Iterator

\*iter //对iter进行解引用，返回迭代器iter指向的元素的引用

iter->men //对iter进行解引用，获取指定元素中名为men的成员。等效于(\*iter).men

++iter //给iter加1，使其指向容器的下一个元素

iter++

--iter //给iter减1，使其指向容器的前一个元素

iter--

iter1==iter2 //比较两个迭代器是否相等，当它们指向同一个容器的同一个元素或者都指向同同一个容器的超出末端的下一个位置时，它们相等

iter1!=iter2

iter+n //在迭代器上加（减）整数n，将产生指向容器中钱前面（后面）第n个元素的迭代器。新计算出来的迭代器必须指向容器中的元素或超出容器末端的下一个元素

iter-n

iter1+=iter2 //将iter1加上或减去iter2的运算结果赋给iter1。两个迭代器必须指向容器中的元素或超出容器末端的下一个元素

iter1-=iter2

iter1-iter2 //两个迭代器的减法，得出两个迭代器的距离。两个迭代器必须指向容器中的元素或超出容器末端的下一个元素

>,>=,<,<= //元素靠后的迭代器大于靠前的迭代器。两个迭代器必须指向容器中的元素或超出容器末端的下一个元素

1) 正向迭代器，定义方法如下：

容器类名::iterator 迭代器名;

2) 常量正向迭代器，定义方法如下：

容器类名::const\_iterator 迭代器名;

3) 反向迭代器，定义方法如下：

容器类名::reverse\_iterator 迭代器名;

4) 常量反向迭代器，定义方法如下：

容器类名::const\_reverse\_iterator 迭代器名

通过迭代器可以读取它指向的元素，\*迭代器名就表示迭代器指向的元素。通过非常量迭代器还能修改其指向的元素。

迭代器都可以进行++操作。反向迭代器和正向迭代器的区别在于：

对正向迭代器进行++操作时，迭代器会指向容器中的后一个元素；

而对反向迭代器进行++操作时，迭代器会指向容器中的前一个元素。

rbegin 成员函数返回指向容器中最后一个元素的迭代器，rend 成员函数返回指向容器中第一个元素前面的位置的迭代器，因此本循环实际上是从后往前遍历整个数组。反向迭代器进行++操作后，会指向容器中的上一个元素

for (vector<int>::reverse\_iterator j = v.rbegin(); j != v.rend(); ++j)

cout << \*j << " ";

String

拷贝初始化用等号，不使用等好的是直接初始化。

string strs //生成空字符串

string s(str) //生成字符串str的复制品

string s=str//生成字符串str的复制品

string s3=“value”//s3是字面值”value“的副本，除了字面值最后的那个空字符外

string s3=（“value”）//与上面的等价

strings(num, c) //生成一个字符串，包含num个c字符

string s(str, stridx) //将字符串str中始于stridx的部分作为构造函数的初值

string s(str, strbegin, strlen) //将字符串str中始于strbegin、长度为strlen的部分作为字符串初值

strings(strs, beg, end) //以区间[beg, end]内的字符作为字符串s的初值

string s(cstr) //以Char字符数组类型cstr作为字符串s的初值

string s=cstr//以Char字符数组类型cstr作为字符串s的初值，cstr是数组名

string s(cstr,char\_len) //以Char类型cstr的前char\_len个字符串作为字符串s的初值

1. size() 和 length()：这两个函数会返回 string 类型对象中的字符个数，且它们的执行效果相同
2. 字符串中元素的访问是允许的，一般可使用两种方法访问字符串中的单一字符：下标操作符[] 和 成员函数at()。两者均返回指定的下标位置的字符,并且是返回字符的引用。

下标操作符 [] 在使用时不检查索引的有效性，如果下标超出字符的长度范围，会示导致未定义行为。对于常量字符串，使用下标操作符时，字符串的最后字符（即 '\0'）是有效的。对应 string 类型对象（常量型）最后一个字符的下标是有效的，调用返回字符 '\0'。

