面向对象编程基础

本课程入选教育部产学合作协同育人项目
课程主页:**http://cpp.njuer.org**
课程老师:陈明 **http://cv.mchen.org**

```
ppt和代码下载地址
git clone https://gitee.com/cpp-njuer-org/book
```

第3章

字符串、向量和数组

- 命名空间的using声明
- 标准库类型 string
- 标准库类型 vector
- 迭代器介绍
- 数组
- 多维数组

命名空间的USING声明

- 使用某个命名空间：例如 using std::cin表示使用命名空间std中的名字cin。

```
//using1.cpp
#include <iostream>
// using 声明，当我们使用cin时，从命名空间std中获取
using std::cin;

int main()
{
    int i;
    cin >> i;//正确 cin和std::cin含义相同
    //cout<< i;//错误 没有对应的using声明，必须使用完整的名字
            //error: ‘cout’ was not declared in this scope; did you mean ‘std::cout’?
    std::cout<< i;//正确 显示地从std中使用cout
    return 0;
}
```

每个名字都需要独立的USING声明

```cpp
//using2.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;
}
```

- 头文件中不应该包含using声明。
  - 头文件的内容会拷贝到所有引用它的文件中去，可能产生名字冲突。
  - 附表A-1列出标准库内部分名字和对应的头文件。

标准库类型 STRING

- 标准库类型string表示可变长的字符序列。
- 使用string类型必须首先包含string头文件
- 标准库类型对一般应用场合有足够的效率

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

string*对象不同于字符串字面值。*

定义和初始化STRING对象

```
//初始化string对象的方式：
string s1;                  // 默认初始化，s1是个空字符串
string s2(s1);              // s2是s1的副本
string s2 = s1;             // 等价于s2(s1)，s2是s1的副本
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";        // 等价于s3("value")，s3是字面值"value"的副本
string s4(n, 'c');          // 把s4初始化为由连续n个字符c组成的串
string s5 = "hiya";         // 拷贝初始化
string s6("hiya");          // 直接初始化
string s7(10, 'c');         // 直接初始化，s7的内容是cccccccccc
string s8=string(10,'c');//拷贝初始化，s8的内容是cccccccccc 等价于
string temp(10,'c');        // temp的内容是cccccccccc
string s8= temp;             // 将temp拷贝给s8。
/**
拷贝初始化（copy initialization）：
     使用等号=将一个已有的对象拷贝到正在创建的对象。
直接初始化（direct initialization）：
     通过括号给对象赋值。
**/
```

STRING对象上的操作

```
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，字符串以空白分割，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
s[n];             // 返回s中第n个字符的引用，位置n从0计起
s1+s2;            // 返回s1和s2连接后的结果
s1=s2;            // 用s2的副本代替s1中原来的字符
s1==s2;           // 如果s1和s2中所含的字符完全一样，则它们相等；
                  // string对象的相等性判断，对字母的大小写敏感。
s1!=s2;           // 同上
<, <=, >, >=      // 利用字符在字典中的顺序进行比较，
                  // 且对字母的大小写敏感
```

读写STRING对象

```cpp
//stringIO.cpp
#include <string>
using std::string;
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

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;
}
```

多个输入输出可以连写

```cpp
//stringIO2.cpp
#include <string>
using std::string;

#include <iostream>
using std::cin; using std::cout; using std::endl;

int main()
{
    string s1, s2;

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

    return 0;
}
```

- 执行读操作>>：忽略掉开头的空白（包括空格、换行符和制表符），直到遇到下一处空白为止。

读取未知数量的STRING对象

```cpp
//stringIO3.cpp
#include <iostream>
using std::cin; using std::cout; using std::endl;

#include <string>
using std::string;

int main()
{
    string word;
    while (cin >> word)        // read until end-of-file. ctrl+d
        cout << word << endl; // write each word followed by a new line
    return 0;
}
```

使用GETLINE读取一整行

```
//stringGetline.cpp

#include <string>
using std::string; using std::getline;
#include <iostream>
using std::cin; using std::cout; using std::endl;
int main()
{
    string line;
    // read input a line at a time until end-of-file
    // line 中不包含换行符
    while (getline(cin, line))
        cout << line << endl;
    return 0;
}
```

- getline：读取一整行，**包括空白符。**

STRING的EMPTY和SIZE操作

```
//stringEmpty.cpp
#include <string>
#include <iostream>
using std::string;
using std::cin; using std::cout; using std::endl;
int main()
{
    string line;
    // 每次读入一行，遇到空行直接跳过
    while (getline(cin, line))
        if (!line.empty())
            cout << line << endl;
    return 0;
}
```

STRING的EMPTY和SIZE操作

```
//stringSize.cpp
#include <string>
using std::string; using std::getline;
#include <iostream>
using std::cin; using std::cout; using std::endl;
int main()
{
    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;
    return 0;
}
```

STRING::SIZE_TYPE类型

```
// s.size()返回string::size_type类型，是一个**无符号**类型的值，
//不要和`int`混用
auto len = line.size(); //len的类型是string::size_type类型
```

