(s1)`	`s2` is a copy of `s1`.
`string s2 = s1`	Equivalent to `s2(s1)`, `s2` is a copy of `s1`.
`string s3("value")`	`s3` is a copy of the string literal, not including the null.
`string s3 = "value"`	Equivalent to `s3("value")`, `s3` is a copy of the string literal.
`string s4(n, 'c')`	Initialize `s4` with `n` copies of the character `'c'`.

3.2.2 string对象上的操作

- string的操作

`os << s`	Writes `s` onto output stream `os`. Returns `os`.
`is >> s`	Reads whitespace-separated string from `is` into `s`. Returns `is`.
`getline(is, s)`	Reads a line of input from `is` into `s`. Returns `is`.
`s.empty()`	Returns `true` if `s` is empty; otherwise returns `false`.
`s.size()`	Returns the number of characters in `s`.
`s[n]`	Returns a reference to the `char` at position `n` in `s`; positions start at 0.
`s1 + s2`	Returns a `string` that is the concatenation of `s1` and `s2`.
`s1 = s2`	Replaces characters in `s1` with a copy of `s2`.
`s1 == s2`	The `strings` `s1` and `s2` are equal if they contain the same characters.
`s1 != s2`	Equality is case-sensitive.
`<, <=, >, >=`	Comparisons are case-sensitive and use dictionary ordering.

读写string对象

```
int main(){
    string s; // empty string
    cin >> s; // read a whitespace-separated string into s
    cout << s << endl; // write s to the output
    return 0;
}
```

使用getline读取一行

```
int main(){
    string line;
    // read input a line at a time until end-of-file
    while (getline(cin, line))
        cout << line << endl;
    return 0;
}
```

string的empty和size操作

```
// read input a line at a time and discard blank lines
while (getline(cin, line))
    if (!line.empty())
cout << line << endl;

string line;


// read input a line at a time and print lines that are longer than 80 characters
while (getline(cin, line))
    if (line.size() > 80)
cout << line << endl;
```

比较string对象

- ==和!判断是否相等
- <, <=, >, >=判断string对象的大小

string对象相加

- string s1 = "hello, ", s2 = "world\n";
- string s3 = s1 + s2; // s3 is hello, world\n
- s1 += s2; // equivalent to s1 = s1 + s2

- string s1 = "hello", s2 = "world"; // no punctuation in s1 or s2
- string s3 = s1 + ", " + s2 + '\n';

- string s4 = s1 + ", "; // ok: adding a string and a literal
- string s5 = "hello" + ", "; // error: no string operand
- string s6 = s1 + ", " + "world"; // ok: each + has a string operand
- string s7 = "hello" + ", " + s2; // error: can't add string literals

3.2.3 处理string对象中的字符

只能放char，不能放string

`isalnum(c)`	`true` if `c` is a letter or a digit.
`isalpha(c)`	`true` if `c` is a letter.
`iscntrl(c)`	`true` if `c` is a control character.
`isdigit(c)`	`true` if `c` is a digit.
`isgraph(c)`	`true` if `c` is not a space but is printable.
`islower(c)`	`true` if `c` is a lowercase letter.
`isprint(c)`	`true` if `c` is a printable character (i.e., a space or a character that has a visible representation).
`ispunct(c)`	`true` if `c` is a punctuation character (i.e., a character that is not a control character, a digit, a letter, or a printable whitespace).
`isspace(c)`	`true` if `c` is whitespace (i.e., a space, tab, vertical tab, return, newline, or formfeed).
`isupper(c)`	`true` if `c` is an uppercase letter.
`isxdigit(c)`	`true` if `c` is a hexadecimal digit.
`tolower(c)`	If `c` is an uppercase letter, returns its lowercase equivalent; otherwise returns `c` unchanged.
`toupper(c)`	If `c` is a lowercase letter, returns its uppercase equivalent; otherwise returns `c` unchanged.

