Dec 6th, 2023

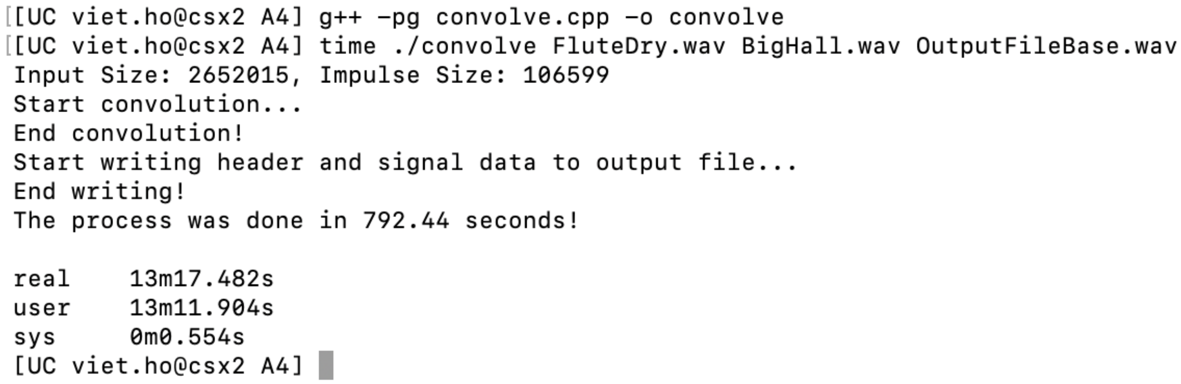
CPSC 501

Assignment #4

Report

Authored by Viet Ho (30122283)

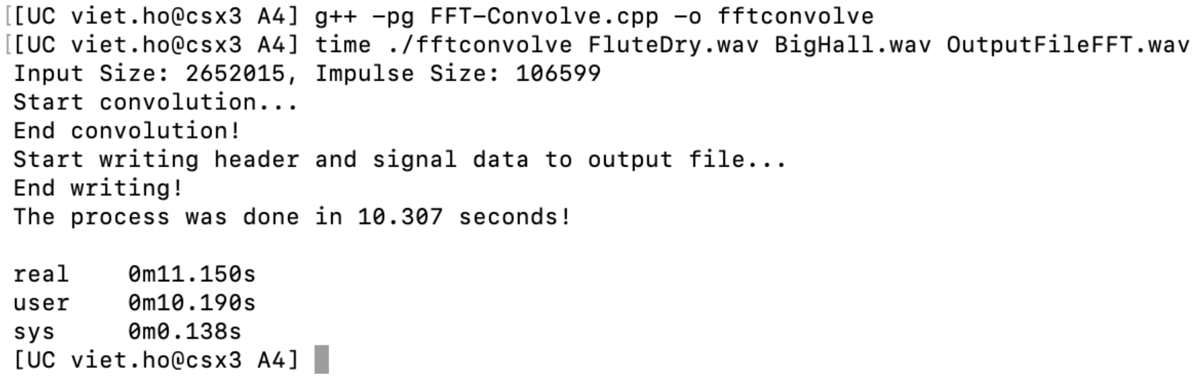
1. **Baseline Program**
   A program where the convolution is implemented directly in the time domain (use the input-side convolution algorithm found on p. 112-115 in the Smith text). The program was invoked from the command line as follows: convolve FluteDry.wav BigHall.wav OutputFileBase.wav.



```
[[UC viet.ho@csx2 A4] g++ -pg convolve.cpp -o convolve
[[UC viet.ho@csx2 A4] time ./convolve FluteDry.wav BigHall.wav OutputFileBase.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 792.44 seconds!

 real    13m17.482s
 user    13m11.904s
 sys     0m0.554s
 [UC viet.ho@csx2 A4] 
```

Program Version	Time Measurement (seconds)	Time Reduction (%)
Baseline (convolve.cpp)	792.44	N/a

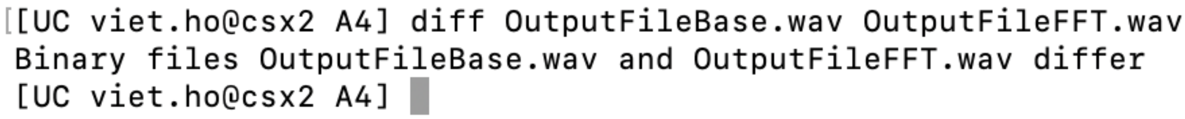
2. **Algorithm-Based Optimization Program**
   A program based on the baseline program, with re-implementing the convolution using a frequency-domain convolution algorithm. The program was invoked from the command line as follows: FFT-Convolve FluteDry.wav BigHall.wav OutputFileFFT.wav.



```
[[UC viet.ho@csx3 A4] g++ -pg FFT-Convolve.cpp -o fftconvolve
[[UC viet.ho@csx3 A4] time ./fftconvolve FluteDry.wav BigHall.wav OutputFileFFT.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 10.307 seconds!

 real    0m11.150s
 user    0m10.190s
 sys     0m0.138s
 [UC viet.ho@csx3 A4] 
```

Program Version	Time Measurement (seconds)	Time Reduction
Baseline (convolve.cpp)	792.44	N/a
FFT (FFT-Convolve.cpp)	10.307	98.7%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileBase.wav OutputFileFFT.wav). There are some rounding when computing FFT convolution, so the result of FFT convolution slightly differs from time-domain convolution (i.e. 1.999 may be considered 2).

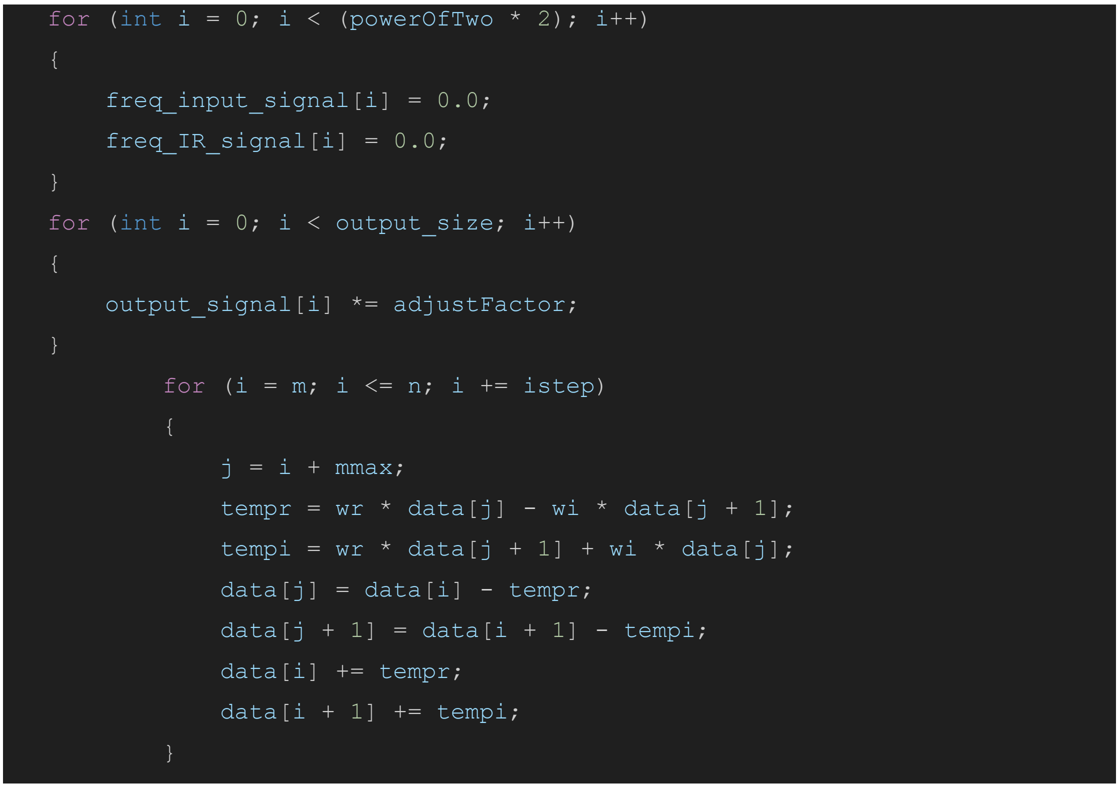


```
[UC viet.ho@csx2 A4] diff OutputFileBase.wav OutputFileFFT.wav
Binary files OutputFileBase.wav and OutputFileFFT.wav differ
[UC viet.ho@csx2 A4]
```