- 假设n是一个负值int，则line.size()<n几乎肯定是true，因为负值n会自动转化成一个比较大的无符号值
- 表达式里有size()函数就不要使用int，避免混用int和unsigned可能带来的问题

比较STRING对象

==和!=检验两个string对象相等或不相等
<,<=,>,>=按（大小写敏感的）字典序：
- 长度不同，较短的string每个字符与较长的string对应位置字符相同，则较短的string小于较长的string。
- 若在某对应位置不一致，则比较结果是第一对相异字符比较结果。

比较STRING对象

```cpp
//stringCompare.cpp
#include <string>
using std::string;
#include <iostream>
using std::cout; using std::endl;
int main()
{
    string str = "Hello";
    string phrase = "Hello World";
    string slang  = "Hiya";
    if (str < phrase) cout << "Hello < Hello World" << endl;
    if (slang > str) cout << "Hiya > Hello" << endl;
    if (slang > phrase) cout << "Hiya > Hello World" << endl;
    return 0;
}
//Hello < Hello World
//Hiya > Hello
//Hiya > Hello World
```

为STRING对象赋值

```
//对于string类而言，允许把一个对象的值赋给另外一个对象
string st1(10,'c'),st2; //st1是ccccccccccc,st2是空字符串
st1 = st2;              //用st2的副本替换st1的内容
                        //此时st1和st2都是空字符串
```

字面值和STRING对象相加

```
//标准库允许把字符字面值和字符串字面值转换成string对象
//所以在需要stirng对象的地方可以使用这两种字面值替代
//stringAdd.cpp
#include <iostream>
using std::cout; using std::endl;
#include <string>
using std::string;
int main()
{
    string s1  = "hello, ", s2 = "world";
    string s3 = s1 + "," + s2 + '\n';   // s3 is hello, world\n
    cout << s1<< s2 <<endl << s3 << endl;
    string s4 = s1+"," ;                  // ok,string+","
    //string s5 = "hello"+",";            //error, " "+" "
    string s6=s1+","+"world";
    //string s7="hello"+","+s2;           //error, " "+" "
    cout<<s4<<endl<<s6<<endl;
    return 0;
}
// 字符串字面值和string是不同的类型。
```

练习

编写一段程序从标准输入中一次读入一行，然后修改该程序使其一次读入一个词。

```
//一次读入一行：
#include <iostream>
#include <string>
using std::string;using std::cin;using std::cout;
using std::endl;using std::getline;
int main()
{
    string s;
    while (getline(cin,s))
    {
        cout << s << endl;
    }
    return 0;
}
```

练习

编写一段程序从标准输入中一次读入一行，然后修改该程序使其一次读入一个词。

```
//一次读入一个词
#include <iostream>
#include <string>
using std::string;using std::cin;using std::cout;
using std::endl;using std::getline;
int main()
{
    string s;
    while (cin >> s)
    {
        cout << s << endl;
    }
    return 0;
}
```

练习

编写一段程序从标准输入中读入多个字符串并将他们连接起来，输出连接成的大字符串。
然后修改上述程序，用空格把输入的多个字符串分割开来。

```
//连接
#include <iostream>
#include <string>
using std::string;using std::cin;
using std::cout;using std::endl;
int main()
{
    string result, s;
    while (cin >> s)
    {
        result += s;
    }
    cout << result << endl;
    return 0;
}
```

练习

编写一段程序从标准输入中读入多个字符串并将他们连接起来，输出连接成的大字符串。
然后修改上述程序，用空格把输入的多个字符串分割开来。

```
//分割
#include <iostream>
#include <string>
using std::string;using std::cin;
using std::cout;using std::endl;
int main()
{
    string result, s;
    while (cin >> s)
    {
        result += s + " ";
    }
    cout << result << endl;
    return 0;
}
```

处理STRING对象中的字符

- cctype头文件定义了一组标准库函数
    - 关于字符特性。
- C++修改了c的标准库，名称为去掉.h，前面加c。
    - c++版本为cctype，c版本为ctype.h
    - 尽量使用c++版本的头文件，即cctype

cctype**头文件中定义了一组标准函数：**

```
isalnum(c)   当c是字母或数字时为真
isalpha(c)   当c是字母时为真
iscntrl(c)   当c是控制字符时为真
isdigit(c)   当c是数字时为真
isgraph(c)   当c不是空格但可以打印时为真
islower(c)   当c是小写字母时为真
isprint(c)   当c是可打印字符时为真
ispunct(c)   当c是标点符号时为真
isspace(c)   当c是空白时为真
             （空格、横向制表符、纵向制表符、回车符、换行符、进纸符）
isupper(c)   当c是大写字母时为真
isxdigit(c)  当c是十六进制数字时为真
tolower(c)   当c是大写字母，输出对应的小写字母；否则原样输出c
toupper(c)   当c是小写字母，输出对应的大写字母；否则原样输出c
```

