@阿西拜-南昌

标准库特殊设施

C++ Primer第五版

第17章

tuple类型

将一些数据组合成单一对象：“快速而随意”的数据结构

```
tuple<size_t, size_t, size_t> threeD; //三个成员都设置为0

//为每个成员提供初始值
tuple<string, vector<double>, int, list<int>>
            someVal(“constants”, {3.14,2,718},42,{0,1,2,3,4,5});

//tuple的这个构造函数是explicit的
tuple<size_t,size_t,size_t> threeD = {1,2,3}; //错误
tuple<size_t,size_t,size_t> threeD{1,2,3}; //正确

//表示输掉交易记录的tuple,包含：ISBN、数量和每册书的价格
auto item = make_tuple("0-999-78345-X",3,20.00);
```

访问tuple的成员

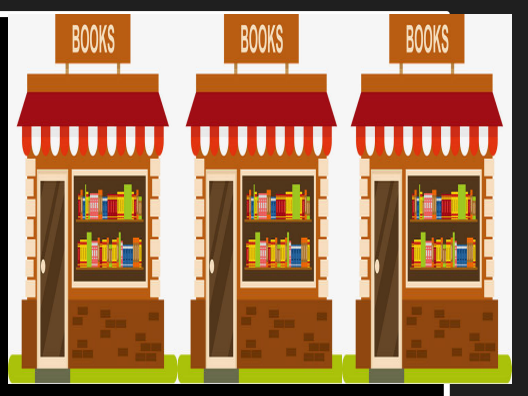
```
auto book = get<0>(item);           //返回item的第一个成员
auto cnt = get<1>(item);            //返回item的第二个成员
auto price = get<2>(item)/cnt;      //返回item的最后一个成员
get<2>(item) *= 0.8;                //打折20%

typedef decltype(item) trans; //trans是item的类型
//返回trans类型对象中成员的数量
size_t sz = tuple_size<trans>::value; //返回3
//cnt的类型与item中第二个成员相同
tuple_element<1,trans>::type cnt = get<1>(item); //cnt是一个int
```

关系和相等运算符

```
tuple<string, string> duo("1","2");
tuple<size_t,size_t> twoD(1,2);
bool b = (duo == twoD);     //错误：不能比较size_t和string
tuple<size_t, size_t, size_t> threeD(1,2,3);
b = (twoD<threeD);          //错误：成员数量不同
tuple<size_t,size_t> origin(0,0);
b = (origin<twoD);          //正确：b为true
```

使用tuple返回多个值




```
//files中的每个元素保存一家书店的销售记录
vector<vector<Sales_data>> files;
//一家书店的索引和两个指向书店vector中元素的迭代器
typedef tuple<vector<Sales_data>::size_type,
              vector<Sales_data>::const_iterator,
              vecotr<Sales_data>::cosnt_iterator> matches;
//files保存每家书店的销售记录
//findBook返回一个vector，每家销售了给定书籍的书店在其中都有一项
vector<matches>
findBook(const vector<vector<Sales_data>> &files,const string &book)
{
    vector<matches> ret; //初始化为空vector
    //对每家书店，查找与给定书籍匹配的记录范围（如果存在的话）
    for(auto it = files.cbegin(); it != files.cend(); ++it){
        //查找具有相同ISBN的Sales_data范围
        auto found = equal_range(it->cbegin(),it->cend(),book,compareIsbn);
        if(found.first != found.second) //此书店销售了给定书籍
        //记住此书店的索引及匹配的范围
            ret.push_back(make_tuple(it - files.cbegin(),
                                     found.first,found.second));
    }
    return ret; //如果未找到匹配记录的话，ret为空
}
```

