```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define SIZE       8
#define PI         3.141592653589793
#define TWO_PI     (2.0 * PI)
#define SWAP(a,b)  tempr=(a);(a)=(b);(b)=tempr



//  The four1 FFT from Numerical Recipes in C,
//  p. 507 - 508.
//  Note:  changed float data types to double.
//  nn must be a power of 2, and use +1 for
//  isign for an FFT, and -1 for the Inverse FFT.
//  The data is complex, so the array size must be
//  nn*2. This code assumes the array starts
//  at index 1, not 0, so subtract 1 when
//  calling the routine (see main() below).

void four1(double data[], int nn, int isign)
{
    unsigned long n, mmax, m, j, istep, i;
    double wtemp, wr, wpr, wpi, wi, theta;
    double tempr, tempi;

    n = nn << 1;
    j = 1;

    for (i = 1; i < n; i += 2) {
        if (j > i) {
            SWAP(data[j], data[i]);
            SWAP(data[j+1], data[i+1]);
        }
        m = nn;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }

    mmax = 2;
    while (n > mmax) {
        istep = mmax << 1;
        theta = isign * (6.28318530717959 / mmax);
        wtemp = sin(0.5 * theta);
        wpr = -2.0 * wtemp * wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m = 1; m < mmax; m += 2) {
            for (i = m; i <= n; i += istep) {
                j = i + mmax;
                tempr = wr * data[j] - wi * data[j+1];
                tempi = wr * data[j+1] + wi * data[j];
                data[j] = data[i] - tempr;
                data[j+1] = data[i+1] - tempi;
                data[i] += tempr;
                data[i+1] += tempi;
            }
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        mmax = istep;
    }
}



// Creates a sine tone with the specified harmonic number.
// The array will be filled with complex numbers, and the
// signal is real (the imaginary parts are set to 0).

void createComplexSine(double data[], int size, int harmonicNumber)
{
    int i, ii;

    for (i = 0, ii = 0; i < size; i++, ii += 2) {
        data[ii] = sin((double)harmonicNumber * (double)i * TWO_PI / (double)size);
        data[ii+1] = 0.0;
    }
}



// Creates a cosine tone with the specified harmonic number.
// The array will be filled with complex numbers, and the
// signal is real (the imaginary parts are set to 0).

void createComplexCosine(double data[], int size, int harmonicNumber)
{
    int i, ii;

    for (i = 0, ii = 0; i < size; i++, ii += 2) {
        data[ii] = cos((double)harmonicNumber * (double)i * TWO_PI / (double)size);
        data[ii+1] = 0.0;
    }
}



// Creates a sawtooth wave, where each harmonic has
// the amplitude of 1 / harmonic_number.
// The array will be filled with complex numbers, and the
// signal is real (the imaginary parts are set to 0)

void createComplexSawtooth(double data[], int size)
{
    int i, ii, j;

    //  Calculate waveform using additive synthesis
    for (i = 0, ii = 0; i < size; i++, ii += 2) {
        data[ii] = 0.0;
        data[ii+1] = 0.0;
        for (j = 1; j <= size/2; j++) {
            data[ii] +=
                (cos((double)j * (double)i * TWO_PI / (double)size)) / (double)j;
        }
    }
}



// Display the real and imaginary parts
// the data contained in the array.

void displayComplex(double data[], int size)
{
    int i, ii;

    printf("\t\tReal part \tImaginary Part\n");

    for (i = 0, ii = 0; i < size; i++, ii += 2)
        printf("data[%-d]: \t%.6f \t%.6f\n", i, data[ii], data[ii+1]);

    printf("\n");
}



// Performs the DFT on the input data,
// which is assumed to be a real signal.
// That is, only data at even indices is
// used to calculate the spectrum.

void complexDFT(double x[], int N)
{
    int n, k, nn;
    double omega = TWO_PI / (double)N;
    double *a, *b;

    // Allocate temporary arrays
    a = (double *)calloc(N, sizeof(double));
    b = (double *)calloc(N, sizeof(double));

    // Perform the DFT
    for (k = 0; k < N; k++) {
        a[k] = b[k] = 0.0;
        for (n = 0, nn = 0; n < N; n++, nn += 2) {
            a[k] += (x[nn] * cos(omega * n * k));
            b[k] -= (x[nn] * sin(omega * n * k));
        }
    }

    // Pack result back into input data array
    for (n = 0, k = 0; n < N*2; n += 2, k++) {
        x[n] = a[k];
        x[n+1] = b[k];
    }

    // Free up memory used for arrays
    free(a);
    free(b);
}



// Takes the results from a DFT or FFT, and
// calculates and displays the amplitudes of
// the harmonics.

void postProcessComplex(double x[], int N)
{
    int i, k, j;
    double *amplitude, *result;

    // Allocate temporary arrays
    amplitude = (double *)calloc(N, sizeof(double));
    result = (double *)calloc(N, sizeof(double));

    // Calculate amplitude
    for (k = 0, i = 0; k < N; k++, i += 2) {
        // Scale results by N
        double real = x[i] / (double)N;
        double imag = x[i+1] / (double) N;
        // Calculate amplitude
        amplitude[k] = sqrt(real * real + imag * imag);
    }

    // Combine amplitudes of positive and negative frequencies
    result[0] = amplitude[0];
    result[N/2] = amplitude[N/2];
    for (k = 1, j = N-1; k < N/2; k++, j--)
        result[k] = amplitude[k] + amplitude[j];


    // Print out final result
    printf("Harmonic \tAmplitude\n");
    printf("DC \t\t%.6f\n", result[0]);
    for (k = 1; k <= N/2; k++)
        printf("%-d \t\t%.6f\n", k, result[k]);
    printf("\n");

    // Free up memory used for arrays
    free(amplitude);
    free(result);
}



int main()
{
    int i;
    double complexData[SIZE*2];

    // Try the DFT on a sawtooth waveform
    createComplexSawtooth(complexData, SIZE);
    displayComplex(complexData, SIZE);
    complexDFT(complexData, SIZE);
    postProcessComplex(complexData, SIZE);

    // Try the FFT on the same data
    createComplexSawtooth(complexData, SIZE);
    displayComplex(complexData, SIZE);
    four1(complexData-1, SIZE, 1);
    postProcessComplex(complexData, SIZE);
}
```