3. First Optimization Program - Loop Unrolling

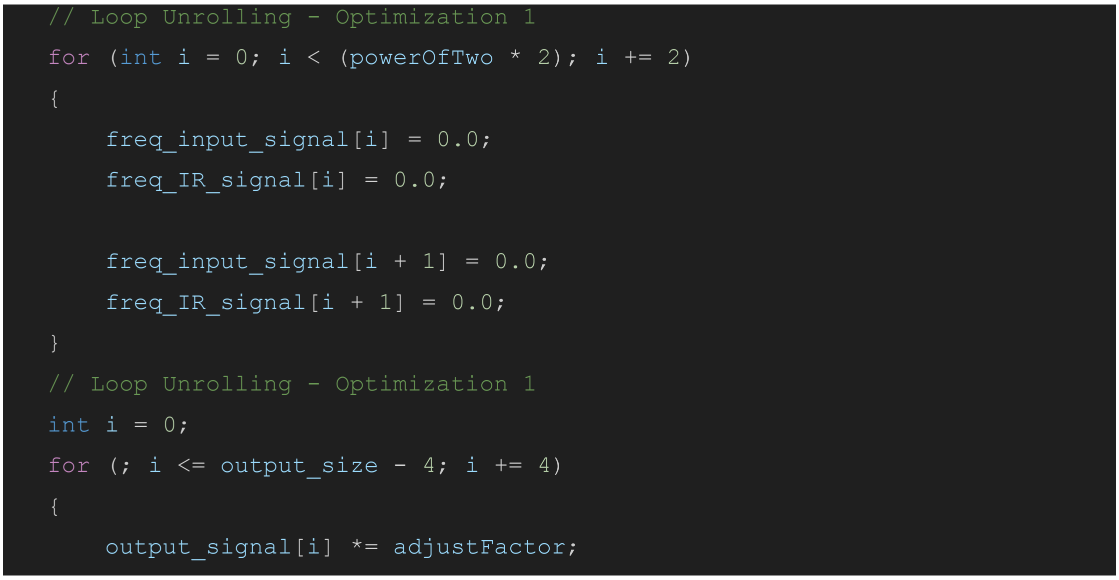
Loop unrolling is an optimization technique that replicates and unfolds a loop's iterations to reduce overhead and improve performance by minimizing loop control instructions. The program was invoked from the command line as follows: FFT-Convolve-1 FluteDry.wav BigHall.wav OutputFileFFT1.wav.

Before:



```
for (int i = 0; i < (powerOfTwo * 2); i++)
{
    freq_input_signal[i] = 0.0;
    freq_IR_signal[i] = 0.0;
}
for (int i = 0; i < output_size; i++)
{
    output_signal[i] *= adjustFactor;
}
        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;
        }
```

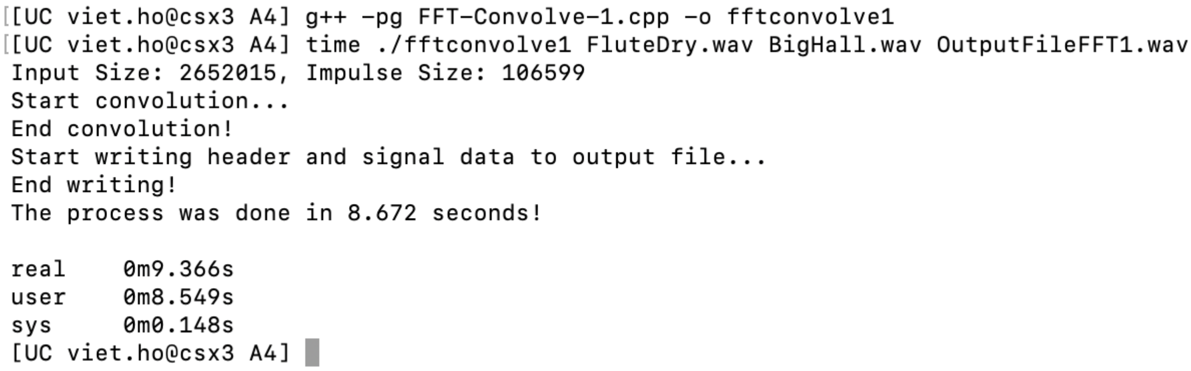
After:



```
// Loop Unrolling - Optimization 1
for (int i = 0; i < (powerOfTwo * 2); i += 2)
{
    freq_input_signal[i] = 0.0;
    freq_IR_signal[i] = 0.0;

    freq_input_signal[i + 1] = 0.0;
    freq_IR_signal[i + 1] = 0.0;
}
// Loop Unrolling - Optimization 1
int i = 0;
for (; i <= output_size - 4; i += 4)
{
    output_signal[i] *= adjustFactor;
```

```
        output_signal[i + 1] *= adjustFactor;
        output_signal[i + 2] *= adjustFactor;
        output_signal[i + 3] *= adjustFactor;
    }

    for (; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
            // Loop Unrolling - Optimization 1
            for (i = m; i <= n; i += istep * 2)
            {
                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;

                if (i + istep <= n)
                {
                    int j_unrolled = i + istep + mmax;
                         double tempr_unrolled = wr * data[j_unrolled] - wi *
data[j_unrolled + 1];
                      double tempi_unrolled = wr * data[j_unrolled + 1] + wi *
data[j_unrolled];
                    data[j_unrolled] = data[i + istep] - tempr_unrolled;
                              data[j_unrolled + 1] = data[i + istep + 1] -
tempi_unrolled;
                    data[i + istep] += tempr_unrolled;
                    data[i + istep + 1] += tempi_unrolled;
                }
            }
```

Result:

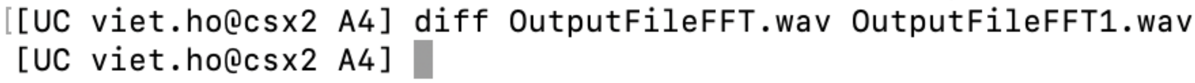


```
[[UC viet.ho@csx3 A4] g++ -pg FFT-Convolve-1.cpp -o fftconvolve1
[[UC viet.ho@csx3 A4] time ./fftconvolve1 FluteDry.wav BigHall.wav OutputFileFFT1.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 8.672 seconds!

 real    0m9.366s
 user    0m8.549s
 sys     0m0.148s
 [UC viet.ho@csx3 A4] 
```

Program Version	Time Measurement (seconds)	Time Reduction

FFT (FFT-Convolve.cpp)	10.307	N/a
FFT-Optimization-1 (FFT-Convolve-1.cpp)	8.672	15.9%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileFFT.wav OutputFileFFT1.wav). There are no differences between them so the diff command doesn't produce an output.

