```c
// OAM by prathamesh 2022200078

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

#define MAX_SIZE 64

void CircularConvolve(float *inputSegment, int segmentSize, float *impulseResponse, float *outputSegment);
void FFT_8_Point(int segmentSize, float input[8][2], float output[8][2]);
void FFT_4_Point(int segmentSize, float input[4][2], float output[4][2]);

int main() {
   int i, j, k, signalLength, impulseLength, segmentSize, segmentLen;
   float signal[MAX_SIZE], tempSegment[MAX_SIZE], outputSegment[MAX_SIZE], finalOutput[MAX_SIZE];
   float impulse[MAX_SIZE];

   // Initialize arrays
   for (i = 0; i < MAX_SIZE; i++) {
      signal[i] = 0;
      impulse[i] = 0;
      finalOutput[i] = 0;
      tempSegment[i] = 0;
   }

   //-------------------- INPUT -----------------------------------

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

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

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

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

   // Display input signal and impulse response
   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]);
}

//-------------------- OAM -----------------------------------------

segmentSize = 8;  // Fixed segment size for FFT processing
segmentLen = segmentSize - impulseLength + 1;  // Length of decomposed signal (L)

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

j = 0;

for (k = 0; k <= signalLength - segmentLen; k += segmentLen) {
   // Split signal into segments
   for (i = 0; i < segmentLen; i++) {
      tempSegment[i] = signal[k + i];
   }

   // Perform circular convolution using FFT
   CircularConvolve(tempSegment, segmentSize, impulse, outputSegment);

   // Overlap and add
   for (i = 0; i < segmentSize; i++) {
      finalOutput[k + i] += outputSegment[i];
   }

   j++;

   // Display decomposed signal and output
   printf("\n\nx%d[n] = ", j);
   for (i = 0; i < segmentSize; i++) {
      printf("  %4.2f  ", tempSegment[i]);
   }

   printf("\ny%d[n] = ", j);
   for (i = 0; i < segmentSize; i++) {
      printf("  %4.2f  ", outputSegment[i]);
   }
}

//-------------------------------------------------------------------

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

```
    printf("\n\n");

}

//==================================================================================

void CircularConvolve(float *inputSegment, int segmentSize, float *impulseResponse, float *outputSegment) {
    int i, k;
    float InputFFT[MAX_SIZE][2], ImpulseFFT[MAX_SIZE][2], ResultFFT[MAX_SIZE][2], TempFFT[MAX_SIZE][2];

    // Initialize FFT arrays
    for (k = 0; k < segmentSize; 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;
    }

    // Copy input segment to FFT array
    for (i = 0; i < segmentSize; i++) {
        TempFFT[i][0] = inputSegment[i];
        TempFFT[i][1] = 0;
    }

    // Compute FFT of input segment
    if (segmentSize == 4) {
        FFT_4_Point(segmentSize, TempFFT, InputFFT);
    } else if (segmentSize == 8) {
        FFT_8_Point(segmentSize, TempFFT, InputFFT);
    }

    // Copy impulse response to FFT array
    for (i = 0; i < segmentSize; i++) {
        TempFFT[i][0] = impulseResponse[i];
        TempFFT[i][1] = 0;
    }

    // Compute FFT of impulse response
    if (segmentSize == 4) {
        FFT_4_Point(segmentSize, TempFFT, ImpulseFFT);
    } else if (segmentSize == 8) {
        FFT_8_Point(segmentSize, TempFFT, ImpulseFFT);
    }

    // Multiply FFTs in the frequency domain
    for (k = 0; k < segmentSize; 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);
    }

    // Inverse FFT
    for (k = 0; k < segmentSize; k++) {
        ResultFFT[k][1] = -ResultFFT[k][1];
    }

    if (segmentSize == 4) {
        FFT_4_Point(segmentSize, ResultFFT, TempFFT);
    } else if (segmentSize == 8) {
        FFT_8_Point(segmentSize, ResultFFT, TempFFT);
    }

    // Normalize the result
    for (i = 0; i < segmentSize; i++) {
        outputSegment[i] = TempFFT[i][0] / segmentSize;
    }
}

//===============================================================================

void FFT_8_Point(int segmentSize, float input[8][2], float output[8][2]) {
    int i, k;
    float W = 6.283185307179586 / segmentSize;
    float EvenFFT[4][2], OddFFT[4][2];

    // Split input into even and odd parts
    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];
    }

    // Perform FFT on even and odd parts
    FFT_4_Point(4, EvenFFT, EvenFFT);
    FFT_4_Point(4, OddFFT, OddFFT);

    // Combine the results
    for (k = 0; k < 4; k++) {
        float angle = W * k;
        float cosAngle = cos(angle);
        float sinAngle = sin(angle);

        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 FFT_4_Point(int segmentSize, float input[4][2], float output[4][2]) {
    int k;
    float W = 6.283185307179586 / segmentSize;
    float G[4][2], H[4][2];

    // Stage-1
    G[0][0] = input[0][0] + input[2][0];
    G[0][1] = input[0][1] + input[2][1];
    G[1][0] = input[0][0] - input[2][0];
    G[1][1] = input[0][1] - input[2][1];

    H[0][0] = input[1][0] + input[3][0];
    H[0][1] = input[1][1] + input[3][1];
    H[1][0] = input[1][0] - input[3][0];
    H[1][1] = input[1][1] - input[3][1];

    // Stage-2
    for (k = 0; k < 4; k++) {
        float angle = W * k;
        float cosAngle = cos(angle);
        float sinAngle = sin(angle);

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