Sales_data没有<运算符

使用函数返回的tuple

```
void reportResults(istream &in, ostream &os,
                   const vector<vector<Sales_data>> &files)
{
    string s; //要查找的书
    while(in >> s){
        auto trans = findBook(files,s); //销售了这本书的书店
        if(trans.empty()){
            cout<<s<<" not found in any stores"<<endl;
            continue; //获得下一本要查找的书
        }
        for(const auto &store:trans) //对每家销售了该书籍的书店
            //get<n>返回store中tuple的指定的成员
            os<<"store"<<get<0>(store)<<" sales: "
                <<accumulate(get<1>(store),get<2>(store),Sales_data(s))
                <<endl;
    }
}
```

tuple支持的操作

`tuple<T1, T2, ..., Tn> t;`	t 是一个 tuple，成员数为 n，第 i 个成员的类型为 Ti。所有成员都进行值初始化
`tuple<T1, T2, ..., Tn> t(v1, v2, ..., vn);`	t 是一个 tuple，成员类型为 T1…Tn，每个成员用对应的初始值 v_i 进行初始化。此构造函数是 explicit 的
`make_tuple(v1, v2, ..., vn)`	返回一个用给定初始值初始化的 tuple。tuple 的类型从初始值的类型推断
`t1 == t2` `t1 != t2`	当两个 tuple 具有相同数量的成员且成员对应相等时，两个 tuple 相等。这两个操作使用成员的==运算符来完成。一旦发现某对成员不等，接下来的成员就不用比较了
`t1 relop t2`	tuple 的关系运算使用字典序 两个 tuple 必须具有相同数量的成员。使用<运算符比较 t1 的成员和 t2 中的对应成员
`get<i>(t)`	返回 t 的第 i 个数据成员的引用：如果 t 是一个左值，结果是一个左值引用；否则，结果是一个右值引用。tuple 的所有成员都是 public 的
`tuple_size<tupleType>::value`	一个类模板，可以通过一个 tuple 类型来初始化。它有一个名为 value 的 public constexpr static 数据成员，类型为 size_t，表示给定 tuple 类型中成员的数量
`tuple_element<i, tupleType>::type`	一个类模板，可以通过一个整型常量和一个 tuple 类型来初始化。它有一个名为 type 的 public 成员，表示给定 tuple 类型中指定成员的类型

bitset类型

标准库定义了bitset类，使得位运算更容易

```
bitset<32> bitvec(1U); //32位；低位为1，其他位为0
```

初始化bitset的方法

bitset<n> b;	b 有 n 位；每一位均为 0。此构造函数是一个 constexpr
bitset<n> b(u);	b 是 unsigned long long 值 u 的低 n 位的拷贝。如果 n 大于 unsigned long long 的大小，则 b 中超出 unsigned long long 的高位被置为 0。此构造函数是一个 constexpr
bitset<n> b(s, pos, m, zero, one);	b 是 string s 从位置 pos 开始 m 个字符的拷贝。s 只能包含字符 zero 或 one；如果 s 包含任何其他字符，构造函数会抛出 invalid_argument 异常。字符在 b 中分别保存为 zero 和 one。pos 默认为 0，m 默认为 string::npos，zero 默认为'0'，one 默认为'1'
bitset<n> b(cp, pos, m, zero, one);	与上一个构造函数相同，但从 cp 指向的字符数组中拷贝字符。如果未提供 m，则 cp 必须指向一个 C 风格字符串。如果提供了 m，则从 cp 开始必须至少有 m 个 zero 或 one 字符

接受一个 string 或一个字符指针的构造函数是 explicit

使用一个整形值来初始化bitset时，将转换为unsigned long long，并被当做位模式来处理

unsigned值初始化bitset

```
//bitvec1比初始值小；初始值中的高位被丢弃
bitset<13> bitvec1(0xbeef); //二进制位序列为1111011101111
//bitvec2比初始值大；它的高位被置为0
bitset<20> bitvec2(0xbeef); //二进制位序列00001011111011101111
//在64位机器中，long long OULL是一个64个0比特，因此~0ULL是64个1
bitset<128> bitvect3(~0ULL); //0~63位为1；63~127位为0
```