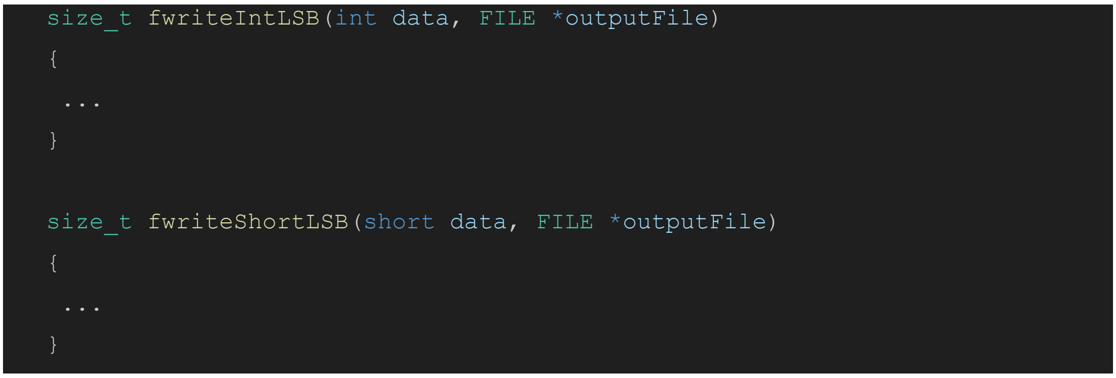


```
[[UC viet.ho@csx2 A4] diff OutputFileFFT.wav OutputFileFFT1.wav
 [UC viet.ho@csx2 A4] █
```

4. Second Optimization Program - Inline Functions

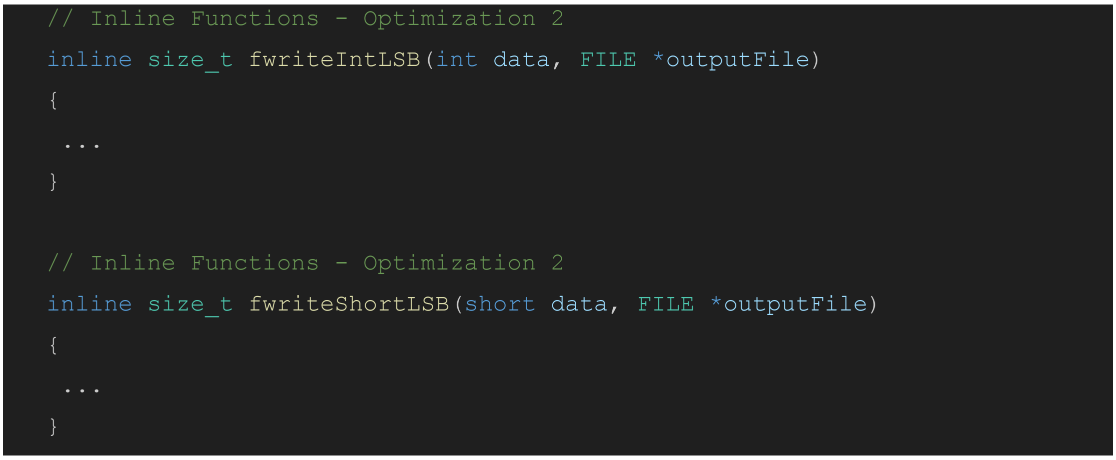
Inline functions are optimization where the function's code is inserted directly at the call site, eliminating the overhead of a function call and potentially improving performance. The program was invoked from the command line as follows: FFT-Convolve-2 FluteDry.wav BigHall.wav OutputFileFFT2.wav.

Before:



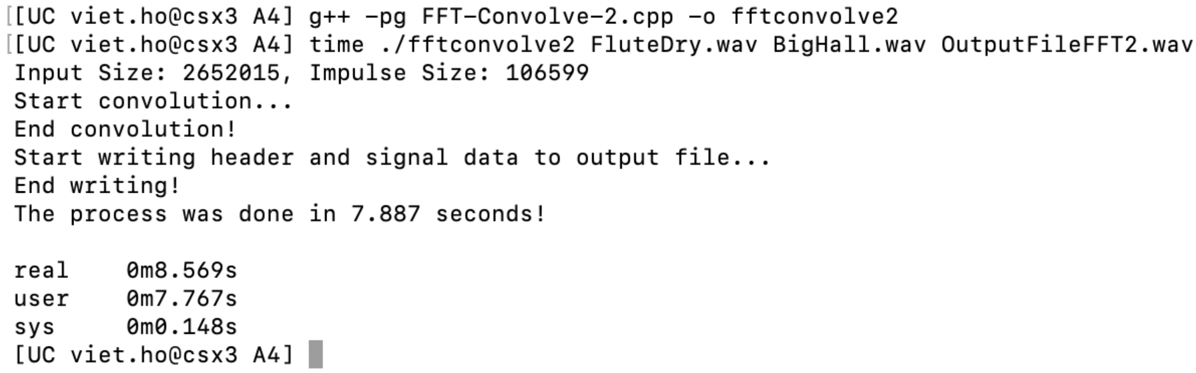
```
  size_t fwriteIntLSB(int data, FILE *outputFile)
  {
   ...
  }

  size_t fwriteShortLSB(short data, FILE *outputFile)
  {
   ...
  }
```

After:



```
  // Inline Functions - Optimization 2
  inline size_t fwriteIntLSB(int data, FILE *outputFile)
  {
   ...
  }

  // Inline Functions - Optimization 2
  inline size_t fwriteShortLSB(short data, FILE *outputFile)
  {
   ...
  }
```

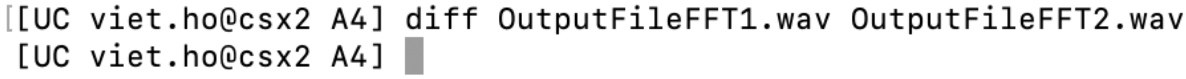
Result:



```
[UC viet.ho@csx3 A4] g++ -pg FFT-Convolve-2.cpp -o fftconvolve2
[UC viet.ho@csx3 A4] time ./fftconvolve2 FluteDry.wav BigHall.wav OutputFileFFT2.wav
Input Size: 2652015, Impulse Size: 106599
Start convolution...
End convolution!
Start writing header and signal data to output file...
End writing!
The process was done in 7.887 seconds!

real    0m8.569s
user    0m7.767s
sys     0m0.148s
[UC viet.ho@csx3 A4]
```

Program Version	Time Measurement (seconds)	Time Reduction
FFT-Optimization-1 (FFT-Convolve-1.cpp)	8.672	N/a
FFT-Optimization-2 (FFT-Convolve-2.cpp)	7.887	9.05%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileFFT1.wav OutputFileFFT2.wav). There are no differences between them so the diff command doesn't produce an output.

