3：字符串，向量，数组

string 和 vector 都是可变长序列

3.1 using声明

```
using namespace::name // using声明格式
using std::cin;
using std::cout;
```

头文件（.h）的代码内容不应有 using 声明，因为头文件的代码会拷贝到引用他的文件中去，避免名字冲突

3.2 string

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

3.2.1 定义和初始化 stting 对象

```
string s1; // 默认初始化，s1是一个空字符串
string s2(s1); // s2是s1的副本， 等同于 string s2 = s1;
string s3("value"); // s3是字面值的副本，包括最后的空字符'\0' 等同于 string s3 = "value"
string s4 = (n,'c'); // n个' c '
```

3.2.2 string 对象上的操作

```
os<<s; // 将s写入输出流os中，返回os
is>>s; // 从is中读取一个字符串赋给s，以空白分隔，返回is
// 返回os，is是为了链式编程

getlines(is,s) ; // 从is中读取一行赋给s返回is
s.empty();
s.size(); // 返回string::size_type类型的值，
s[n];
s1+s2;
s.substr(pos,n); // 从pos位置开始截取长度为n的子串

/*
    *关于s.size() 返回的 size_type;
    *size_type 是一个无符号类型的值
    *要避免和有符号类型的值混用，如：
    * int i = -1; s.size()小于i // 会被判定为正确，因为-1会被转换成一个较大的无符号值.
*/
```

string 对象的比较

```
// 如果两个string对象在某些对应的位置上不一致，则string对象比较的结果是第一对相异字符比较的结果
string str = "hello";
string phrase = "hello world";
string slang = "Hiya";
// 根据比较规则，str 小于 phrase,slang最大
```

string加法

```
// 当把string对象和'char'或"string"相加时，必须确保+两侧有一个strign类型
string s4 = s1 + ","; // 正确
string s5 = "hello" + ","; // 错误
string s6 = "hello" + ","+s2; // 错误

// 字符串字面值和string是不同的类型
```

3.2.3 处理string对象中的字符（char）

cctype头文件中函数，c++:cctype， c:ctype.h

```
isalnum(c); // 是数字或字母时为真
isalpha(c); // 是字母时为真
iscntrl(c); // 是控制字符时为真 \n \t \f（换页）
isdigit(c); //
isgraph(c); // 可打印且不是空格时为真
islower(c);
isprict(c); // 可打印时为真
ispunct(c); // 标点符号为真
isspace(c); // 空白时为真
isupper(c);
isxdigit(c); // 十六进制数字时为真
tolower(c);
toupper(c);
```

处理每个字符

```
// 例1 统计字符串的标点符号的个数：
string s("hello world!!!");
decltype(s.size()) punt_cnt = 0;  // decltype(s.size()) 类型是string::size_type
for(auto c:s)
    if(ispunct(c)
        ++punt_cnt;
cout<<punct_cnt<<endl;

// 例2 把字符串改写成大写
string s1("hello world);
for(auto & c:s) // 引用为了更改原s1中的值
    c = toupper(c);
cout<<s<<endl;
```

只处理一部分字符，使用下标或者迭代器

```
    // 下标只能访问[0,s.size) 范围的变量，访问该范围外或者访问空字符串会产生不可知后果，
所以访问时行检查字符串是否为空
    // 下标的值只要是整形即可，如果带符号将会被转换成无符号类型string::size_type（>=0）
    // 例：依次处理s中的字符改成大写直到处理完全或遇到一个空白
    for(decltype(s.size()) index = 0;index!=s.size() && isspace(s[index]);++index)
        s[index] = toupper(s[index]);
```

使用下标随机访问

```
    // 例：编写一个程序把0~15间的十进制数字换成对应的十六进制形式
    const string hexdigits = "0123456789ABCDEF";
    string result;
    string::size_type index;
    while(cin>>n)
        if(index小于hexdigits.size())
           result += hexdigits[index];
    cout<< result<<endl;
```