处理每个字符 使用基于范围FOR语句

- 语法形式

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

expression是一个对象，用于表示一个序列

declaration负责定义一个变量，用于访问序列中基础元素

每次迭代，declaration部分变量初始化为expression部分的下一个元素值

- 如

```
for (auto c: str)
    statement
for (auto &c: str)//使用引用直接改变字符串中的字符。
    statement
```

处理每个字符 使用基于范围FOR语句

```cpp
//stringFor.cpp
#include <iostream>
using std::cout; using std::endl;
#include <string>
using std::string;
int main()
{
    string str("some string");
    //每行输出str中的一个字符
    for(auto c: str)
        cout<<c<<endl;
    return 0;
}
```

处理每个字符 使用基于范围FOR语句

```
s
o
m
e

s
t
r
i
n
g
```

```cpp
//stringFor2.cpp
//统计标点符号
#include <string>
using std::string;
#include <cctype>
using std::ispunct;
#include <iostream>
using std::cout; using std::endl;
int main()
{
    string s("Hello World!!!");
    // punct_cnt has the same type that s.size returns
    decltype(s.size()) punct_cnt = 0;
    // count the number of punctuation characters in s
    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;
    return 0;
}
//3 punctuation characters in Hello World!!!
```

使用范围FOR语句改变字符串中的字符

```
//stringFor3.cpp
//字符串改大写
#include <string>
using std::string;
#include <cctype>
using std::isupper; using std::toupper;
#include <iostream>
using std::cout; using std::endl;
int main()
{
    string s("Hello World!!!");
    // convert s to uppercase
    string orig = s;
    for (auto &c : s)   // for every char in s (note: c is a reference)
        // c is a reference, so this assignment changes the char in s
        c = toupper(c);
    cout << s << endl;
    return 0;
}
//HELLO WORLD!!!
```

只处理一部分字符

下标运算符[]
- 接受输入参数 string::size_type 类型的值
- 返回值 该位置上字符的引用
- 下标大于等于0 小于s.size()

只处理一部分字符

```
//stringIndex.cpp
//字符改大写
#include <string>
using std::string;
#include <cctype>
using std::toupper;
#include <iostream>
using std::cout; using std::endl;
int main()
{
    string s("some string");
    if(!s.empty())
        cout<<s[0]<<endl;//s
    if(!s.empty())
        s[0]=toupper(s[0]);
    cout<<s<<endl;//Some string
    return 0;
}
```

使用下标进行迭代

```
//stringIndex2.cpp
//首个单词字符改大写
#include <string>
using std::string;
#include <cctype>
using std::toupper;
#include <iostream>
using std::cout; using std::endl;
int main()
{
    string s("some string");
    for(decltype(s.size()) index = 0;
        index != s.size() && !isspace(s[index]);++index) {
        s[index] = toupper(s[index]);
    }
    cout << s << endl;//SOME string
    return 0;
}
// && 逻辑与运算
```

使用下标进行随机访问

```cpp
//stringIndex3.cpp 转16进制
//str[x],[]输入参数为string::size_type类型，给出int整型也会自动转化为该类型
#include <iostream>
using std::cin; using std::cout; using std::endl;
#include <string>
#include <cstddef>
using std::size_t;using std::string;
int main()
{
    const string hexdigits = "0123456789ABCDEF";  // possible hex digits
    cout << "Enter a series of numbers between 0 and 15"
         << " 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;
}
```

练习

```cpp
//编写一段程序，读入一个包含标点符号的字符串，将标点符号去除后输出字符串剩余的部分。
#include <iostream>
#include <string>
#include <cctype>
using std::string;using std::cin;
using std::cout;using std::endl;
int main()
{
    string s = "!!!hello world!!!";
    string result;
    for (auto x : s)
    {
        if (!ispunct(x))
        {
            result += x;
        }
    }
    cout << result << endl;
    return 0;
}
```

标准库类型VECTOR

vector是一个容器，也是一个类模板
- 容器：包含其他对象。
- 类模板：本身不是类，但可以实例化出一个类。
    - vector是一个模板， vector<int>是一个类型。
- 类模板名字后面跟一对尖括号，在括号内放上信息来指定类型，
    - 如vector<int> ivec。
    - vector的元素还可以是vector
        - 如vector<vector<int>>
        - 如vector<vector<int> > 过去的写法，需要加空格

```
//使用vector必须包含适当的头文件
#include <vector>
using std::vector;
```

定义和初始化VECTOR对象

```
vector<T> v1;                //v1是一个空vector，它潜在的元素是T类型的，执行默认初始化
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)，v2中包含v1所有元素的副本
vector<T> v3(n, val);        // v3包含了n个重复的元素，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，每个元素被赋予相应的初始值
vector<T> v5={a, b, c...};   //等价于v5{a, b, c...}

- 默认初始化：  vector<string> svec; //svec 不含元素
vector<int> ivec;            //初始状态为空
vector<int> ivec2(ivec);     //ivec 拷贝ivec2
vector<int> ivec3(ivec);     //ivec 拷贝 ivec3
vector<string> svec(ivec2); //错误 string 不是int
```