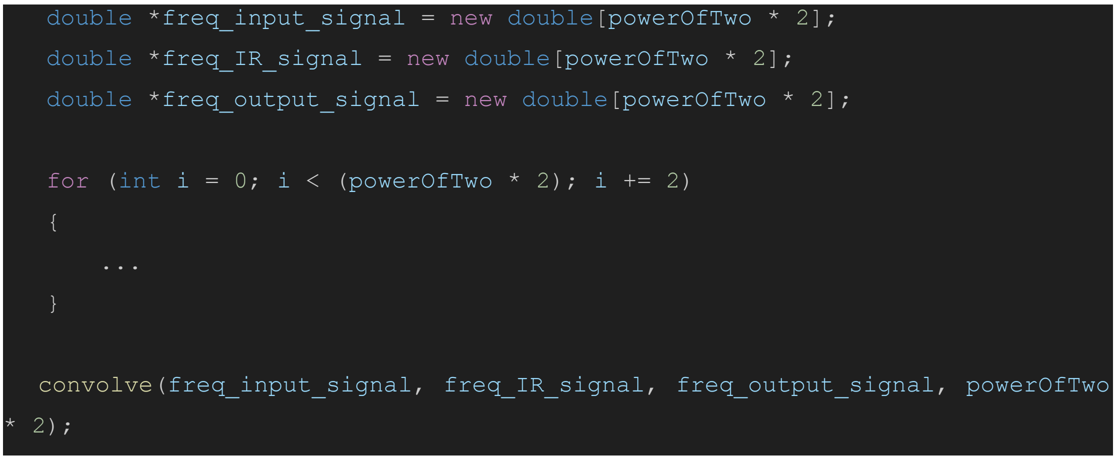


```
[UC viet.ho@csx2 A4] diff OutputFileFFT1.wav OutputFileFFT2.wav
[UC viet.ho@csx2 A4]
```

5. **Third Optimization Program - Avoiding Recomputation**

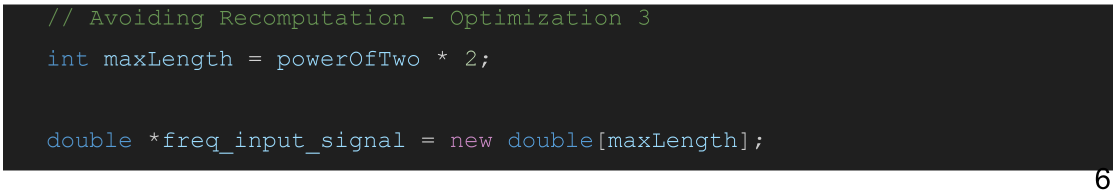
Avoiding recomputation involves storing and reusing previously calculated results to prevent redundant calculations and improve computational efficiency. The program was invoked from the command line as follows: FFT-Convolve-3 FluteDry.wav BigHall.wav OutputFileFFT3.wav.

Before:

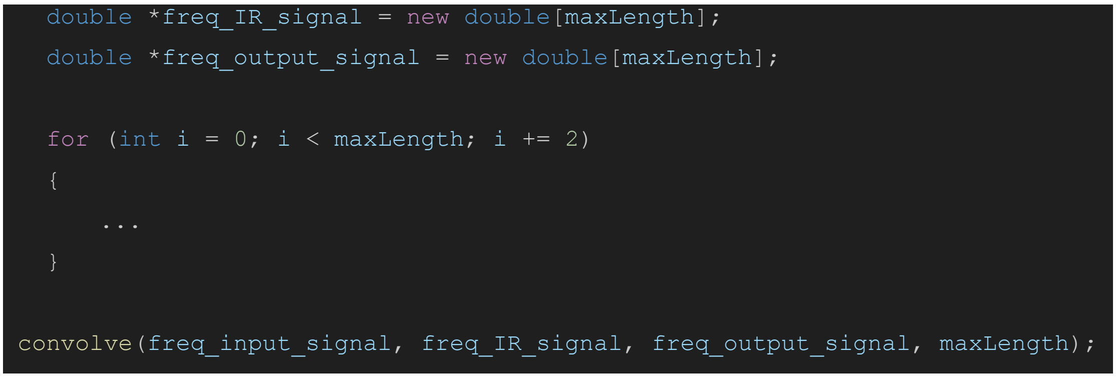


```
   double *freq_input_signal = new double[powerOfTwo * 2];
   double *freq_IR_signal = new double[powerOfTwo * 2];
   double *freq_output_signal = new double[powerOfTwo * 2];

   for (int i = 0; i < (powerOfTwo * 2); i += 2)
   {
       ...
   }

  convolve(freq_input_signal, freq_IR_signal, freq_output_signal, powerOfTwo
* 2);
```

After:

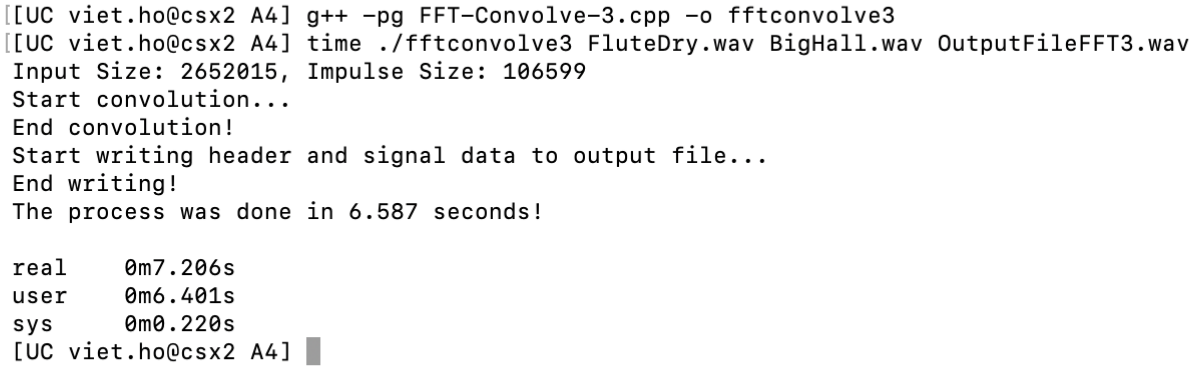


```
   // Avoiding Recomputation - Optimization 3
   int maxLength = powerOfTwo * 2;

   double *freq_input_signal = new double[maxLength];
```



```
  double *freq_IR_signal = new double[maxLength];
  double *freq_output_signal = new double[maxLength];

  for (int i = 0; i < maxLength; i += 2)
  {
      ...
  }

convolve(freq_input_signal, freq_IR_signal, freq_output_signal, maxLength);
```

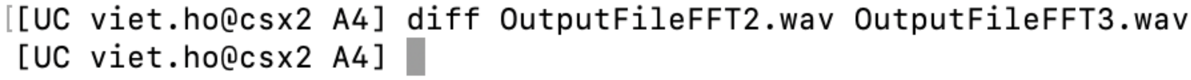
Result:



```
[[UC viet.ho@csx2 A4] g++ -pg FFT-Convolve-3.cpp -o fftconvolve3
[[UC viet.ho@csx2 A4] time ./fftconvolve3 FluteDry.wav BigHall.wav OutputFileFFT3.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 6.587 seconds!

 real    0m7.206s
 user    0m6.401s
 sys     0m0.220s
 [UC viet.ho@csx2 A4] 
```

Program Version	Time Measurement (seconds)	Time Reduction
FFT-Optimization-2 (FFT-Convolve-2.cpp)	7.887	N/a
FFT-Optimization-3 (FFT-Convolve-3.cpp)	6.587	16.48%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileFFT2.wav OutputFileFFT3.wav). There are no differences between them so the diff command doesn't produce an output.