3.3 标准库类型 vector， 对象的集合（引用不是对象）

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

// 对于vector<vector<> >对象的老式声明，右边的>>之间要加空格
```

3.3.1 定义和初始化 vector 对象

```
vector<T> v1; // 执行默认初始化(内置类型的int,string初始化0和空)

vector<T> v2(v1); // 拷贝初始化,等价于vector<T> v2 = v1;
vector<T> v3(n,val); // 初始化为n个val
vector<T> v4(n); // 初始化大小为n，值默认初始化

vector<T> v5{a,b,c}; // 列表初始化，等价于 vector<T> v5 = {a,b,c}
// 关于列表初始化，程序会尽可能的把{}内的值当成元素初始值的列表来处理
// 但当提供的值不能用来初始化列表时，可能向下兼容当成（）来处理，如
vector<string> v7{10}; //  vectoc<string> v7(10);
vector<string>v8{10,"hi"}; // vector<string> v8(10,"hi");
```

3.3.2 向 vector 对象中添加元素

根据 vector 对象能高效增长的特性，先定义空容器再 push_back() 效率更高

？如果循环体内包含有向 vector 对象添加元素的语句，则不能使用范围for循环？，因为这会改变.end

3.3.3其他 vector 操作

- v.size() 返回的类型是 vector 定义的 size_type 类型，下标也是， vector<int>::size_type
- vector 和 string 对象的下标运算符只能访问已存在的元素，不能用来添加元素。
- 确保下标合法的一种有效方法，就是采用范围for语句

3.4 迭代器介绍

3.4.1标题忘了

- string vector`等标准库容器都可以使用迭代器，
- 有效迭代器指向某个元素或尾元素的下个位置
- 如果容器为空，begin和end都返回指向尾后的迭代器
- 解引用 如： *iter 或 iter->
- 迭代器一般都支持的运算符：++ -- == != 没有>< （可爱，哈哈）
- 在for循环中常用 iter != v.end
- .cbegin``.cend 是两个const版本迭代器，返回的是 const_iterator ，只能用于访问元素值，不能用来修改元素值
- vector迭代器失效的情况：vector容量改变

  所以不能在范围for循环中更改容器元素

3.4.2 迭代器运算

前提是迭代器有效，即：迭代器指向某个元素或尾元素的下个位置

所有标准库容器的迭代器都支持一般运算符，即： ++ -- != == ,但对于string和vector提供了更多的运算符

```
iter +/- n // 前后移动若干个位置，结果迭代器仍有效
iter1 - iter2 // 结果是它们之间的距离
\   \-  <  <= //前面的迭代器小，后面的迭代器大
```

3.5数组

- 与vector类似，通过位置访问，存储类型相同的对象
- 与vector不同，数组大小固定，不能随意增加元素

  若不清楚元素的确切个数，使用vector

3.5.1 定义和初始化内置类型的数组

- 数组的声明

```
int a[d]; // a是数组名，d是数组维度（常量表达式）
```

- 当对数组进行列表初始化时，可以忽略维度

```
int a2[] = {0,1,2}; // 自动推导数组维度大小为3
int a3[5] = {0,1,2}; // 等价于 int a3[5] = {0,1,2,0,0};
```

- 字符数组的特殊性***当使用字符串字面值初始化字符数组时，注意空字符的存在***

```
char a3[] = "c++"; // 自动添加空字符，'\0',a3维度为4
const char a4[6] = "daniel"; // 错误，a4有6位置，但"daniel"有7个元素
```

- 数组不允许拷贝和赋值

```
int a2[] = a; // 错误
a2 = a; // 错误
```

- 较复杂的声明（由内向外读）

```
int *ptrs[10]; //ptrs 是含有10个整形指针的数组
int (*ptr)[10] = &arr; // ptr是一个指针，指向一个含有10个整数的数组
int  (&arrRef)[10] = arr; // arrRef 是一个引用，引用一个含有10个整数的数组
```

3.5.2 访问数组元素（范围for或下标）

- 数组自动转化成指针

```
string *p = nums; // 等价于：p = &nums[0];
// 大多数表达式中，使用数组类型的对象是使用一个指向数组首元素的指针如：
int ia[] = {0.1.2};
auto ia2(ia); // ia2 是一个指向ia首元素的整型指针
ia2 = 42; // 错误，ia2是一个指针

// decltype(ia)，返回的是一个整型数组类型，decltype 与 auto 的一个区别也是在此
```

- 指针也是迭代器：指向数组元素的指针可以当成迭代器使用

```
int arr[] = {0.1,2,3,4,5,6,7,8,9};
int *e = &arr[10]; // e是尾后指针，指向一个不存在的元素，类似尾迭代器
// 不能对尾后指针进行解引用或者递增操作