3.3 标准库类型vector

- vector表示对象的集合，其中所有对象的类型都相同。
- 集合中的每个对象都有一个对应的索引，索引用于访问对象。
- 程序中使用vector，需要在程序开始说明：

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

3.3.1 定义和初始化vector

- 初始化vector对象的方式
  - 拷贝初始化
  - 列表初始化
  - 创建指定数量的元素
  - 值初始化

装的内容的格式

vector的变量名称

`vector<T> v1`	`vector` that holds objects of type `T`. Default initialization; `v1` is empty.
`vector<T> v2(v1)`	`v2` has a copy of each element in `v1`.
`vector<T> v2 = v1`	Equivalent to `v2(v1)`, `v2` is a copy of the elements in `v1`.
`vector<T> v3(n, val)`	`v3` has `n` elements with value `val`.
`vector<T> v4(n)`	`v4` has `n` copies of a value-initialized object.
`vector<T> v5{a,b,c...}`	`v5` has as many elements as there are initializers; elements are initialized by corresponding initializers.
`vector<T> v5 = {a,b,c...}`	Equivalent to `v5{a,b,c...}`.

倍数在前

3.3.2 向vector中添加对象

- push_back
  - 向vector的尾部添加元素

```
vector<int> v2; // empty vector
for (int i = 0; i != 100; ++i)
    v2.push_back(i); // append sequential integers to v2
// at end of loop v2 has 100 elements, values 0 . . . 99
Even though we know we ultimately will have 100 elements, we
    define
```

注意：vector不能用下标形式添加元素

见第七页

3.3.3 其他vector操作

`v.empty()`	Returns `true` if `v` is empty; otherwise returns `false`.
`v.size()`	Returns the number of elements in `v`.
`v.push_back(t)`	Adds an element with value `t` to end of `v`.
`v[n]`	Returns a reference to the element at position `n` in `v`.
`v1 = v2`	Replaces the elements in `v1` with a copy of the elements in `v2`.
`v1 = {a,b,c...}`	Replaces the elements in `v1` with a copy of the elements in the comma-separated list.
`v1 == v2`	`v1` and `v2` are equal if they have the same number of elements and each element in `v1` is equal to the corresponding element in `v2`.
`v1 != v2`	
`<, <=, >, >=`	Have their normal meanings using dictionary ordering.

3.5 数组

- 数组是复合类型，类似于标准库vector的数据结构

- 与vector比较
  - 类似之处，数组中的对象也具有相同的类型
  - 不同之处，数组的大小确定不变

- 如果不清楚元素的确切个数，使用vector

3.5.1 定义或初始化内置数组

长度

- 数组的声明形如a[d]，其中a是数组的名字，d是数组的纬度

```
unsigned cnt = 42; // not a constant expression
constexpr unsigned sz = 42; // constant expression
int arr[10]; // array of ten ints
int *parr[sz]; // array of 42 pointers to int
string bad[cnt]; // error: cnt is not a constant expression
string strs[get_size()]; // ok if get_size is constexpr, error otherwise
```

- 默认情况下，数组的元素被默认初始化

显示初始化数组元素

```
const unsigned sz = 3;
int ia1[sz] = {0,1,2}; // array of three ints with values 0, 1, 2
int a2[] = {0, 1, 2}; // an array of dimension 3
int a3[5] = {0, 1, 2}; // equivalent to a3[] = {0, 1, 2, 0, 0}
string a4[3] = {"hi", "bye"}; // same as a4[] = {"hi", "bye", ""}
int a5[2] = {0,1,2}; // error: too many initializers
```

有考点
不可以用变量作为长度。

字符数组的特殊性

```
char a1[] = {'C', '+', '+'}; // list initialization, no null
char a2[] = {'C', '+', '+', '\0'}; // list initialization, explicit null
char a3[] = "C++"; // null terminator added automatically
const char a4[6] = "Daniel"; // error: no space for the null!
```