从一个string初始化bitset


```
bitset<32> bitvec4("1100");          //2,3两位为1，剩余两位为0

string str("11111110000000011001101");
bitset<32> bitvec5(str,5,4);         //从str[5]开始的四个二进制位，1100
bitset<32> bitvec6(str,str.size()-4); //使用最后四个字符
```

bitset操作

```
bitset<32> bitvec(1U);               //32位；低位为1，剩余位为0
bool is_set = bitvec.any();          //true,因为有1位置位
bool is_not_set = bitvec.none();     //false，因为有1位置位了
bool all_set = bitvec.all();         //false,因为只有1位置位
size_t onBits = bitvec.count();      //返回1
size_t sz = bitvec.size();           //返回32
bitvec.flip();                       //翻转bitvec中的所有位
bitvec.reset();                      //将所有位复位
bitvec.set();                        //将所有位置位

bitvec[0] = 0;                       //将第一位复位
bitvec[31] = bitvec[0];              //将最后一位设置为与第一位一样
bitvec[0].flip();                    //翻转第一位
~bitvec[0];                          //等价操作，也是翻转第一位
bool b = bitvec[0];                  //将bitvec[0]的值转换为bool类型
```

下标运算符对const属性进行了重载。

- const的版本返回true或false
- 非const版本，允许指定位的值

提取bitset值

```
unsinged long ulong = bitvec3.to_ulong();
cout<<"ulong = " << ulong <<endl;
//bitset中的值的大小，不能大于转换的类型
//否则会抛出overflow_error异常
```

bitset的IO运算符

```
bitset<16> bits;
cin >> bits; //从cin读取最多16个0或1
cout << "bits: "<< bits <<endl; //打印刚刚读取的内容
```

使用bitset

```
bool status;
//使用位运算符的版本
unsigned long quizA = 0;    //此值被当做位集合使用
quizA |=  1UL  <<  27;      //指出第27个学生通过了测试
status = quitzA & (1UL <<27); //检查第27个学生是否通过了测试
quizA &= ~(1UL  <<  27);    //第27个学生未通过测试

//使用标准库类bitset完成等价工资
bitset<30> quizB;           //每个学生分配一位，所有位都被初始化为0
quizB.set(27);              //指出第27个学生通过了测试
status =  quizB[27];        //检查第27个学生是否通过了测试
quizB.reset(27);            //第27个学生未通过测试
```

正则表达式（regular expression）

描述字符序列的方法

```
//查找不在字符c之后的字符串ei
string pattern("[^c]ei");
//我们需要包含pattern的整个单词
pattern = "[[:alpha:]]*" + pattern + "[[:alpha:]]*";
regex r(pattern);  //构造一个用于查找模式的regex
smatch results;    //定义一个对象保存搜索结果
//定义一个string保存与模式匹配和不匹配的文本
string test_str="receipt freind theif receive";
//用r在test_str中查找与pattern匹配的子串
if(regex_search(test_str,results,r))//如果有匹配子串
      cout<<results.str()<<endl; //打印匹配的单词
```

- regex：表示有一个正则表达式的类
- smatch：容器类，保存在string中搜索的结果
- regex_search：寻找第一个与正则表达式匹配的子序列

指定regex对象的选项：指定一些标志来影响regex如何操作

```
//识别扩展名
//一个或多个字母或数字字符后接一个'.'再接"cpp"或"cxx"或"cc"
regex r("[[:alnum:]]+\\.(cpp|cxx|cc)$",regex::icase);
smatch results;
string filename;
while(cin>>filename)
      if(regex_search(filename,results,r))
            cout<<results.str()<<endl;//打印匹配结构
```

- 在匹配中忽略大小写

正则表达式的语法是否正确是在运行时解析的

```
try{
      //错误：alnum漏掉了右括号，构造函数会抛出异常
      regex r("[[:alnum:]+\\.(cpp|cxx|cc)$",regex::icase);
}catch(regex_error e)
{ cout<<e.what()<<"\ncode:"<<e.code()<<endl; }
```