// c++11中获取首，尾指针的更安全的方法：
int *beg = begin(ia);
int *end = end(ia);
```

- 指针运算,指向数组元素的指针可以解引用，++，比较，+/- n,指针相减。。。

```
constexpr size_t sz = 5;
int arr[sz] = {1,2,3,4,5};
itn *p = arr+sz; // p是尾后指针
auto n = end(arr) -begin(arr); // n=5=元素数量,n的类型为ptrdiff_t，带符号类型，与机器有关

// 利用指针遍历数组：
int *b = arr,*e = arr+sz;
while(b<e){
    ++b;
}
```

- 内置的下标运算符所用的索尼值不是无符号类型，这一点与vector和string不同（他们是无符号类型）

```
int *p = &ia[2]; // p指向索引为2的元素
int j = p[1]; // p[1]等价于*(p+1),j = ia[3];
int k = p[-2]; // k = ia[0];
```

3.5.4 c风格字符串（3.5.3呢？）

- 什么是c风格字符串： ***字符串字面值（等价于以空字符'\0'结束的字符数组）***
- c标准库string函数（cstring)

```
strlen(p); // 返回p的长度，空字符不计算
strcmp(p1,p2); // 根据p1-p2返回正，0，负
strcat(p1,p2); // 将p2附加到这p1上，返回p1
strcpy(p1,p2); // 将p2拷贝到p1，返回p1

// 传入上述函数的指针，必须指向以空字符\0结束的字符数组
char ca[] = {'a','b'};
cout<<strlen(ca)<<endl; // 错误，ca不以'\0'结束
```

- 比较字符串

```
// 对于标准库string对象，可以直接比较
string s1 = "a string example";
string s2 = "a different string";
s1<s2; // true

// 对比c风格字符串比较的是指针，而不是字符串本身
const char ca1[] = "A string example";
const char ca2[] = "A different string";
ca1 < ca2; // 错误，试图比较两个无关的指针

// 改正：
strcmp(ca1,ca2)<0;
```

- 连接字符串

```
// 对于string对象来说，可以直接相加
string largeStr = s1+" "+s2;
// 对于c风格字符串，ca1+ca2相当于把两个指针相加，非法，拼接可以通过strcat和strcpy实现
strcpy(largeStr,ca1);
strcat(largeStr," ");
strcat(largeStr,ca12);
// 注意：largeStr大小要给足，而且其所有内容一旦改变，就必须检查空间是否足够
```

使用标准库string更安全更高效

3.5.5 与旧代码的接口

- 混用string对象和c风格字符串

```
// 用字符串字面值初始化string对象：
string s("Hello world"); // 字符串字面值等价于空字符结尾的字符数组
// 同样的，在string相关的操作中任何以空字符结尾的字符数组都等价于字符串字面值

// 反过来，不能用string对象初始化c风格字符串
char *str = s; // 错误
const char *str = s.c_str(); // 可以，.c_str返回一个指针，指向以\0结尾的字符数组
// 注意：调用s.c_str()函数后再改变s，可能会使之前的返回数组失去效用，要用的话提前拷贝c_
str();
```

- 使用数组初始化vector对象

```
// 数组不能被另一个数组或vector初始化，但可以用数组初始化vector对象
vector<int> vec(begin(int_arr),end(int_arr));
```

总结：多用vector，迭代器，string

3.6 多维数组

- 多维数组的初始化：

```
int ia[3][4] = {
    {0,1,2,3},
    {4,5,6,7},
    {8,9,10,11}
};

// 等价于：
int ia[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

// 显式初始化多维数组每行的首元素
int ia[3][4] = {{0},{4},{8}};

// 显式初始化第一行，其他默认初始化
int ix[3][4] = {0,3,6,9 };


// 用范围for循环遍历多维数组，外层循环一定要用引用，防止数组被自动转换成指针
for(const auto &row :ia)
    for(auto col:row)
```

- 指针与多维数组

```
int ia[3][4];
int(*p)[4] = ia; // p是一个指针，指向ia的第一行或：指向含有4个整数的数组

// 注意：int *ip[4]; ip是整形指针的数组(有无括号的区别)

for(auto p = begin(ia);p!=end(ia);++p){
    for(auto q = begin(*p);q!=end(*p);++q){
```