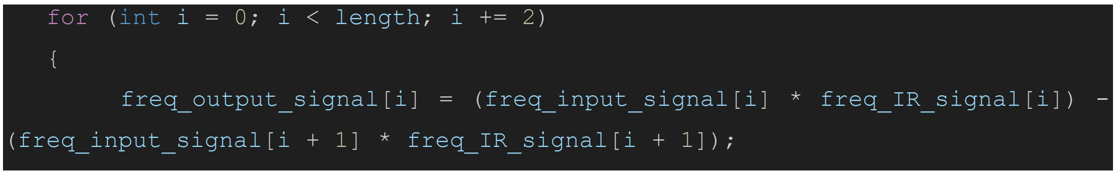


```
[[UC viet.ho@csx2 A4] diff OutputFileFFT2.wav OutputFileFFT3.wav
 [UC viet.ho@csx2 A4] 
```

6. Fourth Optimization Program - Optimizing Memory Access

Optimizing memory access involves minimizing cache misses, utilizing data locality, and aligning data structures to improve the efficiency of accessing and retrieving information from the computer's memory. The program was invoked from the command line as follows: FFT-Convolve-4 FluteDry.wav BigHall.wav OutputFileFFT4.wav.

Before:



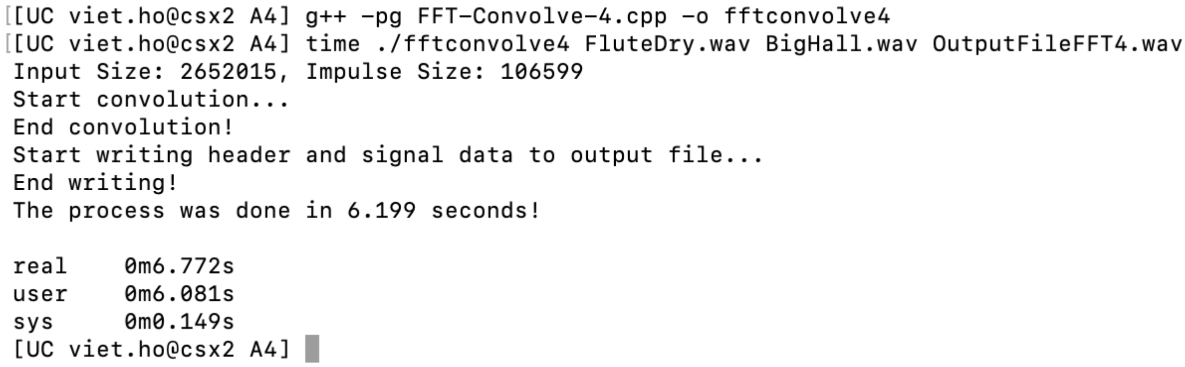
```
  for (int i = 0; i < length; i += 2)
  {
       freq_output_signal[i] = (freq_input_signal[i] * freq_IR_signal[i]) -
(freq_input_signal[i + 1] * freq_IR_signal[i + 1]);
```

```
                freq_output_signal[i + 1] = (freq_input_signal[i + 1] *
freq_IR_signal[i]) + (freq_input_signal[i] * freq_IR_signal[i + 1]);
    }
```

After:

```
    // Optimizing Memory Access - Optimization 4
    for (int i = 0; i < length; i += 2)
    {
        double inputReal = freq_input_signal[i];
        double inputImag = freq_input_signal[i + 1];
        double irReal = freq_IR_signal[i];
        double irImag = freq_IR_signal[i + 1];

        freq_output_signal[i] = (inputReal * irReal) - (inputImag * irImag);
            freq_output_signal[i + 1] = (inputImag * irReal) + (inputReal *
irImag);
    }
```

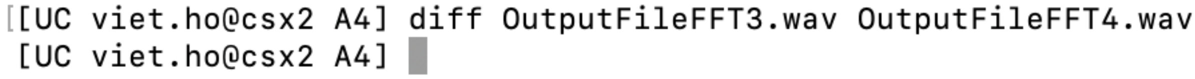
Result:



```
[[UC viet.ho@csx2 A4] g++ -pg FFT-Convolve-4.cpp -o fftconvolve4
[[UC viet.ho@csx2 A4] time ./fftconvolve4 FluteDry.wav BigHall.wav OutputFileFFT4.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 6.199 seconds!

 real    0m6.772s
 user    0m6.081s
 sys     0m0.149s
 [UC viet.ho@csx2 A4]
```

Program Version	Time Measurement (seconds)	Time Reduction
FFT-Optimization-3 (FFT-Convolve-3.cpp)	6.587	N/a
FFT-Optimization-4 (FFT-Convolve-4.cpp)	6.199	5.89%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileFFT3.wav OutputFileFFT4.wav). There are no differences between them so the diff command doesn't produce an output.

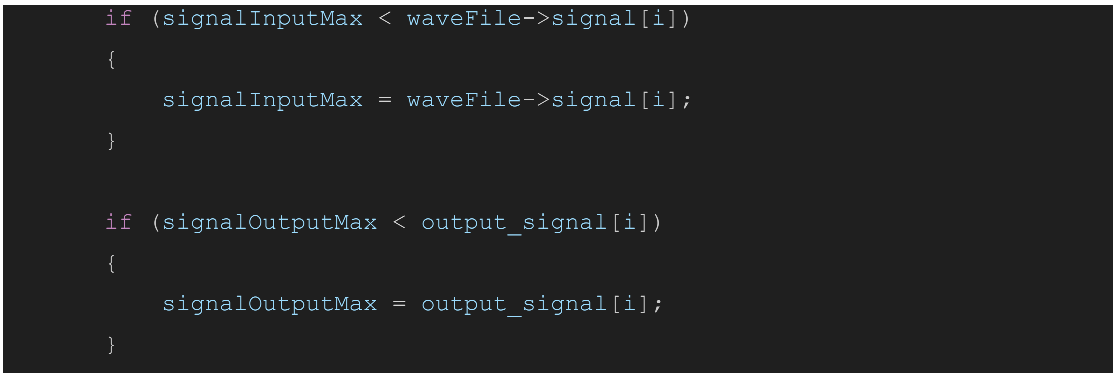


```
[[UC viet.ho@csx2 A4] diff OutputFileFFT3.wav OutputFileFFT4.wav
 [UC viet.ho@csx2 A4]
```

7. **Fifth Optimization Program - Optimizing Conditional Statements**
   Optimizing conditional statements involves reordering conditions based on their likelihood, simplifying complex expressions, and utilizing branch prediction to enhance the efficiency of decision-making in code execution.

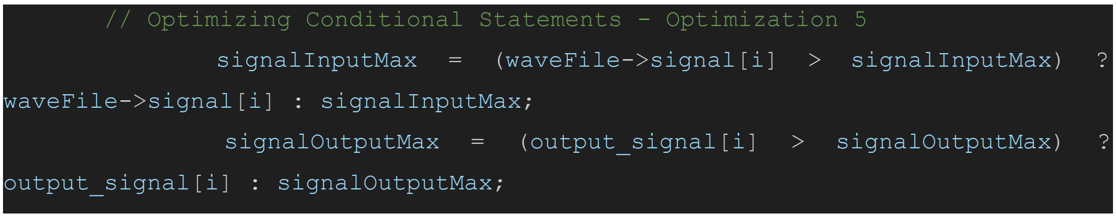
The program was invoked from the command line as follows: FFT-Convolve-5 FluteDry.wav BigHall.wav OutputFileFFT5.wav.