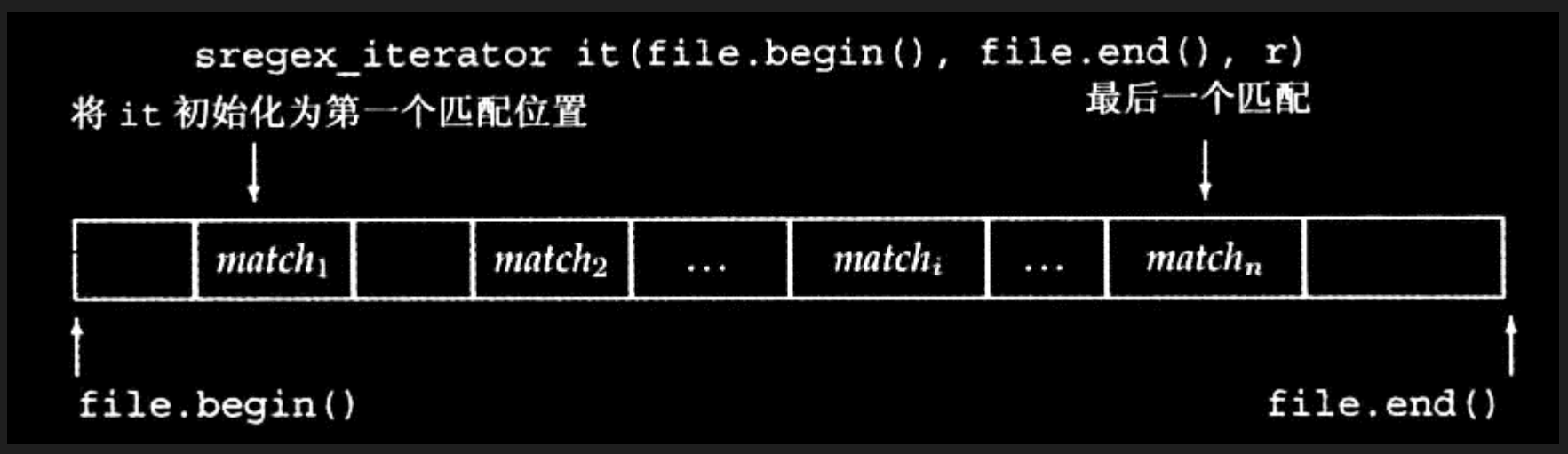
```
C:\WINDOWS\system32\cmd.exe
regular expression error
code:4
```

正则表达式类和输入序列类型

```
regex r("[[:alnum:]]+\\.(cpp|cxx|cc)$",regex::icase);
smatch results;    //将匹配string输入序列，而不是char*
//cmatch results; //将匹配字符数组输入序列
if( regex_search("myfile.cc",results,r)) //错误：输入为char*
      cout<<results.str()<<endl;
```

使用sregex_iterator来获得所有匹配


```
string file("Hello cei world freind meizhu");
//查找前一个字符不是c的字符串ei
string pattern("[^c]ei");
pattern = "[[:alpha:]]*" + pattern + "[[:alpha:]]*";
regex r(pattern,regex::icase);
//它将反复调用regex_search来虚招文件中的所有匹配
for(sregex_iterator it(file.begin(),file.end(), r),end_it; it != end_it; ++it)
      cout<< it->str() <<endl; //匹配的单词
```



显示匹配单词出现的上下文

```
//循环头与之前一样
for(sregex_iterator it(file.begin(), file.end(), r), end_it;it != end_it; ++it) {
            auto pos = it->prefix().length();       //前缀的大小
            pos = pos > 40 ? pos - 40 : 0;          //我们想要最多40个字符

            cout << it->prefix().str().substr(pos) //浅醉的最后一部分
            << "\n\t\t>>>" << it->str() << " <<<\n" //匹配的单词
            << it->suffix().str().substr(0,40)      //后缀的第一部分
            << endl;
}
```

```
C:\WINDOWS\system32\cmd.exe
Hello cei world
                >>>freind <<<
meizhu hoho

                >>>meizhu <<<
hoho
```

