随机数

C库使用rand()来生成随机数，该函数生成均匀分布的伪随机数。rand函数存在一些问题。C++引入一组协作的类：随机数引擎（random-number engine）和随机数分布类（random-number distribution）。这两个类，可以更加灵活自由地生成各类随机数。

随机数引擎和分布

随机数引擎（random-number engines）是一个函数对象类（function-object），其中定义了一个调用运算符，返回一个随机unsigned整数。我们通过调用一个随机数引擎来生成**原始**随机数：

```
default_random_engine e;     // 生成unsigned随机数字
for(size_tw i=0;i<10;++i){
    cout<<e()<<" ";// e()调用对象来生成一个随机数（unsigned）
}
```

原始随机数，一般来说不可以直接使用（值范围等不符号要求），这时候 搭配分布类型一起使用。

为得到指定范围内的数字，使用一个分布类型的对象：

```
// generate number range from 0 to 9 (inclusive) uniformtly.
uniform_int_distribution<unsigned> u(0,9);
default_random_engine e;
for(size_t i=0;i<10;++i)
    cout<<u(e)<<" ";// u作为随机数源，每个调用返回在指定范围内并服从均匀分布的值
// output: 0 1 7 4 5 2 6 6 9
```

这里，分布类型也是函数对象类。定义了一个调用运算符，接受一个随机数引擎作为参数。分布对象使用它的引擎参数生成随机数，并将其映射到指定的分布。

> 随机数发生器，指的是分布对象和引擎对象的组合。

引擎生成一个数值序列

对于一个给定的发生器，每次运行都会返回相同的数值序列。例如，

```
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;
}
```

每次调用该bad_randVec都会产生相同的100个数字，即

```
vector<unsigned> v1(bad_randVec());
// v1={9,2,3,4,1,2,....}
vector<unsigned> v2(bad_randVec());
// v2也是{9,2,3,4,1,2,....}
```

正确做法应该是将引擎和分布对象定义为static，即引擎和分布对象保持状态，第一次调用会使用u(e)生成的前100个随机数，第二次调用会获得接下来的100个。依此类推。

设置随机数发生器种子

我们可以通过提供一个种子 (seed) 来达到这一目的。引擎利用该种子从序列中的一个新位置重新开始生成随机数。

```
default_random_engine e1;    // 默认种子
default_random_engine e2(2147483646);    // 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);
// e3 和 e4 使用相同的种子，因此将会产生相同的序列
```

可以使用time生成比较随机的种子。

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

但是，也可能不是那么随机。例如，如果程序作为一个自动过程的一部分反复运行，time返回值可能都是相同的。

其他随机数分布

生成随机实数

C中的做法：

```
rand()/RAND_MAX
```

这种做法，不一定正确，因为随机整数的精度通常低于随机浮点数，这样有些浮点值就永远不会产生。

C++引用的uniform_real_distribution类型，可以处理从随机整数到随机浮点数的映射。

```
default_random_engine e;
uniform_real_distribution<double> u(0,1);
```

```
for(size_t i=0;i<10;++i)
    cout<<u(e);
```

这和生成unsigned的方式一模一样，但是这里使用了新的分布类型uniform_real_distribution。

每个分布模板都有一个默认模板实参，如前介绍的。需要用一个空的<>指出使用默认实参。

生成其他分布的随机数

可以生成正态分布的序列：normal_distribution生成浮点值。

```
default_random_engine e;
normal_distribution<> n(4,1.5); // 均值4，标准差1.4，使用默认模板实参，double
vector<unsigned> vals(9);   //
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())
        ++vals[v];          统计每个数出现的次数
}
```

更多分布，很少用到，略。