Before:



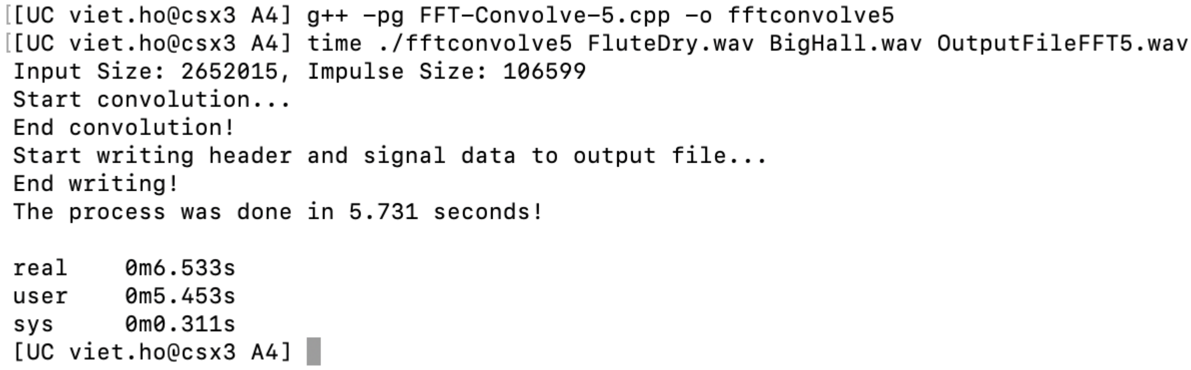
```
        if (signalInputMax < waveFile->signal[i])
        {
            signalInputMax = waveFile->signal[i];
        }

        if (signalOutputMax < output_signal[i])
        {
            signalOutputMax = output_signal[i];
        }
```

After:



```
        // Optimizing Conditional Statements - Optimization 5
                signalInputMax = (waveFile->signal[i] > signalInputMax) ?
waveFile->signal[i] : signalInputMax;
                signalOutputMax = (output_signal[i] > signalOutputMax) ?
output_signal[i] : signalOutputMax;
```

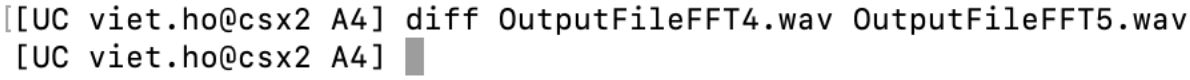
Result:



```
[[UC viet.ho@csx3 A4] g++ -pg FFT-Convolve-5.cpp -o fftconvolve5
[[UC viet.ho@csx3 A4] time ./fftconvolve5 FluteDry.wav BigHall.wav OutputFileFFT5.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 5.731 seconds!

 real    0m6.533s
 user    0m5.453s
 sys     0m0.311s
 [UC viet.ho@csx3 A4] 
```

Program Version	Time Measurement (seconds)	Time Reduction
FFT-Optimization-4 (FFT-Convolve-4.cpp)	6.199	N/a
FFT-Optimization-5 (FFT-Convolve-5.cpp)	5.731	7.55%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileFFT4.wav OutputFileFFT5.wav). There are no differences between them so the diff command doesn't produce an output.

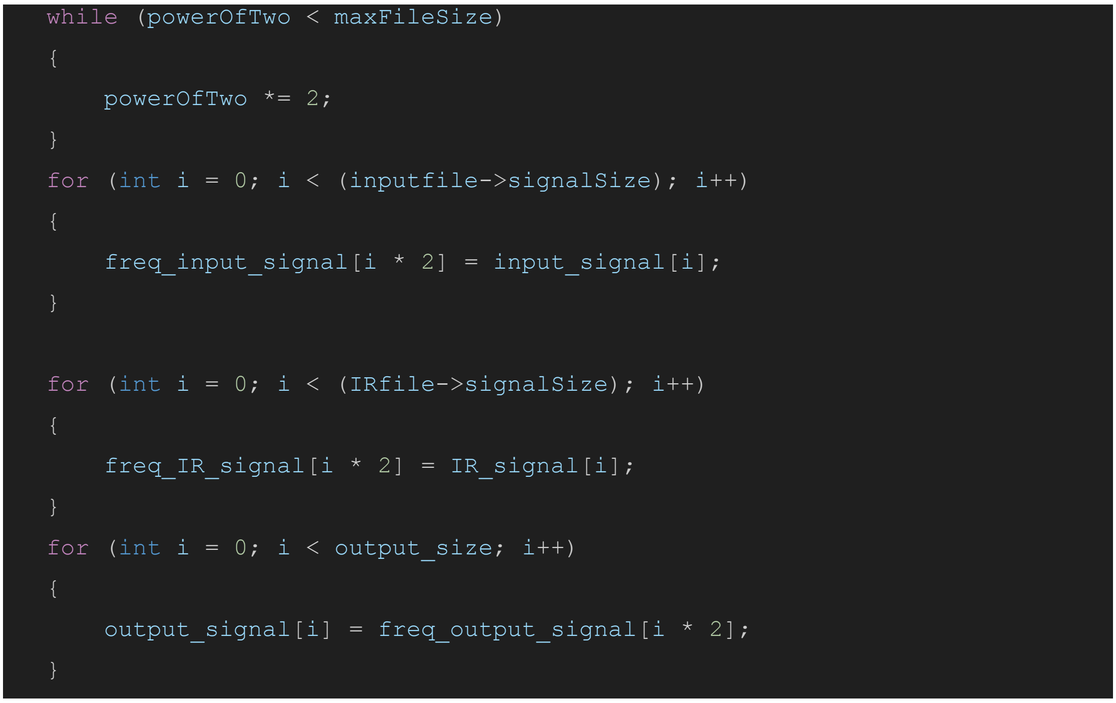


```
[[UC viet.ho@csx2 A4] diff OutputFileFFT4.wav OutputFileFFT5.wav
 [UC viet.ho@csx2 A4] 
```