列表初始化VECTOR对象

- 列表初始化： `vector<string> v{"a", "an", "the"};` （C++11）
  - 使用=只能提供一个初始值
  - 类内初始值，只能用拷贝初始化或使用花括号形式初始化
  - 提供初始值元素值列表，初始值放入花括号，进入列表初始化

创建指定数量的元素

```
vector<int> ivec(10,-1);          //10个int类型的元素，每个都被初始化为-1
vector<string> svec(10,"hi!"); //10个string类型的元素，
                                  //每个都被初始化为"hi!"
```

值初始化

```
// 可以只提供vector对象容纳的元素数量而不用略去初始值
// 此时库会创建一个值初始化元素初值，赋给容器中所有元素
vector<int> ivec(10);       //10个元素，每个都初始化为0
vector<string> svec(10);    //10个元素，每个都是空string对象
// 如果只提供元素数量而没有设定初始值，只能使用直接初始化
vector<int> vi =10;         //错误，必须使用直接初始化形式指定向量大小
```

列表初始值 元素数量

```
//通过花括号 圆括号区分
vector<int> v1(10);          //v1有10个元素，每个值都是0
vector<int> v2{10};          //v2有1个元素，值为10
vector<int> v3(10,1);        //v3有10个元素，每个值为1
vector<int> v4{10,1};        //v4有2个元素，值分别为10和1
//如果初始化使用了花括号的形式，但提供的值不能用来列表初始化，
//就要考虑用这样的值构造vector对象
vector<string> v5{"hi"};     //列表初始化，v5有一个元素
vector<string> v6("hi");     //错误，不能使用字符串字面值构建vector对象
vector<string> v7{10};       //v7有10个默认初始化元素
vector<string> v8{10,"hi"}; //v8有10个值为"hi"的元素
```

练习

```
//下列的vector对象各包含多少个元素？这些元素的值分别是多少？

vector<int> v1;          // size:0,  no values.
vector<int> v2(10);      // size:10, value:0
vector<int> v3(10, 42); // size:10, value:42
vector<int> v4{ 10 };      // size:1,  value:10
vector<int> v5{ 10, 42 }; // size:2,  value:10, 42
vector<string> v6{ 10 };  // size:10, value:""
vector<string> v7{ 10, "hi" };  // size:10, value:"hi"
```

向VECTOR对象中添加元素

v.push_back() 在尾部增加元素。

```
vector<int> v2;
for(int i=0;i!=100;i++)
    v2.push_back(i);
//循环结束后v2有100个元素，0-99
```

```
//从标准输入中读取单词，将其作为vector对象的元素存储
string word;
vector<string> text;
while(cin>>word){
    text.push_back(word); //把word添加到text后面
}
```

- vector 对象能高速增长
- 范围for语句体不应改变其所遍历序列的大小

练习

```
//编写一段程序，用cin读入一组整数并把它们存入一个vector对象。
#include <iostream>
#include <string>
#include <cctype>
#include <vector>

using std::cin;using std::cout;
using std::endl;using std::vector;

int main()
{
    vector<int> v;
    int i;
    while (cin >> i)
    {
        v.push_back(i);
    }
    return 0;
}
```

练习

```
//改写上题程序，不过这次读入的是字符串。
#include <iostream>
#include <string>
#include <cctype>
#include <vector>

using std::cin;using std::cout;using std::endl;
using std::vector;using std::string;

int main()
{
    vector<string> v;
    string i;
    while (cin >> i)
    {
        v.push_back(i);
    }
    return 0;
}
```

其他VECTOR操作

```
v.emtpy()          //如果v不含有任何元素，返回真；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中的元素拷贝替换v1中的元素
v1 = {a,b,c...}    //用列表中元素的拷贝替换v1中的元素
v1 == v2           //v1和v2相等当且仅当它们的元素数量相同且对应位置的元素值都相同
v1 != v2           //同上
<,<=,>, >=         //以字典顺序进行比较
```

```
//size_type需要指定由哪种类型定义
vector<int>::size_type //正确
vector::size_type      //错误
```

```cpp
//score.cpp
//统计分数
#include <string>
using std::string;
#include <vector>
using std::vector;
#include <iostream>
using std::cin; using std::cout; using std::endl;
int main()
{
    // hold the grades we read from the standard input
    vector<unsigned> grades;

    // 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;
```

```
    // equivalent program using iterators instead of subscripts
    vector<unsigned> alt_scores(11, 0);  // 11 buckets, all initially 0
    // for each grade in the input
    for (auto it = grades.begin(); it != grades.end(); ++it) {
        unsigned i = *it;
        // increment the counter for the current cluster
        ++(*(alt_scores.begin() + i/10));
    }

    cout << "alt_scores.size = " << alt_scores.size() << endl;
    for (auto it = alt_scores.begin(); it != alt_scores.end(); ++it)
        cout << *it << " " ;
    cout << endl;

}
```

```
/*
10 20 30 30 90 90 100
grades.size = 7
10 20 30 30 90 90 100
scores.size = 11
0 1 1 2 0 0 0 0 0 2 1
alt_scores.size = 11
0 1 1 2 0 0 0 0 0 2 1
*/
```