使用子匹配操作

```
//两个子表达式：1、点之前表示文件名的部分，2、表示文件扩展名
regex r("([[:alnum:]]+)\\.(cpp|cxx|cc)$",regex::icase);
smatch results;
string filename;
while(cin>>filename)
        if(regex_search(filename,results,r))
                cout<<results.str(1)<<endl; //打印第一个子表达式
```

```
C:\WINDOWS\system32\c
aa. cpp
aa
```

子表达式用于数据验证

```
"(\\()?(\\d{3})(\\))?([-. ])?(\\d{3})([-. ])?(\\d{4})";
```

正则表达式中的模式通常包含一个或多个子表达式

1. (\\()?表示区号部分可选的左括号
2. (\\d{3})表示区号
3. (\\))?表示区号部分可选的右括号
4. ([-.])?表示区号部分可选的分隔符
5. (\\d{3})表示号码的下三位数字
6. ([-.])?表示可选的分隔符
7. (\\d{4})表示号码的最后四位数字

```
bool valid(const smatch& m)
{
        //如果区号前有一个左括号
        if(m[1].matched)
                //则区号后必须有一个右括号
                return m[3].matched &&
                (m[4].matched == 0 || m[4].str() == " ");
        else
                //否则，区号后不能有右括号
                //另两个组成部分间的分隔符必须匹配
                return !m[3].matched
                && m[4].str() == m[6].str();
}
```

```
string phone = "(\\()?(\\d{3})(\\))?([-. ])?(\\d{3})([-. ])?(\\d{4})";
regex r(phone); //regex对象，用于查找我们的模式
smatch m;
string s;
//从输入文件中读取每条记录
while(getline(cin,s)) {
        //对每个匹配的电话号码
        for(sregex_iterator it(s.begin(), s.end(), r), end_it;
                it != end_it; ++it)
                //检查号码的格式是否合法
                if(valid(*it))
                        cout<<"valid: "<< it->str() << endl;
                else
                        cout<<"not valid: "<< it->str() << endl;
}
```

使用regex_replace将找到的序列替换为另一个序列

```
int main()
{
	string phone = "(\\()?(\\d{3})(\\))?([-. ])?(\\d{3})([-. ])?(\\d{4})";

	regex r(phone); //regex对象，用于查找我们的模式

	string fmt = "$2.$5.$7"; //将号码格式改为ddd.ddd.dddd
	string fmt2 = "$2. $5. $7";
	string number = "(908) 555-1800";

	cout << regex_replace(number, r, fmt) << endl;
	cout << regex_replace(number, r, fmt2, format_no_copy) << endl;

	number = "(08) 555-1800";

	cout << regex_replace(number, r, fmt) << endl;
	cout << regex_replace(number, r, fmt2, format_no_copy) << endl;
}
```

```
C:\WINDOWS\system32\cmd.exe
908.555.1800
908. 555. 1800
(08) 555-1800
```

不输出输入序列中未匹配的部分

随机数

C++程序不应该使用库函数rand

```
#include <iostream>
#include <random>
using namespace std;

int main()
{
    default_random_engine e; //生成随机无符号数
    for(size_t i = 0; i<5; ++i)
        cout<<e()<<" ";

    cout<<endl;
```

C:\WINDOWS\system32\cmd.exe

```
3499211612 581869302 3890346734 3586334585 545404204
3499211612 581869302 3890346734 3586334585 545404204
```

```
    default_random_engine e2;
    for(size_t i = 0; i<5; ++i)
        cout<<e2()<<" ";
}
```

随机数引擎操作	
Engine e;	默认构造函数；使用该引擎类型默认的种子
Engine e(s);	使用整型值 s 作为种子
e.seed(s)	使用种子 s 重置引擎的状态
e.min()	此引擎可生成的最小值和最大值
e.max()	
Engine::result_type	此引擎生成的 unsigned 整型类型
e.discard(u)	将引擎推进 u 步；u 的类型为 unsigned long long

