

```

/*
 * sl2c_matrices.c
 *
 * This file provides the following functions for working with SL2CMatrices:
 *
 * void sl2c_copy(SL2CMatrix dest, CONST SL2CMatrix source);
 * void sl2c_invert(CONST SL2CMatrix a, SL2CMatrix inverse);
 * void sl2c_complex_conjugate(CONST SL2CMatrix a, SL2CMatrix conjugate);
 * void sl2c_product(CONST SL2CMatrix a, CONST SL2CMatrix b, SL2CMatrix product);
 * void sl2c_adjoint(CONST SL2CMatrix a, SL2CMatrix adjoint);
 * Complex sl2c_determinant(CONST SL2CMatrix a);
 * void sl2c_normalize(SL2CMatrix a);
 * Boolean sl2c_matrix_is_real(CONST SL2CMatrix a);
 */

```

```
#include "kernel.h"
```

```

void sl2c_copy(
    SL2CMatrix    dest,
    CONST SL2CMatrix    source)
{
    int i,
        j;

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            dest[i][j] = source[i][j];
}

```

```

void sl2c_invert(
    CONST SL2CMatrix    a,
    SL2CMatrix    inverse)
{
    Complex temp;

    temp = a[0][0];
    inverse[0][0] = a[1][1];
    inverse[1][1] = temp;

    inverse[0][1] = complex_negate(a[0][1]);
    inverse[1][0] = complex_negate(a[1][0]);
}

```

```

void sl2c_complex_conjugate(
    CONST SL2CMatrix    a,
    SL2CMatrix    conjugate)
{
    int i,
        j;

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            conjugate[i][j] = complex_conjugate(a[i][j]);
}

```

```

void sl2c_product(
    CONST SL2CMatrix    a,
    CONST SL2CMatrix    b,
    SL2CMatrix    product)
{
    int i,
        j;
    SL2CMatrix temp;

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            temp[i][j] = complex_plus(
                (
                    complex_mult(a[i][0], b[0][j]),
                    complex_mult(a[i][1], b[1][j])
                )
            );
}

```

```

        );

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            product[i][j] = temp[i][j];
}

void sl2c_adjoint(
    CONST SL2CMatrix  a,
    SL2CMatrix  adjoint)
{
    int  i,
        j;
    SL2CMatrix  temp;

    /*
     * We initially write the result into a temporary matrix,
     * so that if the matrices a and adjoint are the same the
     * [1][0] entry won't overwrite the [0][1] entry.
     */

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            temp[i][j] = complex_conjugate(a[j][i]);

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            adjoint[i][j] = temp[i][j];
}

Complex sl2c_determinant(
    CONST SL2CMatrix  a)
{
    return(
        complex_minus (
            complex_mult(a[0][0], a[1][1]),
            complex_mult(a[0][1], a[1][0])
        )
    );
}

void sl2c_normalize(
    SL2CMatrix  a)
{
    /*
     * If the matrix a is nonsingular, normalize it to have determinant one.
     * Otherwise, generate an error message and quit.
     */

    int  i,
        j;
    Complex det,
        factor;

    det = sl2c_determinant(a);

    if (complex_nonzero(det) == FALSE)
        uFatalError("sl2c_normalize", "sl2c_matrices");

    factor = complex_sqrt(det);

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            a[i][j] = complex_div(a[i][j], factor);
}

Boolean sl2c_matrix_is_real(
    CONST SL2CMatrix  a)
{
    int i,

```

```
        j;  
  
    for (i = 0; i < 2; i++)  
        for (j = 0; j < 2; j++)  
            if (a[i][j].imag != 0.0)  
                return FALSE;  
  
    return TRUE;  
}
```