8. Sixth Optimization Program - Strength Reduction

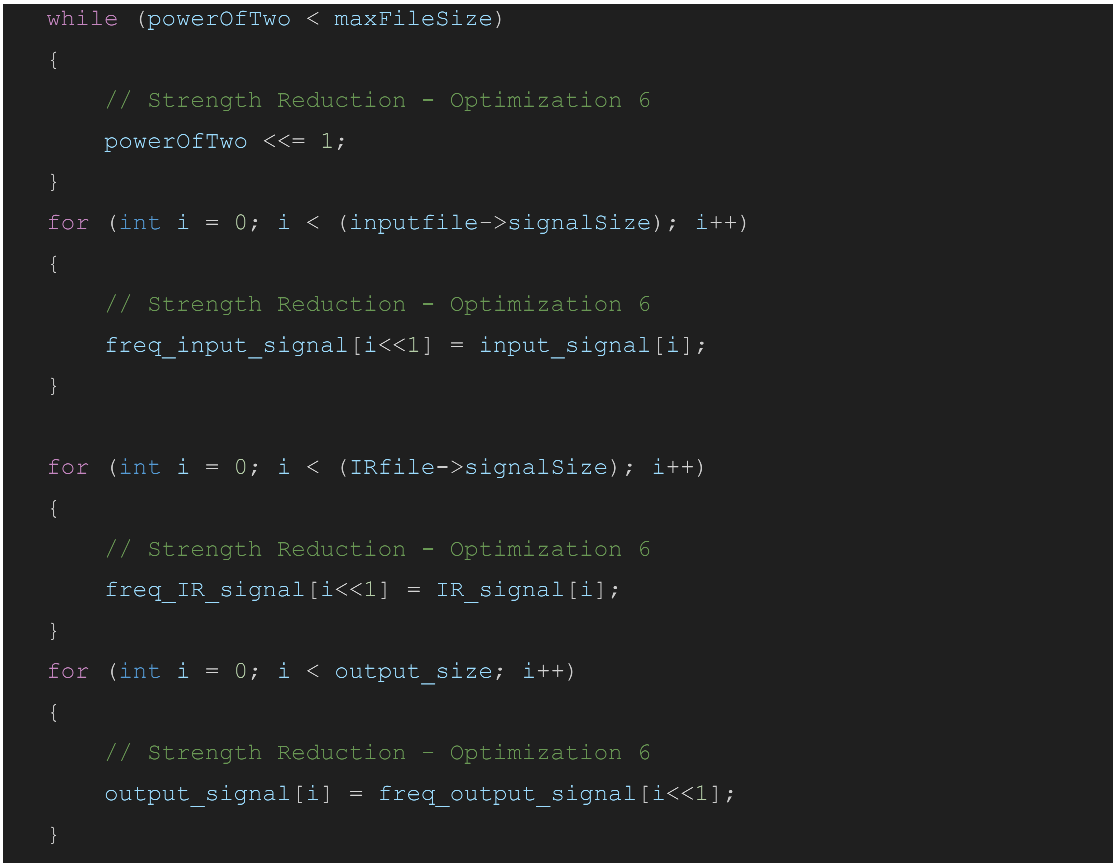
Strength reduction is an optimization technique that replaces expensive operations, such as multiplication, with equivalent but less computationally intensive operations, like shifting, to improve code performance. The program was invoked from the command line as follows: FFT-Convolve-6 FluteDry.wav BigHall.wav OutputFileFFT6.wav.

Before:



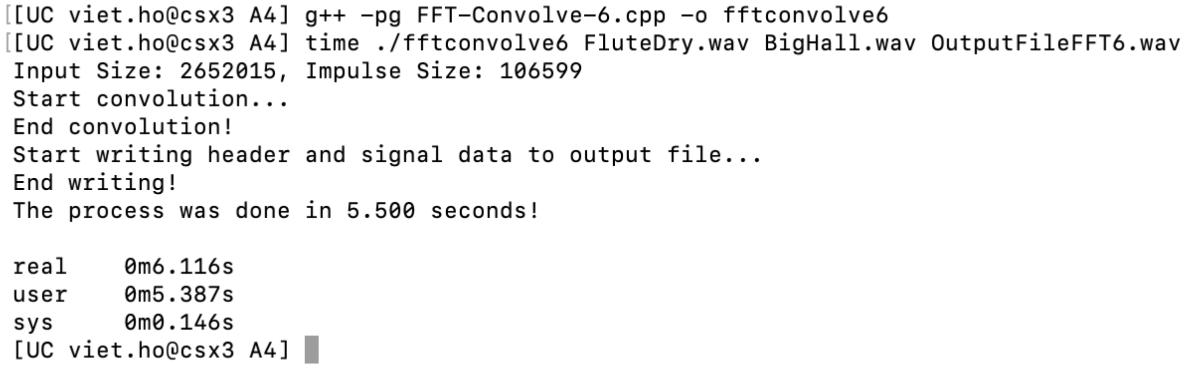
```
while (powerOfTwo < maxFileSize)
{
    powerOfTwo *= 2;
}
for (int i = 0; i < (inputfile->signalSize); i++)
{
    freq_input_signal[i * 2] = input_signal[i];
}

for (int i = 0; i < (IRfile->signalSize); i++)
{
    freq_IR_signal[i * 2] = IR_signal[i];
}
for (int i = 0; i < output_size; i++)
{
    output_signal[i] = freq_output_signal[i * 2];
}
```

After:



```
while (powerOfTwo < maxFileSize)
{
    // Strength Reduction - Optimization 6
    powerOfTwo <<= 1;
}
for (int i = 0; i < (inputfile->signalSize); i++)
{
    // Strength Reduction - Optimization 6
    freq_input_signal[i<<1] = input_signal[i];
}

for (int i = 0; i < (IRfile->signalSize); i++)
{
    // Strength Reduction - Optimization 6
    freq_IR_signal[i<<1] = IR_signal[i];
}
for (int i = 0; i < output_size; i++)
{
    // Strength Reduction - Optimization 6
    output_signal[i] = freq_output_signal[i<<1];
}
```

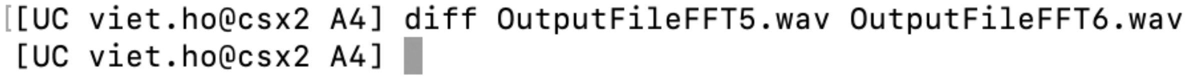
Result:



```
[[UC viet.ho@csx3 A4] g++ -pg FFT-Convolve-6.cpp -o fftconvolve6
[[UC viet.ho@csx3 A4] time ./fftconvolve6 FluteDry.wav BigHall.wav OutputFileFFT6.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 5.500 seconds!

 real    0m6.116s
 user    0m5.387s
 sys     0m0.146s
 [UC viet.ho@csx3 A4]
```

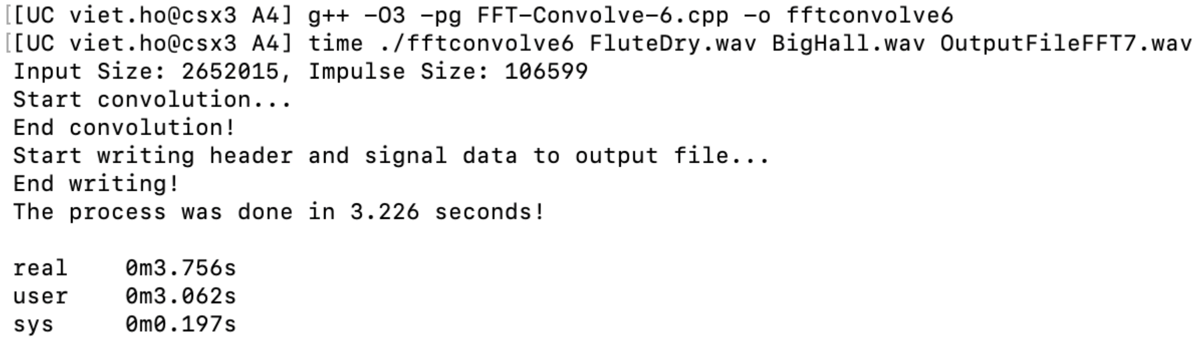
Program Version	Time Measurement (seconds)	Time Reduction
FFT-Optimization-5 (FFT-Convolve-5.cpp)	5.731	N/a
FFT-Optimization-6 (FFT-Convolve-6.cpp)	5.500	4.03%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileFFT5.wav OutputFileFFT6.wav). There are no differences between them so the diff command doesn't produce an output.



```
[[UC viet.ho@csx2 A4] diff OutputFileFFT5.wav OutputFileFFT6.wav
 [UC viet.ho@csx2 A4]
```