平均分布

```
//生成0到9之间（包含）均匀分布的随机数
uniform_int_distribution<unsigned> u(0,9);
default_random_engine e; //生成无符号随机整数
for(size_t i=0; i<10; ++i)
    //将u作为随机数源
    //每个调用返回执行范围内并服从均匀分布的值
    cout<<u(e)<<" ";
```

随机数发生器

引擎生成一个数值序列

```
//每次调用这个函数都会生成相同的100个数！
vector<unsigned> bad_randVec()
{
    default_random_engine e;
    uniform_int_distribution<unsigned> u(0,9);
    vector<unsigned> ret;
    for(size_t i=0;i<100;++i)
        ret.push_back(u(e));
    return ret;
}

int main()
{
    vector<unsigned> v1(bad_randVec());
    vector<unsigned> v2(bad_randVec());
    //将打印“equal”
    cout<<((v1 == v2) ? "equal":"not equal")<<endl;
}
```

```
static default_random_engine e;
static uniform_int_distribution<unsigned> u(0,9);
```

设置随机数发生器种子

```
default_random_engine e1;                    //使用默认种子
default_random_engine e2(3147442);           //使用给定的种子
default_random_engine e3;                    //使用默认种子
e3.seed(32767);                              //调用seed设置一个新种子
default_random_engine e4(32767);             //将种子设置为32767
for(size_t i=0; i !=100; ++i){
    if(e1() == e2())
        cout<<"unseeded match at literation:"<<i<<endl;
    if(e3()!= e4())
        cout<<"seeded differs at literation:"<<i<<endl;
}
```

以时间为种子：只适用于间隔为秒级或更长的应用

```
default_random_engine e(time(0));
```

生成随机实数

```
default_random_engine e; //生成无符号随机整数
//0到1（包含）的均匀分布
uniform_real_distribution<double> u(0,1);
for(size_t i = 0; i<10; ++i)
      cout<<u(e)<<" ";
```

使用分布的默认结果类型

```
//空<>表示我们希望使用默认结果类型
uniform_real_distribution<> u(0,1); //默认生成double值
```

生成非均匀分布的随机数

```
default_random_engine e;          //生成随机整数
normal_distribution<> n(4,1.5);   //均值4，标准差1.5
vector<unsigned> vals(9);         //9个元素均为0
for(size_t i = 0; i != 200; ++i){
      unsigned v = lround(n(e));   //舍入到最接近的整数
      if(v<vals.size())            //如果结果在范围内
            ++vals[v];             //统计每个数出现了多少次
}
for(size_t j=0; j != vals.size(); ++j)
      cout<<j<<": "<< string(vals[j],'*')<<endl;
```

Microsoft Visual Studio 调试控制台

```
0: ***
1: **********
2: ********************
3: ******************************************
4: ************************************************************
5: ****************************
6: ************************
7: *****
8: *
```

```
string resp;
default_random_engine e; //e应保持状态，所以必须在循环外定义
bernoulli_distribution b; //默认是50/50的机会
do{
      bool first = b(e); //如果未true，则程序先行
      cout<<(first?"We go first":"You get to go first")<<endl;
      //传递谁先行的指示，进行游戏
      cout<<((play(first))?"sorry, you lost":"congrats, you won")<<endl;
      cout<<"play again?Enter 'yes' or 'no'"<<endl;
}while(cin>>resp&&resp[0] == 'y');
```