不能用下标形式添加元素

- vector对象（以及string对象）的下标运算符，只能对确知已存在的元素执行下标操作，不能用于添加元素。

练习

```
//读入一组整数并把他们存入一个vector对象，将每对相邻整数的和输出出来。
//改写你的程序，这次要求先输出第一个和最后一个元素的和，
//接着输出第二个和倒数第二个元素的和，以此类推。
//q3_20.cpp
#include <iostream>
#include <string>
#include <cctype>
#include <vector>
using std::cin;using std::cout;
using std::endl;using std::vector;using std::string;

int main()
{
    vector<int> ivec;
    int i;
    while (cin >> i)
    {
        ivec.push_back(i);
    }
```

```
    for (int i = 0; i < ivec.size() - 1; ++i)
    {
        cout << ivec[i] + ivec[i + 1] << endl;
    }

    cout << "---------------------------------" << endl;

    int m = 0;
    int n = ivec.size() - 1;
    while (m < n)
    {
        cout << ivec[m] + ivec[n] << endl;
        ++m;
        --n;
    }
    return 0;
}
```

迭代器介绍

- 所有标准库容器都可以使用迭代器。
- 类似于指针类型，迭代器也提供了对对象的间接访问。

使用迭代器

```
vector<int>::iterator iter
auto b = v.begin();返回指向第一个元素的迭代器。
auto e = v.end();返回指向最后一个元素的下一个的迭代器。
如果容器为空， begin()和 end()返回的是同一个迭代器，都是尾后迭代器。
使用解引用符*访问迭代器指向的元素。
容器：可以包含其他对象；但所有的对象必须类型相同。
迭代器（iterator）：每种标准容器都有自己的迭代器。C++倾向于用迭代器而不是下标遍历元素。
```

标准容器迭代器的运算符:

```
*iter             返回迭代器iter所指向的元素的引用
iter->mem         等价于(*iter).mem
++iter            令iter指示容器中的下一个元素
--iter            令iter指示容器中的上一个元素
iter1 == iter2    判断两个迭代器是否相等
```

```
//字母改大写
string s("some string");
if(s.begin()!=s.end()){
    auto it = s.begin();
    *it = toupper(*it);
}
// Some string
```

将迭代器从一个元素移动到另一个元素

- 使用++运算符

```
//首字母改大写
for(auto it=s.begin();it!=s.end()&&!isspace(*it);++it)
    *it = toupper(*it);
```

- 使用迭代器和!=
    - 所有的标准库迭代器都定义了==和!=

迭代器类型

- 一般来说无需知道迭代器精确类型

```
vector<int>::iterator it;         //it 能读写元素
string::iterator it2;             //it2能读写字符

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

//const_iterator 和常量指针差不多，iterator的对象可读可写
//如果vector或string是常量，只能用const_iterator
```

BEGIN END 运算符

- 返回的具体类型由对象是否是常量决定是否是const_iterator还是iterator
- 为了便于得到const_iterator 返回值，c++11引入cbegin cend

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

解引用和成员访问操作

```
(*it).empty(); //解引用it，调用结果对象的empty成员
*it.empty();   //错误，试图访问it的empty成员，it是个迭代器，没有empty成员
//箭头运算符 -> 把解引用和成员访问两个操作结合在一起。
//it->mem 和 (*it).mem意义相同
//循环遍历text，直到遇到空字符串为止
for(auto it=text.cbegin();
    it!=text.cend()&& !it->empty();++it)
    cout<<*it<<endl;
```

- 谨记：但凡是使用了迭代器的循环体，都不要向迭代器所属的容器添加元素。
- 会使迭代器失效

练习

```cpp
//q3_23.cpp
//编写一段程序，创建一个含有10个整数的vector对象，
//然后使用迭代器将所有元素的值都变成原来的两倍。
//输出vector对象的内容，检验程序是否正确。
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> v(10, 1);
    for (auto it=v.begin(); it!=v.end(); it++){
        *it *= 2;
    }
    for (auto one : v){
        cout << one <<endl;
    }
    return 0;
}
```

迭代器运算

vector和string迭代器支持的运算：

iter + n	迭代器加上一个整数值仍得到一个迭代器， 迭代器指示的新位置和原来相比向前移动了若干个元素。 结果迭代器或者指示容器内的一个元素， 或者指示容器尾元素的下一位置。
iter - n	迭代器减去一个证书仍得到一个迭代器， 迭代器指示的新位置比原来向后移动了若干个元素。 结果迭代器或者指向容器内的一个元素， 或者指示容器尾元素的下一位置。
iter1 += n	迭代器加法的复合赋值语句，将iter1加n的结果赋给iter1
iter1 -= n	迭代器减法的复合赋值语句，将iter2减n的加过赋给iter1
iter1 - iter2	两个迭代器相减的结果是它们之间的距离，也就是说， 将运算符右侧的迭代器向前移动差值个元素后得到左侧的迭代器。 参与运算的两个迭代器必须指向的是同一个容器中的元素 或者尾元素的下一位置。
>、>=、<、<=	迭代器的关系运算符，如果某迭代器
difference_type	保证足够大以存储任何两个迭代器对象间的距离，可正可负。

