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

#define MAX_SIZE 64

void CircularConvolve(float *input, int size, float *impulseResponse, float *output);
void FFT8Point(int size, float input[8][2], float output[8][2]);
void FFT4Point(int size, float input[4][2], float output[4][2]);

int main() {
   int i, j, k, signalLength, impulseLength, size, length;
   float signal[MAX_SIZE], paddedSignal[MAX_SIZE], tempOutput[MAX_SIZE], result[MAX_SIZE];
   float impulse[MAX_SIZE];

   for (i = 0; i < MAX_SIZE; i++) {
      signal[i] = 0;
      impulse[i] = 0;
      result[i] = 0;
      paddedSignal[i] = 0;
   }

   printf("\nEnter the length of x[n]: ");
   scanf("%d", &signalLength);

   printf("\nEnter the values of x[n]: ");
   for (i = 0; i < signalLength; i++) {
      scanf("%f", &signal[i]);
   }

   printf("\nEnter the length of h[n]: ");
   scanf("%d", &impulseLength);

   printf("\nEnter the values of h[n]: ");
   for (i = 0; i < impulseLength; i++) {
      scanf("%f", &impulse[i]);
   }

   printf("\nx[n] = ");
   for (i = 0; i < signalLength; i++) {
      printf("  %4.2f  ", signal[i]);
   }

   printf("\nh[n] = ");
   for (i = 0; i < impulseLength; i++) {
      printf("  %4.2f  ", impulse[i]);
   }

   size = 8;
   length = size - impulseLength + 1;
```

```
    printf("\n\nLength of decomposed input signal: L = %d", length);
    printf("\nLength of decomposed output signal: N = %d", size);

    for (i = 0; i < length; i++) {
        paddedSignal[i + impulseLength - 1] = signal[i];
    }

    j = 0;
    int overlapIndex = 0;

    for (k = 0; k <= signalLength; k += length) {
        CircularConvolve(paddedSignal, size, impulse, tempOutput);

        for (i = 0; i < length; i++) {
            result[k + i] = tempOutput[i + impulseLength - 1];
        }

        j++;

        printf("\n\nx[n] = ");
        for (i = 0; i < size; i++) {
            printf("  %4.2f  ", paddedSignal[i]);
        }

        printf("\ny[n] = ");
        for (i = 0; i < size; i++) {
            printf("  %4.2f  ", tempOutput[i]);
        }

        overlapIndex = k + length - (impulseLength - 1);
        for (i = 0; i < size; i++) {
            paddedSignal[i] = signal[overlapIndex + i];
        }
    }

    printf("\n\nLinear Convolution Output using Overlap Save Method:");
    printf("\ny[n] = ");
    for (i = 0; i < (signalLength + impulseLength - 1); i++) {
        printf("  %4.2f  ", result[i]);
    }
    printf("\n\n");

    return 0;
}

void CircularConvolve(float *input, int size, float *impulseResponse, float *output) {
    int i, k;
    float inputFFT[MAX_SIZE][2], impulseFFT[MAX_SIZE][2], resultFFT[MAX_SIZE][2], tempFFT[MAX_SIZE][2];
```

```
for (k = 0; k < size; k++) {
   inputFFT[k][0] = 0;
   inputFFT[k][1] = 0;
   impulseFFT[k][0] = 0;
   impulseFFT[k][1] = 0;
   resultFFT[k][0] = 0;
   resultFFT[k][1] = 0;
}

for (i = 0; i < size; i++) {
   tempFFT[i][0] = input[i];
   tempFFT[i][1] = 0;
}

if (size == 4) {
   FFT4Point(size, tempFFT, inputFFT);
} else if (size == 8) {
   FFT8Point(size, tempFFT, inputFFT);
}

for (i = 0; i < size; i++) {
   tempFFT[i][0] = impulseResponse[i];
   tempFFT[i][1] = 0;
}

if (size == 4) {
   FFT4Point(size, tempFFT, impulseFFT);
} else if (size == 8) {
   FFT8Point(size, tempFFT, impulseFFT);
}

for (k = 0; k < size; k++) {
   float a = inputFFT[k][0];
   float b = inputFFT[k][1];
   float c = impulseFFT[k][0];
   float d = impulseFFT[k][1];

   resultFFT[k][0] = (a * c) - (b * d);
   resultFFT[k][1] = (b * c) + (a * d);
}

for (k = 0; k < size; k++) {
   resultFFT[k][1] = -resultFFT[k][1];
}

if (size == 4) {
   FFT4Point(size, resultFFT, tempFFT);
} else if (size == 8) {
```

```
    FFT8Point(size, resultFFT, tempFFT);
  }

  for (i = 0; i < size; i++) {
    output[i] = tempFFT[i][0] / size;
  }
}

void FFT8Point(int size, float input[8][2], float output[8][2]) {
  int i, k;
  float angle = 6.283185307179586 / size;
  float evenFFT[4][2], oddFFT[4][2];

  for (i = 0; i < 4; i++) {
    evenFFT[i][0] = input[2 * i][0];
    evenFFT[i][1] = input[2 * i][1];
    oddFFT[i][0] = input[2 * i + 1][0];
    oddFFT[i][1] = input[2 * i + 1][1];
  }

  FFT4Point(size / 2, evenFFT, evenFFT);
  FFT4Point(size / 2, oddFFT, oddFFT);

  for (k = 0; k < 4; k++) {
    float currentAngle = angle * k;
    float cosAngle = cos(currentAngle);
    float sinAngle = sin(currentAngle);

    output[k][0] = evenFFT[k][0] + (oddFFT[k][0] * cosAngle - oddFFT[k][1] * sinAngle);
    output[k][1] = evenFFT[k][1] + (oddFFT[k][1] * cosAngle + oddFFT[k][0] * sinAngle);

    output[k + 4][0] = evenFFT[k][0] - (oddFFT[k][0] * cosAngle - oddFFT[k][1] * sinAngle);
    output[k + 4][1] = evenFFT[k][1] - (oddFFT[k][1] * cosAngle + oddFFT[k][0] * sinAngle);
  }
}

void FFT4Point(int size, float input[4][2], float output[4][2]) {
  int k;
  float angle = 6.283185307179586 / size;
  float evenFFT[2][2], oddFFT[2][2];

  evenFFT[0][0] = input[0][0] + input[2][0];
  evenFFT[0][1] = input[0][1] + input[2][1];
  evenFFT[1][0] = input[0][0] - input[2][0];
  evenFFT[1][1] = input[0][1] - input[2][1];

  oddFFT[0][0] = input[1][0] + input[3][0];
  oddFFT[0][1] = input[1][1] + input[3][1];
  oddFFT[1][0] = input[1][0] - input[3][0];
```

```
    oddFFT[1][1] = input[1][1] - input[3][1];

    for (k = 0; k < 4; k++) {
        float currentAngle = angle * k;
        float cosAngle = cos(currentAngle);
        float sinAngle = sin(currentAngle);

        output[k][0] = evenFFT[k % 2][0] + (oddFFT[k % 2][0] * cosAngle - oddFFT[k % 2][1] * sinAngle);
        output[k][1] = evenFFT[k % 2][1] + (oddFFT[k % 2][1] * cosAngle + oddFFT[k % 2][0] * sinAngle);
    }
}
```