不允许拷贝和赋值

```
int a[] = {0, 1, 2}; // array of three ints
int a2[] = a; // error: cannot initialize one array with another
a2 = a; // error: cannot assign one array to another
```

3.5.2 访问数组元素

- 使用数组下标或for语句数组元素。数组下标一般为size_t类型

```
unsigned scores[11] = {}; // 11 buckets, all value initialized to 0 unsigned
    grade;
while (cin >> grade) {
    if (grade <= 100)
            ++scores[grade/10]; // increment the counter for the current cluster
}

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

例：数组元素的引用

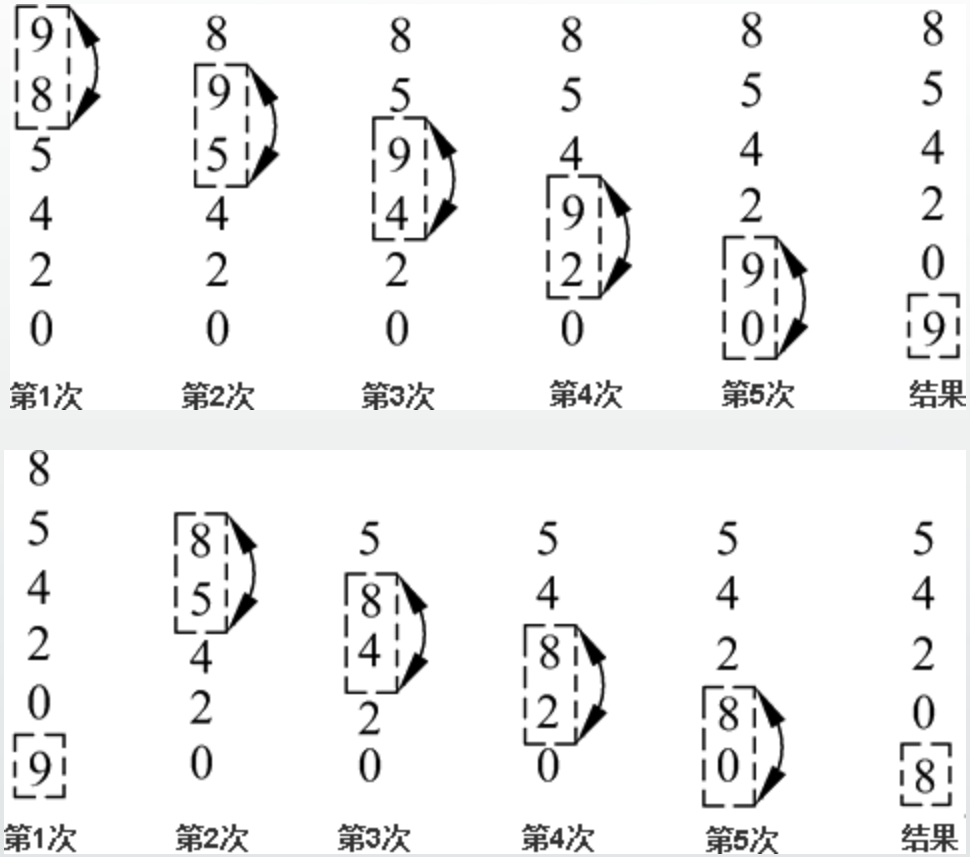
```
#include <iostream>
using namespace std;
int main( ){
  int i, a[10];
  for (i=0;i<=9;i++)
    a[i]=i;
  for (i=9;i>=0;i--)
    cout<<a[i]<<" ";
  cout<<endl;
  return 0;
}
```

例：用数组来处理求Fibonacci数列问题