IO库再探

通过操纵符改变格式状态

```
bool b;
cout << "default bool values: " << true << " " << false
     << "\nalpha bool values: " << boolalpha
     << true << " " << false << endl;

bool bool_val = false;
cout << boolalpha    // 设置cout内部状态
     << bool_val
     << noboolalpha; // 恢复默认状态
cout << endl;

const int ival = 15, jval = 1024;  // const
```

```
default bool values: 1 0
alpha bool values: true false
false
```

```
cout << "default: " << 20 << " " << 1024 << endl;
cout << "octal: " << oct << 20 << " " << 1024 << endl;
cout << "hex: " << hex << 20 << " " << 1024 << endl;
cout << "decimal: " << dec << 20 << " " << 1024 << endl;
```

```
default: 20 1024
octal: 24 2000
hex: 14 400
decimal: 20 1024
```

hex 十六进制　octal 八进制　decimal 十进制　对浮点数没有影响

showbase 打印整数并且显示进制

```
cout << showbase;   // show the base when printing integral values
cout << "default: " << 20 << " " << 1024 << endl;
cout << "in octal: " << oct  << 20 << " " << 1024 << endl;
cout << "in hex: " << hex  << 20 << " " << 1024 << endl;
cout << "in decimal: " << dec << 20 << " " << 1024 << endl;
cout << noshowbase;  //reset the state of the stream
```

取消

```
default: 20 1024
in octal: 024 02000
in hex: 0x14 0x400
in decimal: 20 1024
```

```
cout << uppercase << showbase << hex
   << "printed in hexadecimal: " << 20 << " " << 1024
   << nouppercase << noshowbase << dec << endl;
```

uppercase：把小写改为大写

```
printed in hexadecimal: 0X14 0X400
```

```
double pi = 3.14;
cout << pi << " " << hexfloat << pi
   << defaultfloat << " " << pi << endl;
```

```
3.14 0x1.91eb85p+1 3.14
```

指定打印精度

```
//cout.precision返回当前精度值
cout<<"Precision: "<< cout.precision()
     <<", Value: "<<sqrt(2.0)<<endl;
//cout.precision(12)将打印精度设置为12位数字
cout.precision(12);
cout<<"Precision: "<< cout.precision()
     <<", Value: "<<sqrt(2.0)<<endl;
//另一种设置精度的方法是使用setprecision操纵符
cout<<setprecision(3);
cout<<"Precision: "<< cout.precision()
     <<", Value: "<<sqrt(2.0)<<endl;
```

```
Precision: 6, Value: 1.41421
Precision: 12, Value: 1.41421356237
Precision: 3, Value: 1.41
```

指定浮点数计数法

```
cout << "default format: " << 100 * sqrt(2.0) << '\n'
     << "scientific: " << scientific << 100 * sqrt(2.0) << '\n'
     << "fixed decimal: " << fixed << 100 * sqrt(2.0) << '\n'
     << "hexadecimal: " << hexfloat << 100 * sqrt(2.0) << '\n'
     << "use defaults: " << defaultfloat << 100 * sqrt(2.0)
     << "\n\n";

cout << uppercase
     << "scientific: " << scientific << sqrt(2.0) << '\n'
     << "fixed decimal: " << fixed << sqrt(2.0) << '\n'
     << "hexadecimal: " << hexfloat << sqrt(2.0) << "\n\n"
     << nouppercase;
```

```
default format: 141.421
scientific: 1.414214e+002
fixed decimal: 141.421356
hexadecimal: 0x1.1ad7bcp+7
use defaults: 141.421

scientific: 1.414214E+000
fixed decimal: 1.414214
hexadecimal: 0x1.6a09E6p+0
```

输出补白

```
int i = -16;
double d = 3.14159;

// pad the first column to use a minimum of 12 positions in the output
cout << "i: " << setw(12) << i << "next col" << '\n'
    << "d: " << setw(12) << d << "next col" << '\n';

// pad the first column and left-justify all columns
cout << left
    << "i: " << setw(12) << i << "next col" << '\n'
    << "d: " << setw(12) << d << "next col" << '\n'
    << right;          // restore normal justification

// pad the first column and right-justify all columns
cout << right
    << "i: " << setw(12) << i << "next col" << '\n'
    << "d: " << setw(12) << d << "next col" << '\n';

// pad the first column but put the padding internal to the field
cout << internal
    << "i: " << setw(12) << i << "next col" << '\n'
    << "d: " << setw(12) << d << "next col" << '\n';

// pad the first column, using # as the pad character
cout << setfill('#')
    << "i: " << setw(12) << i << "next col" << '\n'
    << "d: " << setw(12) << d << "next col" << '\n'
    << setfill(' ');   // restore the normal pad character
```

```
i:          -16next col
d:      3.14159next col
i: -16          next col
d: 3.14159      next col
i:          -16next col
d:      3.14159next col
i: -            16next col
d:      3.14159next col
i: -##########16next col
d: #####3.14159next col
```