函数 at() 在使用时会检查下标是否有效。如果给定的下标超出字符的长度范围，系统会抛出 out\_of\_range 异常。

1. string类的迭代器处理：

string类提供了向前和向后遍历的迭代器iterator，迭代器提供了访问各个字符的语法，类似于指针操作，迭代器不检查范围。

用string::iterator或string::const\_iterator声明迭代器变量，const\_iterator不允许改变迭代的内容。常用迭代器函数有：

const\_iterator begin()const;

iterator begin(); //返回string的起始位置

const\_iterator end()const;

iterator end(); //返回string的最后一个字符后面的位置

const\_iterator rbegin()const;

iterator rbegin(); //返回string的最后一个字符的位置

const\_iterator rend()const;

iterator rend(); //返回string第一个字符位置的前面

rbegin和rend用于从后向前的迭代访问，通过设置迭代器string::reverse\_iterator,string::const\_reverse\_iterator实现

1. string类的删除函数

iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置

iterator erase(iterator it);//删除it指向的字符，返回删除后迭代器的位置

string &erase(int pos = 0, int n = npos);//删除pos开始的n个字符，返回修改后的字符串

1. string类的插入函数：

string &insert(int p0, const char \*s);

string &insert(int p0, const char \*s, int n);

string &insert(int p0,const string &s);

string &insert(int p0,const string &s, int pos, int n);

//前4个函数在p0位置插入字符串s中pos开始的前n个字符

string &insert(int p0, int n, char c);//此函数在p0处插入n个字符c

iterator insert(iterator it, char c);//在it处插入字符c，返回插入后迭代器的位置

void insert(iterator it, const\_iterator first, const\_iterator last);//在it处插入[first，last）之间的字符

void insert(iterator it, int n, char c);//在it处插入n个字符c

1. string类的替换函数：

string &replace(int p0, int n0,const char \*s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const char \*s, int n);//删除p0开始的n0个字符，然后在p0处插入字符串s的前n个字符

string &replace(int p0, int n0,const string &s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const string &s, int pos, int n);//删除p0开始的n0个字符，然后在p0处插入串s中从pos开始的n个字符

string &replace(int p0, int n0,int n, char c);//删除p0开始的n0个字符，然后在p0处插入n个字符c

string &replace(iterator first0, iterator last0,const char \*s);//把[first0，last0）之间的部分替换为字符串s

string &replace(iterator first0, iterator last0,const char \*s, int n);//把[first0，last0）之间的部分替换为s的前n个字符

string &replace(iterator first0, iterator last0,const string &s);//把[first0，last0）之间的部分替换为串s

string &replace(iterator first0, iterator last0,int n, char c);//把[first0，last0）之间的部分替换为n个字符c

string &replace(iterator first0, iterator last0,const\_iterator first, const\_iterator last);//把[first0，last0）之间的部分替换成[first，last）之间的字符串

ARRY

array<T,N> 模板定义了一种相当于标准数组的容器类型。它是一个有 N 个 T 类型元素的固定序列。除了需要指定元素的类型和个数之外，它和常规数组没有太大的差别。显然，不能增加或删除元素

使用 array 容器类型时，需要在源文件中包含头文件 array。数组容器非常好用，这里有一个示例，展示了如何创建具有 100 个 double 型元素的 array<>:

std::array<double,100> data;

如果定义了一个 array 容器，却没有为元素指定初始值，那么元素就不会被初始化；但是可以如下所示，将它们初始化为 0 或者和默认元素类型等效的值：

std::array<double, 100> data {};

使用该语句初始化后，容器中所有的元素都会变为 0.0。形参 N 必须是一个常量表达式(constant expression)并且容器中元素的个数不能变。当然，当创建 array 容器的实例时，要像创建常规数组那样，对元素进行初始化：

std::array<double, 10> values {0.5, 1.0, 1.5, 2.0}

1. size()//返回数组的大小
2. empty()//判断数组是否为空
3. max\_size()//返回数组的最大值；array<int,100> max\_size返回的值是100
4. 访问元素 可以用下标，也可以用at()函数返回值；front（）函数返回第一个元素，back()返回最后一个元素，以上返回的均是引用类型；data()返回T类型的指针；
5. Void fill(const type value)函数 将数组元素全都设置为value
6. 交换函数void swap(),交换的两个数组大小必须相等

array<int, 5> arr1 = { 1, 2, 3, 4, 5 };

array<int, 5> arr2 = { 5, 4, 3, 2, 1 };

swap( arr1, arr2 );

arr1.swap( arr2 ); // 等价于： swap( arr1, arr2 );