```
#include <iostream>
#include <iomanip>
using namespace std;
int main( ){
  int i;
  int f[20]={1,1}; //f[0]=1,f[1]=1
  for(i=2;i<20;i++)
    f[i]=f[i-2]+f[i-1]; //在i的值为2时,f[2]=f[0]+f[1],依此类推
  for(i=0;i<20;i++){ //此循环的作用是输出20个数
    if(i%5==0) cout<<endl; //控制换行,每行输出5个数据
    cout<<setw(8)<<f[i]; //每个数据输出时占8列宽度
  }
  cout<<endl; //最后执行一次换行
  return 0;
}
```

```
       1       1       2       3       5
       8      13      21      34      55
      89     144     233     377     610
     987    1597    2584    4181    6765
```

例：用起泡法对10个数排序



```
#include <iostream>
using namespace std;
int main( ){
   int a[11];
   int i, j, t;
   cout<<"input 10 numbers ： "<<endl;
   for (i=1;i<11;i++)  //输入a[1]~a[10]
      cin>>a[i];
   cout<<endl;
   for (j=1;j<=9;j++) //共进行9趟比较
      for(i=1;i<=10-j;i++)//在每趟中要进行(10-j)次两两比较
         if (a[i]>a[i+1]){ //如果前面的数大于后面的数
            t=a[i];a[i]=a[i+1];a[i+1]=t;
         }//交换两个数的位置, 使小数上浮
   cout<<"the sorted numbers ： "<<endl;
   for(i=1;i<11;i++) //输出10个数
      cout<<a[i]<<" ";
   cout<<endl;
   return 0;
}
```

3.5.3 指针和数组

- 指针变量既然可以指向变量，也可以指向数组元素（把某一元素的地址放到一个指针变量中）。所谓数组元素的指针就是数组元素的地址。

```
int a[10];  //定义一个整型数组a，它有10个元素
int *p; //定义一个基类型为整型的指针变量p
p=&a[0]; //将元素a[0]的地址赋给指针变量p，使p指向a[0]
```

- 在C++中，数组名代表数组中第一个元素（即序号为0的元素）的地址。因此，下面两个语句等价：

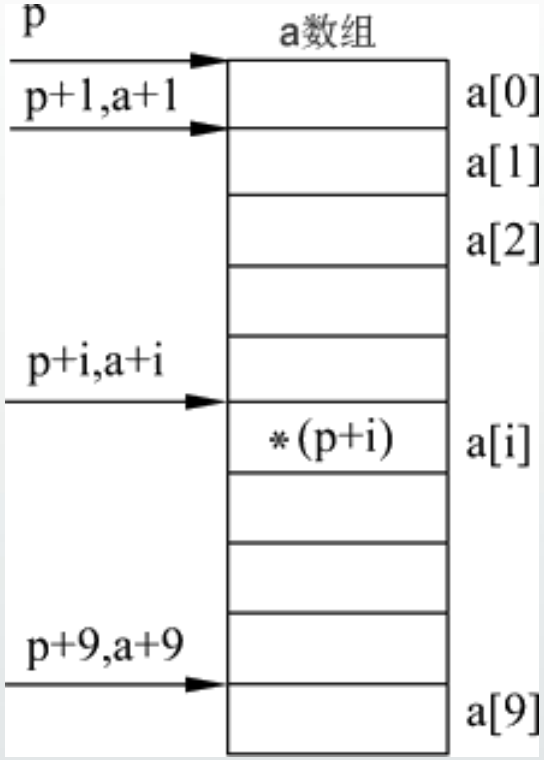
```
p=&a[0];
p=a;
```

- 在定义指针变量时可以给它赋初值：

