#include <iostream>

#include <vector>

#include <tuple>

#include <bitset>

#include <regex>

#include <random>

using namespace std;

void TupleTest();

void BitsetTest();

void RegexTest();

void Randomtest();

void IOTest();

int main()

{

TupleTest();

BitsetTest();

RegexTest();

Randomtest();

IOTest();

cout << "enter key" << endl;

while (cin.get() != EOF) // ctrl + z

{

}

return 0;

}

// 当我们像将一些对象组合成单一对象时可以使用tuple

//

void TupleTest()

{

tuple<int, string> tuple1 = {1, "ab"};

get<0>(tuple1); // 返回第一个数据

get<0>(tuple1) = 2; // 设置第一个数据

int size = tuple\_size<decltype(tuple1)>::value; // 获取 tuple1 的大小

// 声明一个与 tuple1 0 一样类型的变量

// 为什么可以这样使用？？

// 因为模板是在编译时实例化，所以编译后下面的代码是 int a;

tuple\_element<0, decltype(tuple1)>::type a;

}

// 位运算

void BitsetTest()

{

bitset<16> bit1(0xfffe);

bitset<8> bit2("11001111");

bit1.any(); // 是否存在置位的二进制

bit1.all(); // 是否都置位了

bit1.count(); // 置位的位数

bit1.test(8); // 第 9 个位置是否置位

bit1.set(8, true); // 将第 9 个位置置位

bit1.reset(8); // 将第 9 个位置复位

bit1.to\_string(); // 返回字符串

}

// 正则表达式

void RegexTest()

{

string testStr = "abcdefgh";

// 生成正则，如果正则表达式错误，会抛出 regex\_error 异常

// 需要注意的是，正则是在运行时编译，编译过程缓慢

// 另有 wregex 宽字符版本

// () 用于子表达式 (a.\*?c) 为第 1 个子表达式，从 1 开始

regex r("(a.\*?c)d(ef)");

// 用于 string 类型的匹配结果

// 另有 cmatch 和 wssmatch

// cmatch 对应 const char \*

// wsmarch 对应 wstring

smatch sresult;

if (regex\_search(testStr, sresult, r))

{ // 匹配

cout << sresult.str() << endl;

cout << sresult.str(1) << endl; // 打印第一个子表达式

}

string replacefmt = "$1.$2"; // 替换格式，$1 表示第一个表达式

// 将匹配的字符串替换为replacefmt格式输出为

// testStr为abcdefgh，匹配为abcdef，格式为abc.ef，替换为abc.efgh

cout << regex\_replace(testStr, r, replacefmt) << endl;

}

// 随机数

void Randomtest()

{

// 随机数引擎

// 为什么使用static？如果不使用，那每次调用Randomtest都将生成相同的随机数

static default\_random\_engine random1;

cout << random1() << endl;

// 指定范围

static uniform\_int\_distribution<unsigned> u(0, 9);

cout << u(random1) << endl;

static uniform\_real\_distribution<double> ud(0, 1);

cout << ud(random1) << endl;

default\_random\_engine random2(8737); // 使用种子，相同的种子生成相同的随机数序列

// 非均匀分布随机数

normal\_distribution<> n(4, 1.5); // 均值4，标准差1.5

cout << n(random1) << endl;

}

// IO流

void IOTest()

{

// 操作符

// boolalpha 操作符使 cout 的bool打印变为 trur 或 false

cout << boolalpha << true << noboolalpha << endl;

// 控制输入格式

// 默认情况下，<< 会忽略 空白符（空格，制表，换行，换页，回车）

// 使用 noskipws，空白符也会读取，使用 noskipws 我们只能一个字符的读取，而不能使用 string

char c;

cin >> noskipws;

while (cin >> c)

{

cout << c;

};

cin >> skipws;

cout << endl;

cout << cin.fail() << endl;

// 未格式化输入输出操作

char c1;

cin.get(c1); // 读取一个字节

cout.put(c1); // 输出一个字节

c1 = cin.peek(); // 读取一个字节，但不从流中删除它

cin.unget(); // 向后移动一个字节（当我们读取一个字节时，向前移动一个字节）

cin.putback(c1); // 将读取的字节放回去

// 流随机访问

auto ipos = cin.tellg(); // 获取当前流的位置

cin.seekg(ipos); // 设置当前流的位置

auto opos = cout.tellp(); // 同上，istream 以 g 结尾，ostream 以 p 结尾

cout.seekp(opos);

}