C++ complex复数类用法详解

复数是 a+bi 形式的数，其中 a 和 b 是真数，在 C++ 代码中是浮点值，i 是根号 -1。a 被称作复数的实数部分，b 乘以 i 被称作虚数部分。

使用复数的程序一般都很专业，例如，复数可以用于电气和电磁理论、数字信号处理，当然也可以用于数学。复数可以用来生成非常复杂的 Mandelbrot 集合和 Julia 集合的分形图。

complex 头文件定义了用于处理复数的功能。complex<T> 模板类型的实例表示的是复数，这里定义了 3 个特化类型：complex<float>、complex<double>、complex<long double>。在这一节中，全部使用 complex<double>，但其他特化类型的操作是基本相同的。

生成表示复数的对象

complex<double> 类型的构造函数接受两个参数，第一个参数是实部的值，第二个部分是虚部的值。例如：

```
01.  std::complex<double> z1 {2, 5}; // 2 + 5i
02.  std::complex<double> z; // Default parameter values, are 0 so 0 + 0i
```

它也有拷贝构造函数，因此可以按如下方式复制 z1:

```
01.  std::complex<double> z2 {z1}; // 2 + 5i
```

显然，我们需要复数常量以及复数对象，命名空间 std::literals::complex_literals 中定义了 3 个运算符函数，在这个命名空间中，命名空间 literals 和 complex_literals 都是内联定义的。在对 std::literals::complex_literals、std::literals 或 std::complex_literals 使用 using 指令之后，就可以访问用于复数常量的运算符函数。假设使用了一个或多个这种指令，并且 using std::complex 对这一节余下的代码都有效。

运算符 ""i() 函数定义了实部为 0 的 complex<double> 类型的常量。因此，3i 是一个等同于 complex<double>{0，3} 的常量。当然，可以用实部和虚部表示复数。例如：

```
01.  z = 5.0 + 3i; // z is now complex<double>{5, 3}
```

这展示了如何定义两部分都是非零值的复数，并顺便说明已经为复数对象实现了赋值运算符。可以对 complex<float> 常量使用后缀if，对 complex<long double> 常量使用后缀il，例如 22if 或 3.

这些后缀是由函数 operator""if() 和 operator""il() 定义的。注意，不能写成 1.0+i 或 2.0+il，因为这里的 i 和 il 会被解释为变量名，必须写成 1.0 +Ii 和 2.0+1.0il。

所有的复数类型都定义了成员函数real()和imag()，它们可以用来访问对象的实部或虚部，或者用提供的参数设置这些部分。例如：

```
complex<double> z{1.5, -2.5}; // z: 1.5 - 2.5i
z.imag(99); // z: 1.5 + 99.0i
z.real(-4.5); // z: -4.5 + 99.0i
std::cout << "Real part: " << z.real()<< " Imaginary part: " << z.imag()<<
std::endl;
// Real part: -4.5 Imaginary part: 99
```

real() 和 imag() 接受参数的版本什么都不会返回。

有为复数对象实现流的插入和提取的非成员函数模板。当从流中读取一个复数时，它可能只有实部，例如 55，或者括号中只有实部，例如(2.6)，或者实部和虚部在由一个逗号隔开的括号中，例如(3,-2)。如果只提供了实部，虚部会为 0。下面是一个示例：

```
complex<double> z1, z2, z3; // 3 default objects 0+0i
std:: cout << "Enter 3 complex numbers: ";
std::cin >> z1 >> z2 >> z3;    // Read 3 complex numbers
std::cout << " z1 = " << z1 <<" z2 = " << z2 << " z3 = "<< z3 <<
std::endl;
```

下面是示例的输入和输出结果：

```
Enter 3 complex numbers: -4 (6) (-3, 7)
z1 = (-4,0) z2 = (6,0) z3 = (-3,7)
```

如果输入的一个复数没有括号，就不会有虚部。但是，在括号中可以省略虚部。复数的输出周围总是有括号，虚部即使为 0 也会被输出。

复数的运算

complex 类模板为有复数操作数的二元运算符 +、-、*、/ 及一元 + 和 - 运算符定义了非成员函数。成员函数定义了 +=、-=、*= 和 /=。下面是使用它们的一些示例：

```
complex<double> z {1,2};    // 1+2i
auto z1 = z + 3.0;  // 4+2i
auto z2 = z*z + (2.0 + 4i); // -1+8i
auto z3 = z1 - z2;  // 5-6i
z3 /= z2;   // 815385-0.523077i
```

注意，复数对象和数值常量之间的运算需要数值常量是正确的类型。不能将整数常量加到complex<double> 对象上；为了能够进行这个运算，必须写成 2.0。

复数上的比较和其他运算

一些非成员函数模板可以用来比较两个复数对象相等或不相等。也有 == 和 !=运算可以用来比较复数对象和数值，这里数值会被看作虚部为 0 的复数。为了相等，所有的部分都必须相等，如果操作数的实部或虚部不同，它们就不相等。例如：

```
01.  complex<double> z1 {3,4};   // 3+4i
02.  complex<double> z2 {4,-3};  // 4-3i
03.  std::cout << std::boolalpha<<(z1 == z2) << " "  // false
04.  << (z1 != (3.0 + 4i)) << " "  // false
05.  << (z2 == 4.0 - 3i) << '\n';  // true
```

注释中的结果很清楚。注意在最后一个比较中，编译器会将 4.0-3i 看作复数。

另一种比较复数的方法是比较它们的量。各部分值和复数的实部及虚部都相同的向量的量和复数相同，是两部分平分和的平方根。非成员函数模板 abs() 接受 complex<T> 类型的参数，并返回一个T类型的量。下面是一个将 abs() 函数应用到前面的代码段中定义的 z1 和 z2 上的示例：

```
01.  std::cout << std::boolalpha
02.  << (std::abs(z1) == std::abs(z2))    // true
03.  << " " <<std::abs(z2 + 4.0 + 9i);    // 10
```

最后的输出值是 10，因为作为 abs() 的参数的表达式的计算结果是 (8.0+6i)；82 和 62 是 100，平方根是 10。

- norm() 函数模板会返回复数的量的平方。
- arg() 模板会返回以弧度为单位的相角，是复数 z 对应的 std::atan(z.imag()/z.real())。
- conj() 函数模板会返回共轭复数，是 a+bi 和 a-bi。
- polar() 函数模板接受量和相角作为参数，并返回和它们对应的复数对象。
- prqj() 函数模板返回的复数是复数参数在黎曼球上的投影。

一些非成员函数模板提供了一整套的三角函数，并为复数参数提供了双曲函数。也有用于复数参数的 cmath 版本的函数 exp()、pow()、log()、log10() 和 sqrt()。下面是一个有趣的示例：

```
01.  complex<double> zc {0.0, std::acos(-1)};
02.  std::cout << (std::exp (zc) +1.0) << '\n'; // (0, 1.22465e-16) or zero
     near enough
```

acos(-1) 是 π，所以这揭示了欧拉方程令人震惊的真相，π 和欧拉数 e 是有关联的：$e^{i\pi}+1=0$。