控制输入格式

```
char ch;
cin>>noskipws; //设置cin读取空白符
while (cin >> ch)
      cout << ch;
cout << endl;
cin>>skipws;  //将cin恢复到默认状态
```

```
a b c   d
a b c   d
```

使用未格式化IO操作get和put来读取和写入一个字符

@阿西拜-南昌

```
int ch;    // use an int, not a char to hold the return from get()

// loop to read and write all the data in the input
while ((ch = cin.get()) != EOF)
        cout.put(ch);
cout << endl;
```

```
a b c  d
a b c  d
```

将一个流当作一个无解释的字节序列来处理

单字节底层IO操作

is.get(ch)	从 istream is 读取下一个字节存入字符 ch 中。返回 is
os.put(ch)	将字符 ch 输出到 ostream os。返回 os
is.get()	将 is 的下一个字节作为 int 返回
is.putback(ch)	将字符 ch 放回 is。返回 is
is.unget()	将 is 向后移动一个字节。返回 is
is.peek()	将下一个字节作为 int 返回，但不从流中删除它

多字节底层IO操作

is.get(sink, size, delim)

从 is 中读取最多 size 个字节，并保存在字符数组中，字符数组的起始地址由 sink 给出。读取过程直至遇到字符 delim 或读取了 size 个字节或遇到文件尾时停止。如果遇到了 delim，则将其留在输入流中，不读取出来存入 sink

is.getline(sink, size, delim)

与接受三个参数的 get 版本类似，但会读取并丢弃 delim

is.read(sink, size)

读取最多 size 个字节，存入字符数组 sink 中。返回 is

is.gcount()

返回上一个未格式化读取操作从 is 读取的字节数

os.write(source, size)

将字符数组 source 中的 size 个字节写入 os。返回 os

is.ignore(size, delim)

读取并忽略最多 size 个字符，包括 delim。与其他未格式化函数不同，ignore 有默认参数：size 的默认值为 1，delim 的默认值为文件尾

sink是char数组，用来保存数据

流随机访问


```
// open for input and output and preposition file pointers to end-of-file
// file mode argument
   fstream inOut("data/copyOut",
                 fstream::ate | fstream::in | fstream::out);
   if (!inOut) {
      cerr << "Unable to open file!" << endl;
      return EXIT_FAILURE; // EXIT_FAILURE
       }

   // inOut is opened in ate mode, so it starts out positioned at the end
   auto end_mark = inOut.tellg();// remember original end-of-file position
   inOut.seekg(0, fstream::beg); // reposition to the start of the file
   size_t cnt = 0;              // accumulator for the byte count
   string line;                 // hold each line of input

   // while we haven't hit an error and are still reading the original data
   while (inOut && inOut.tellg() != end_mark
         && getline(inOut, line)) { // and can get another line of input
      cnt += line.size() + 1;       // add 1 to account for the newline
      auto mark = inOut.tellg();    // remember the read position
      inOut.seekp(0, fstream::end); // set the write marker to the end
      inOut << cnt;                 // write the accumulated length
      // print a separator if this is not the last line
      if (mark != end_mark) inOut << " ";
      inOut.seekg(mark);            // restore the read position
       }
   inOut.seekp(0, fstream::end);    // seek to the end
   inOut << "\n";                   // write a newline at end-of-file
```

```
abcd
efg
hi
j
5 9 12 14
```