使用迭代器运算 二分搜索

```
//biSearch.cpp
#include<iostream>
#include<string>
using std::string;using std::iterator;
using std::cout;using std::endl;
int main(){
    string text="abcdefghi";
    cout<<text<<endl;
    auto beg = text.begin(),end=text.end();
    auto mid = beg+(end-beg)/2;
    char sought = 'c';
    while(mid!=end&&*mid!=sought){
        if(sought<*mid)   end=mid;
        else   beg=mid+1;
        mid=beg+(end-beg)/2;
    }
    if(*mid==sought){
        cout<<"found:"<<*mid<<"@"<<mid-text.begin()<<endl;
    }
    return 0;
}
```

```
//abcdefghi
//found:c@2
```

使用迭代器运算 二分搜索

```
//biSearch2.cpp
#include<iostream>
#include<vector>
#include<string>
using std::string;using std::iterator;
using std::vector;
using std::cout;using std::endl;
int main(){
    vector<string> text{"a","ab","abc","abcd","abcdefghi"};
    for(auto s:text)
        cout<<s<<endl;
    auto beg = text.begin(),end=text.end();
    auto mid = beg+(end-beg)/2;
    string sought = "abc";
```

```
    while(mid!=end&&*mid!=sought){
        if(sought<*mid)         end=mid;
        else                beg=mid+1;
        mid=beg+(end-beg)/2;
    }
    if(*mid==sought){
        cout<<"found:"<<*mid<<"@"<<mid-text.begin()<<endl;
    }
    return 0;
}
//a
//ab
//abc
//abcd
//abcdefghi
//found:abc@2
```

练习

```cpp
//划分分数段的程序是使用下标运算符实现的，请利用迭代器改写该程序实现完全相同的功能。
#include <vector>
#include <iostream>
using std::vector; using std::cout; using std::cin; using std::endl;
int main()
{
    vector<unsigned> scores(11, 0);
    unsigned grade;
    while (cin >> grade)
    {
        if (grade <= 100)
            ++*(scores.begin() + grade / 10);
    }
    for (auto s : scores)
        cout << s << " ";
    cout << endl;
    return 0;
}
```

练习

```
//在二分搜索程序中，为什么用的是 mid = beg + (end - beg) / 2,
//而非 mid = (beg + end) / 2 ; ?
```

因为两个迭代器相互之间支持的运算只有 `-` ，而没有 `+` 。
但是迭代器和迭代器差值（整数值）之间支持 `+`。

数组

- 存放相同对象的容器，数组大小固定
- 如果不清楚元素确切个数，请使用vector

定义和初始化内置数组

- 形如 a[d],a是数组名，d是数组维度，是常量表达式。
- 长度必须是constexpr表达式，或者不写，让编译器自己推断。
- 数组不允许直接赋值给另一个数组。

```
unsigned cnt=42;//不是常量表达式
constexpr unsigned sz=3;//常量表达式
int *parr[sz];//ok
string bad[cnt];//错误，不是常量表达式
//显式初始化
int a[]={0,1,3};
int b[sz]={0,2,3};
```

字符数组特殊性

```
char a3[]="CPP";//自动添加\0空字符
const char a4[6]="Daniel";//错误，空间不够大，\0放不下
```

不允许拷贝和赋值

```
int a[]={0,2,2};
int a2[]=a;//错误 不允许
a2=a;//错误 不能直接赋值
```

理解复杂的数组声明

```
int *ptrs[10];//数组，存10个int*指针
int &p1[10] = ...;//错误 不存在引用数组
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
//从数组名开始，由内向外阅读
```

访问数组元素

- 数组下标的类型：`size_t` 。cstddef 头文件
- 字符数组的特殊性：结尾处有一个空字符，如
    - char a[] = "hello";
- 用数组初始化 `vector`：
    - int a[] = {1,2,3,4,5};
    - vector v(begin(a), end(a));

数组和指针

- 使用数组时，编译器一般会把它转换成指针。
- **指针访问数组**：在表达式中使用数组名时，名字会自动转换成指向数组的第一个元素的指针。
- 标准库函数 begin end。begin(ia) end(ia)
    - int ia[]={1,2,3};
    - auto n = end(ia) - begin(ia); //n为元素数量
- 内置下标运算符所用的索引不是无符号数

C风格字符串

- 从C继承来的字符串。
- 用空字符结束（\0）。
- 对大多数应用来说，使用标准库 string比使用C风格字符串更安全、更高效。
- 获取 string 中的 cstring：
    - const char *str = s.c_str();

C标准库String函数，定义在<cstring> 中：

函数	介绍
strlen(p)	返回p的长度，空字符不计算在内
strcmp(p1, p2)	比较p1和p2的相等性。如果p1==p2，返回0；如果p1>p2，返回一个正值；如果p1<p2，返回一个负值。
strcat(p1, p2)	将p2附加到p1之后，返回p1
strcpy(p1, p2)	将p2拷贝给p1，返回p1

