**第一个库 tuple**

**简介：像pair只能存放两个类型，如果你想要存放更多的类型，同时不想创作一个类就使用tuple**

**#include<tuple>**

tuple<int, vector<string>, string> a(10, { "小松","xiaomei" }, "一个人");

//初始化有三种方式第一种

tuple<int, vector<string>, string> b{ 10, { "美小","xiaomei" }, "一个大人" };

//第二种

tuple<int, string, string> c;

string bb("daxie"), cc("china");

c = make\_tuple(11, bb, cc);

//第三种

auto di1 = get<0>(a);

auto di2 = get<1>(a);

//tuple的get函数是得到其中的存放的数据

auto di3 = \*(di2.begin() + 1);

size\_t shumu = tuple\_size<decltype(a)>::value;

//计算一共有几个数据

tuple\_element<0, decltype(a)>::type cnt = get<0>(a);

//得到变量类型

**第二个库 bitset**

**简介是为了更加方便的操作字节的库**

bitset<32> bitset1("01"); //第一种初始化特别实用的初始化而且直接初始化字符串初始化

bitset<32> bitset2(100); //整数初始化

cout << bitset2.to\_ulong() << endl;//更多的操作请观看643页的表格 使用时候再去观看

**第三个库 正则表达式**

**库在#include<regex>**

string pattern("[0-9]{17,}");  **//首先书写正则表达式**

5 //**然后我们就将表达式放在里面**

smatch results;  **//创建一个我们需要存放的匹配变量**

if (regex\_search(lie,results, r)) { **//lie变量是我们需要处理的文件**

cout << results.str() << endl;}

regex r(pattern,regex::icase); **//其中的regex::icase与我们的io流的操作类似**

**作用是告诉编译器如何处理文件 regex::icase表示匹配过程中忽略大小写**

**具体看647页**

**对于我们书写的正则表达式我们可以判断他是不是正确的**

try{

regex r(pattern);

}

catch (regex\_error e) {

}

**what函数是获得什么错误 code函数是获得错误的序列号 具体看649页**

**不同类型存在不同的表达式库**

**char\*和string表达式库不一样但是我们的操作一样与io库类似 具体看650页**

**如果我们想要打印全部的匹配就需要使用regex的迭代器**

if (regex\_search(lie,results, r)) {

cout << results.str() << endl;}  **//只能够打印匹配的第一个**

for (sregex\_iterator it(lie.begin(), lie.end(), r),end\_it; it != end\_it; it++) {

cout << it->str().size() << endl;

cout << it->str() << endl;

}

**这个函数我们需要注意sregex\_iterator it(lie.begin(), lie.end(), r)这个**

**他是调用sregex\_search(b,e,r);来定位到第一个匹配的**

**我们还需要注意it迭代器所拥有的运算符 和其他功能 651页 和653的表格**

**这个其他功能指的 it->str().size() 计算大小**

**最后我们还有一个子表达式**

**我们前面的是处理匹配到的每一个表达式**

**而这个是处理每一表达式内部的子表达式**

if (regex\_search(lie,results, r)) {  **//还是前面的例子**

cout << results.str(0) << endl;}

**如果我将正则表达式** string pattern("[0-9]{17,}");

**改成** string pattern("（\w{15}）+（[0-9]{17,}）");

**这样我们** cout << results.str(1) << endl;} **取出的是匹配的字符**

cout << results.str(2) << endl;}  **取出的是匹配的数字**

cout << results.str(0) << endl;}  **注意 取出的是匹配的全部**

**同样他还有跟多的操作比如** results.length() **具体看656页**

**最后还有一个regex\_replace 用于匹配到后再代替为一个新形式**

**例子**  20011227 **转换成2001-12-27**

string fmt（“$2.$5.$7”）;

regex\_replace（number，r，fmt）；

number **是要处理的文件 r是匹配的正则表达式 fmt是一个正则表达式 同时又是一个字符串**

regex\_replace（number，r，fmt，match\_default）；

match\_default  **//与前面的作用一样是如何处理**

**但是我们要先声明的命名空间**

using namespace std:::regex\_constants;

**第四个库随机引擎**

**我们一般认为产生一个随机引擎来产生原始随机数**

**其次我们需要随机分布来指定范围**

**我们需要种子来产生不一样的随机数**

static uniform\_int\_distribution<unsigned> u(0, 80);  **随机分布**

static default\_random\_engine e; **随机引擎**

**他们来个都有一个调用运算符所以我们**

**随机引擎 的 调用运算符不接受参数**  e（）

**随机分布 的 调用运算符接受一个随机引擎** u（e）**产生一个随机数**

uniform\_int\_distribution<unsigned> 的<unsigned>**所以表示随机分布是一个模板**

default\_random\_engine e（12332）；这个12332**就是一个种子**

default\_random\_engine e（time（0））；**利用时间来产生一个数字；**

**但是这种产生的种子面对反复运行的程序就会失去作用**