```
int *p=&a[0]; //p的初值为a[0]的地址
```

也可以写成

```
int *p=a; //作用与前一行相同
```



如果p的初值为&a[0]，则：

- 1) p+i和a+i就是a[i]的地址，或者说，它们指向a数组的第i个元素。
- 2) *(p+i)或*(a+i)是p+i或a+i所指向的数组元素，即a[i]。
- 3) 指向数组元素的指针变量也可以带下标，如p[i]与*(p+i)等价。

3.6 多维数组

- 严格来说，C++没有多维数组，通常说的多维数组是数组的数组

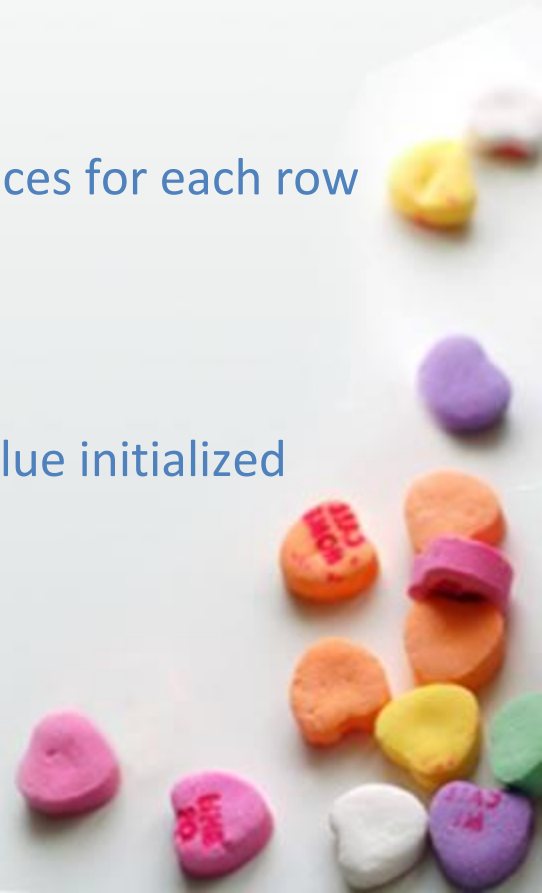
- 具有两个下标的数组称为二维数组。对于二维数组来说，常把第一个为度成为行，第二个为度称为列。
- 例如有3个学生，每个学生有4门课的成绩，成绩数据是一个二维表，如下表所示。

学生序号	课程1	课程2	课程3	课程4	课程5
学生1	85	78	99	96	88
学生2	76	89	75	97	75
学生3	64	92	90	73	56

- 想表示第3个学生第4门课的成绩，就需要指出学生的序号和课程的序号两个因素。在C++中以s[3][4]表示，它代表数据73。

多维数组初始化

```
int ia[3][4] = { // three elements; each element is an array of size 4
    {0, 1, 2, 3}, // initializers for the row indexed by 0
    {4, 5, 6, 7}, // initializers for the row indexed by 1
    {8, 9, 10, 11} // initializers for the row indexed by 2
    };
// equivalent initialization without the optional nested braces for each row
int ia[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
// explicitly initialize only element 0 in each row
int ia[3][4] = {{ 0 }, { 4 }, { 8 }};
// explicitly initialize row 0; the remaining elements are value initialized
int ix[3][4] = {0, 3, 6, 9};
```



多维数组的下标引用

```
constexpr size_t rowCnt = 3, colCnt = 4;
int ia[rowCnt][colCnt]; // 12 uninitialized elements
// for each row
for (size_t i = 0; i != rowCnt; ++i) {
    // for each column within the row
    for (size_t j = 0; j != colCnt; ++j) {
        // assign the element's positional index as its value
        ia[i][j] = i * colCnt + j;
    }
}
```

使用范围for语句处理多维数组

```
isize_t cnt = 0;
for (auto &row : ia) // for every element in the outer array
    for (auto &col : row) { // for every element in the inner array
        col = cnt; // give this element the next value
        ++cnt; // increment cnt
    }
```

实验03.1

- 编写并测试3*3矩阵转置函数，使用数组保存3*3矩阵

使用你

内容

- 3.1 命名空间的using声明
- 3.2 标准库类型string
- 3.3 标准库类型vector
- 3.4 数组
- 3.5 多维数组

Q & A