尽量使用vector和迭代器，少用数组

多维数组

- 严格来说C++没有多维数组
    - 多维数组其实是数组的数组

```
int ia[3][4];//3行4列数组
int arr[10][20][30] = {0};//所有元素初始化为0
```

多维数组的初始化

```
int ia[3][4] = {
    {0,1,2,3},
    {4,5,6,7},
    {8,9,10,11}
};
int ia[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
int ia[3][4]={{0},{4},{8}};//初始化每行首个元素
int ix[3][4]={0,3,6,9};//显示初始化第1行，其它初始化为0
```

多维数组的下标引用

- 表达式的下标运算符和数组维度一样多，结果是给定类型元素
- 表达式的下标运算符数量小于数组维度，结果是给定索引处的内层数组

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

多维数组的下标引用

```
//两层嵌套for循环处理多维数组元素
constexpr size_t rowCnt=3,colCnt=4;
int ia[rowCnt][colCnt];//12个未初始化元素
//对于每一行
for(size_t i=0;i!=rowCnt;++i){
    //对于行内每一列
    for(size_t j=0;j!=colCnt;++j){
        //将元素位置索引作为值
        ia[i][j]=i*colCnt+j;
    }
}
```

使用范围FOR语句处理多维数组

```
//c++11新标准
size_t cnt=0;
for(auto &row:ia){
    for(auto &col:row){
        col=cnt;
        ++cnt;
    }
}

//能编译通过
for(const auto &row:ia){
    for(auto col:row){
        cout<<col<<endl;
    }
}
```

使用范围FOR语句处理多维数组

```
//不能编译通过
for(auto row:ia){
    for(auto col:row){
        //...
    }
}
//row 的类型是int*，内层循环将不合法
```

- 使用范围for语句时，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。

指针和多维数组

- 程序使用多维数组名字，自动转为指向数组首元素的指针

```
//多维数组实际是数组的数组
int ia[3][4];
int (*p)[4]=ia; //p指向含有4个整数的数组
p=&ia[2];       //p指向ia的尾元素
//圆括号不可少
int *ip[4];     //整型指针的数组
```

指针和多维数组

- 使用auto或decltype，能尽可能避免在数组前加指针类型

```
//输出ia中每个元素的值
//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;
}
```

指针和多维数组

- 使用标准库函数begin end更简洁

```
//p指向ia的第一个数组
for(auto p=begin(ia);p!=end(ia);++p){
    //q指向内层数组首元素
    for(auto q=begin(*p);q!=end(*p);++q){
        cout<<*q<<' ';//输出q所指的整数值
    }
    cout<<endl;
}
//使用auto关键字 不再烦心类型是什么
```

类型别名简化多维数组的指针

- 使用类型别名，让读写一个指向多维数组的指针变得简单一点

```
using int_array = int[4];//c++11新标准 类型别名的声明
tpyedef int int_array[4];//等价的typedef声明
//输出ia中每个元素的值 每个内层数组各占一行
for(int_array *p=ia;p!=ia+3;++p){
    for(int *q=*p;q!=*p+4;++q){
        cout<<*q<<' ';
    }
    cout<<endl;
}
//用int_array定义外层循环控制变量，让程序简明
```

练习

```
//分别使用for语句、下标运算符、指针，输出二维数组元素。直接写出数据类型。
#include <iostream>
using std::cout; using std::endl;

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

```cpp
    // range for
    for (const int(&row)[4] : arr)
        for (int col : row)
            cout << col << " ";
    cout << endl;

    // for loop
    for (size_t i = 0; i != 3; ++i)
        for (size_t j = 0; j != 4; ++j) cout << arr[i][j] << " ";
    cout << endl;

    // using pointers.
    for (int(*row)[4] = arr; row != arr + 3; ++row)
        for (int *col = *row; col != *row + 4; ++col) cout << *col << " ";
    cout << endl;

    return 0;
}
```

实践课

- 本场景将使用一台配置了Aliyun Linux 2的ECS实例（云服务器）
    - 使用Vim编辑C++代码
    - 使用g++编译运行这段代码
    - 编辑一个 **README.md** 文档，键入本次实验心得。
    - 使用git进行版本控制 可使用之前的代码仓库

```
- 云服务器（Elastic Compute Service，简称ECS）
- Aliyun Linux 2是阿里云推出的 Linux 发行版
- Vim是从vi发展出来的一个文本编辑器。
- g++ 是c++编译器
```

```
//3道编程题
1.使用迭代器编写划分分数段程序
2.使用迭代器编写二分查找算法
3.分别使用for语句、下标运算符、指针，输出二维数组元素。直接写出数据类型。
```

从课程主页cpp.njuer.org 打开实验课 cloud shell界面
使用Vim编辑c++代码和markdown文档，使用git进行版本控制
●单击屏幕右侧创建资源
●资源创建完毕后， 使用命令安装git工具和g++工具
//本地虚拟机这些工具已经装好，不必运行这两行
●yum install -y git
●yum install -y gcc-c++
●使用git工具进行版本控制
●mkdir test 建立文件夹test 或 git clone你之前的网络git仓库test(或其它名字)
●cd test 进入文件夹test
●git init 表示文件夹版本库初始化 或 对于clone的仓库该步骤不操作
(clone的仓库，可移动旧文件到目录weekN: mkdir -p weekN ; mv 文件名 weekN;)

