

```

/*
 * complex.c
 *
 * This file provides the standard complex arithmetic functions:
 *
 * Complex complex_minus      (Complex z0, Complex z1),
 * Complex complex_plus      (Complex z0, Complex z1),
 * Complex complex_mult      (Complex z0, Complex z1),
 * Complex complex_div       (Complex z0, Complex z1),
 * Complex complex_sqrt      (Complex z),
 * Complex complex_conjugate  (Complex z),
 * Complex complex_negate    (Complex z),
 * Complex complex_real_mult  (double r, Complex z),
 * Complex complex_exp        (Complex z),
 * Complex complex_log        (Complex z, double approx_arg);
 * double complex_modulus    (Complex z);
 * double complex_modulus_squared (Complex z);
 * Boolean complex_nonzero    (Complex z);
 * Boolean complex_infinite   (Complex z);
 */

#include "kernel.h"

Complex Zero      = { 0.0, 0.0};
Complex One       = { 1.0, 0.0};
Complex Two       = { 2.0, 0.0};
Complex Four      = { 4.0, 0.0};
Complex MinusOne  = {-1.0, 0.0};
Complex I         = { 0.0, 1.0};
Complex TwoPiI    = { 0.0, TWO_PI};
Complex Infinity  = {1e34, 0.0};

Complex complex_plus(
    Complex z0,
    Complex z1)
{
    Complex sum;

    sum.real = z0.real + z1.real;
    sum.imag = z0.imag + z1.imag;

    return sum;
}

Complex complex_minus(
    Complex z0,
    Complex z1)
{
    Complex diff;

    diff.real = z0.real - z1.real;
    diff.imag = z0.imag - z1.imag;

    return diff;
}

Complex complex_div(
    Complex z0,
    Complex z1)
{
    double mod_sq;
    Complex quotient;

    mod_sq = z1.real * z1.real + z1.imag * z1.imag;
    if (mod_sq == 0.0)
    {
        if (z0.real != 0.0 || z0.imag != 0.0)
            return Infinity;
        else
            uFatalError("complex_div", "complex");
    }
    quotient.real = (z0.real * z1.real + z0.imag * z1.imag)/mod_sq;

```

```
    quotient.imag = (z0.imag * z1.real - z0.real * z1.imag)/mod_sq;

    return quotient;
}

Complex complex_mult(
    Complex z0,
    Complex z1)
{
    Complex product;

    product.real = z0.real * z1.real - z0.imag * z1.imag;
    product.imag = z0.real * z1.imag + z0.imag * z1.real;

    return product;
}

Complex complex_sqrt(
    Complex z)
{
    double mod,
           arg;
    Complex root;

    mod = sqrt(complex_modulus(z)); /* no need for safe_sqrt() */
    if (mod == 0.0)
        return Zero;
    arg = 0.5 * atan2(z.imag, z.real);
    root.real = mod * cos(arg);
    root.imag = mod * sin(arg);

    return root;
}

Complex complex_conjugate(
    Complex z)
{
    z.imag = - z.imag;

    return z;
}

Complex complex_negate(
    Complex z)
{
    z.real = - z.real;
    z.imag = - z.imag;

    return z;
}

Complex complex_real_mult(
    double r,
    Complex z)
{
    Complex multiple;

    multiple.real = r * z.real;
    multiple.imag = r * z.imag;

    return multiple;
}

Complex complex_exp(
    Complex z)
{
    double modulus;
    Complex result;
```

```
    modulus = exp(z.real);
    result.real = modulus * cos(z.imag);
    result.imag = modulus * sin(z.imag);

    return result;
}

Complex complex_log(
    Complex z,
    double approx_arg)
{
    Complex result;

    if (z.real == 0.0 && z.imag == 0.0)
    {
        uAcknowledge("log(0 + 0i) encountered");
        result.real = - DBL_MAX;
        result.imag = approx_arg;
        return result;
    }

    result.real = 0.5 * log(z.real * z.real + z.imag * z.imag);

    result.imag = atan2(z.imag, z.real);
    while (result.imag - approx_arg > PI)
        result.imag -= TWO_PI;
    while (approx_arg - result.imag > PI)
        result.imag += TWO_PI;

    return result;
}

double complex_modulus(
    Complex z)
{
    return sqrt(z.real * z.real + z.imag * z.imag); /* no need for safe_sqrt() */
}

double complex_modulus_squared(
    Complex z)
{
    return (z.real * z.real + z.imag * z.imag);
}

Boolean complex_nonzero(
    Complex z)
{
    return (z.real || z.imag);
}

Boolean complex_infinite(
    Complex z)
{
    return (z.real == Infinity.real && z.imag == Infinity.imag);
}
```