```
vim test1.cpp
g++ ./test1.cpp 编译
./a.out 执行程序

vim test2.cpp
g++ ./test2.cpp 编译
./a.out 执行程序

vim test3.cpp
g++ ./test3.cpp 编译
./a.out 执行程序
```

- git add . 加入当前文件夹下所有文件到暂存区
- git config --global user.email "you@example.com"
- git config --global user.name "Your Name"
- git commit –m "test1" 表示提交到本地,备注test1
- vim readme.md 键入新内容（实验感想),按ESC 再按：wq退出
- git add .
- git commit –m "test2" 表示提交到本地,备注test2
- git log 可看git记录
- 键入命令并截图，并提交到群作业。

```
cat test* readme.md
git log
```

附加题：

1)使用tmux工具辅助编程
2)并在gitee.com上注册用户名，并建立test项目。或使用之前建立的仓库。
把阿里云平台的本次作业test目录git push到你的git仓库
地址一般为： **https://gitee.com/你的用户名/test.git**

```
//写完作业后
确认自己当前目录里就是作业目录test。不是的话可用cd ..和cd 文件夹改变当前目录。
git remote add origin https://gitee.com/你的用户名/test.git //git clone的仓库不需要此步骤.
//你的用户名是全英文
git push -u origin "master"
输入用户名密码 回车
浏览器打开 https://gitee.com/你的用户名/test 查看作业保存情况。
```

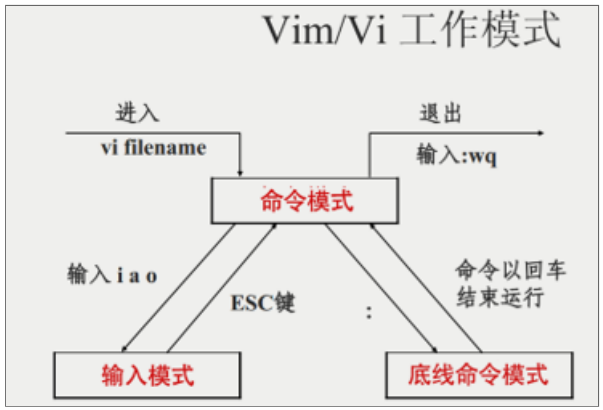
提交

- 截图或复制文字，提交到群作业。
- 填写网页实验报告栏，发布保存。本次报告不需要分享提交
- 填写问卷调查 **https://rnk6jc.aliwork.com/o/cppinfo**

关于使用TMUX

sudo yum install -y tmux
cd ~ && wget **https://cpp.njuer.org/tmux** && mv tmux .tmux.conf
tmux 进入会话 .
前缀按键prefix= ctrl+a,
prefix+c创建新面板,
prefix+:分屏,
prefix+k选上面,prefix+j选下面,
prefix+1选择第一,prefix+n选择第n,
prefix+d脱离会话
tmux attach-session -t 0 回到会话0

VIM 共分为三种模式



- 命令模式
  - 刚启动 `vim`，便进入了命令模式。其它模式下按`ESC`，可切换回命令模式
    - `i` 切换到输入模式，以输入字符。
    - `x` 删除当前光标所在处的字符。
    - `:` 切换到底线命令模式，可输入命令。
- 输入模式
  - 命令模式下按下`i`就进入了输入模式。
    - `ESC`，退出输入模式，切换到命令模式
- 底线命令模式
  - 命令模式下按下`:`（英文冒号）就进入了底线命令模式。
    - `wq` 保存退出

VIM 常用按键说明

除了 i, Esc, :wq 之外，其实 vim 还有非常多的按键可以使用。命令模式下：

- 光标移动
  - j下 k上 h左 l右
  - w前进一个词 b后退一个词
  - Ctrl+d 向下半屏 ctrl+u 向上半屏
  - G 移动到最后一行 gg 第一行 ngg 第n行
- 复制粘贴
  - dd 删一行 ndd 删n行
  - yy 复制一行 nyy复制n行
  - p将复制的数据粘贴在下一行 P粘贴到上一行
  - u恢复到前一个动作 ctrl+r重做上一个动作
- 搜索替换
  - /word 向下找word ? word 向上找
  - n重复搜索 N反向搜索
  - :1,$s/word1/word2/g从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2

VIM 常用按键说明

底线命令模式下：

- :set nu　显示行号
- :set nonu 取消行号
- :set paste　粘贴代码不乱序

【注：把caps lock按键映射为ctrl，能提高编辑效率。】

MARKDOWN 文档语法

```
# 一级标题
## 二级标题
*斜体* **粗体**
- 列表项
  - 子列表项
> 引用
[超链接](http://asdf.com)
![图片名](http://asdf.com/a.jpg)

|表格标题1|表格标题2|
|-|-|
|内容1|内容2|
```

谢谢