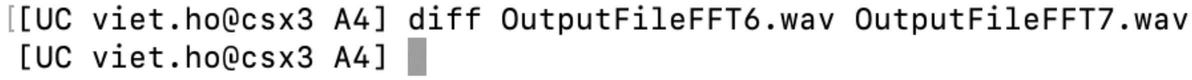
9. **Compiler Optimization - Compiler Flag -O3**
The -O3 compiler flag activates the highest level of optimization, enabling aggressive optimizations to improve performance, potentially at the expense of larger executable size and longer compilation times.



```
[[UC viet.ho@csx3 A4] g++ -O3 -pg FFT-Convolve-6.cpp -o fftconvolve6
[[UC viet.ho@csx3 A4] time ./fftconvolve6 FluteDry.wav BigHall.wav OutputFileFFT7.wav
 Input Size: 2652015, Impulse Size: 106599
 Start convolution...
 End convolution!
 Start writing header and signal data to output file...
 End writing!
 The process was done in 3.226 seconds!

 real    0m3.756s
 user    0m3.062s
 sys     0m0.197s
```

Program Version	Time Measurement (seconds)	Time Reduction
FFT-Optimization-6 (FFT-Convolve-6.cpp) (compiled without -O3)	5.500	N/a
FFT-Optimization-6 (FFT-Convolve-6.cpp) (compiled with -O3)	3.226	41.35%

For regression tests, I have used diff command-line to compare two files line by line (diff OutputFileFFT6.wav OutputFileFFT7.wav). There are no differences between them so the diff command doesn't produce an output.



```
[[UC viet.ho@csx3 A4] diff OutputFileFFT6.wav OutputFileFFT7.wav
 [UC viet.ho@csx3 A4] █
```

Profiler Report:

1. **Baseline Program**

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.99    792.33   792.33        1   792.33   792.33  convolve(double*, int, double*, int,
double*, int)
  0.00    792.36     0.03  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.00    792.38     0.02        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.00    792.38     0.01        2     0.01     0.01  WAVEFile::dataToSignal()
  0.00    792.39     0.01        1     0.01   792.40  createOutputFile(char*)
   0.00     792.40       0.01          1       0.01      0.01   adjustOutputSignal(WAVEFile*,
double*, int)
  0.00    792.41     0.01        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00    792.41     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00    792.41     0.00        1     0.00     0.01  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)
     0.00            792.41              0.00                    1              0.00              0.00
__static_initialization_and_destruction_0()

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
              else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
              function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
              the function in the gprof listing. If the index is
              in parenthesis it shows where it would appear in
              the gprof listing if it were to be printed.
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
                         Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.00% of 792.41 seconds
```

```
index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00  792.41                  main [1]
                0.01  792.39      1/1          createOutputFile(char*) [2]
                0.00    0.01      2/2        WAVEFile::readWAVEFile(char const*) [7]
-----------------------------------------------
                0.01  792.39      1/1          main [1]
[2]    100.0    0.01  792.39       1         createOutputFile(char*) [2]
                792.33    0.00      1/1          convolve(double*, int, double*, int, double*, int)
[3]
                0.02    0.00 2758613/2758617     fwriteShortLSB(short, _IO_FILE*) [4]
                0.02    0.00      2/2         shortToDouble(WAVEFile*, double*) [5]
                0.01    0.00      1/1         adjustOutputSignal(WAVEFile*, double*, int) [8]
                 0.00    0.01      1/1          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[9]
-----------------------------------------------
              792.33    0.00      1/1          createOutputFile(char*) [2]
[3]     100.0  792.33    0.00      1        convolve(double*, int, double*, int, double*, int)
[3]
-----------------------------------------------
                    0.00      0.00        4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [9]
                0.02    0.00 2758613/2758617     createOutputFile(char*) [2]
[4]      0.0    0.03    0.00 2758617        fwriteShortLSB(short, _IO_FILE*) [4]
-----------------------------------------------
                0.02    0.00      2/2         createOutputFile(char*) [2]
[5]      0.0    0.02    0.00      2        shortToDouble(WAVEFile*, double*) [5]
-----------------------------------------------
                0.01    0.00      2/2         WAVEFile::readWAVEFile(char const*) [7]
[6]      0.0    0.01    0.00      2        WAVEFile::dataToSignal() [6]
-----------------------------------------------
                0.00    0.01      2/2         main [1]
[7]      0.0    0.00    0.01      2        WAVEFile::readWAVEFile(char const*) [7]
                0.01    0.00      2/2         WAVEFile::dataToSignal() [6]
-----------------------------------------------
                0.01    0.00      1/1         createOutputFile(char*) [2]
[8]      0.0    0.01    0.00      1        adjustOutputSignal(WAVEFile*, double*, int) [8]
-----------------------------------------------
                0.00    0.01      1/1         createOutputFile(char*) [2]
[9]       0.0     0.00    0.01      1        writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[9]
                0.01    0.00      5/5         fwriteIntLSB(int, _IO_FILE*) [10]
                0.00    0.00      4/2758617     fwriteShortLSB(short, _IO_FILE*) [4]
-----------------------------------------------
                 0.01    0.00      5/5          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[9]
[10]     0.0    0.01    0.00      5        fwriteIntLSB(int, _IO_FILE*) [10]
-----------------------------------------------
                0.00    0.00      1/1         _GLOBAL__sub_I_inputfile [18]
[17]     0.0    0.00    0.00      1        __static_initialization_and_destruction_0() [17]
-----------------------------------------------
```

This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.

This line lists:

index A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child `/' the total number of times the child

```
                 was called.  Recursive calls by the child are not
                 listed in the number after the `/'.

     name        This is the name of the child.  The child's index
                 number is printed after it.  If the child is a
                 member of a cycle, the cycle number is printed
                 between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
Index by function name

  [10] fwriteIntLSB(int, _IO_FILE*) [8] adjustOutputSignal(WAVEFile*, double*, int) [6]
WAVEFile::dataToSignal()
   [5] shortToDouble(WAVEFile*, double*) [9] writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [7] WAVEFile::readWAVEFile(char const*)
                      [4]      fwriteShortLSB(short,      _IO_FILE*)      [17]
__static_initialization_and_destruction_0()
  [2] createOutputFile(char*) [3] convolve(double*, int, double*, int, double*, int)
```

2. **Algorithm-Based Optimization Program**

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 97.82      9.88     9.88        3     3.29     3.29  four1(double*, unsigned long, int)
  1.09      9.99     0.11        1     0.11    10.08  createOutputFile(char*)
  0.50     10.04     0.05        1     0.05     0.05  convolve(double*, double*, double*, int)
  0.20     10.06     0.02        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.20     10.08     0.02        2     0.01     0.01  WAVEFile::dataToSignal()
  0.20     10.10     0.02        1     0.02     0.02  adjustOutputSignal(WAVEFile*, double*,
int)
  0.00     10.10     0.00  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.00     10.10     0.00        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00     10.10     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00     10.10     0.00        1     0.00     0.00  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)
     0.00          10.10          0.00               1          0.00          0.00
__static_initialization_and_destruction_0()

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
```

```
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.10% of 10.10 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00   10.10                 main [1]
                0.11    9.97       1/1           createOutputFile(char*) [2]
                0.00    0.02       2/2           WAVEFile::readWAVEFile(char const*) [7]
-----------------------------------------------
                0.11    9.97       1/1           main [1]
[2]     99.8    0.11    9.97       1         createOutputFile(char*) [2]
                9.88    0.00       3/3           four1(double*, unsigned long, int) [3]
                0.05    0.00       1/1           convolve(double*, double*, double*, int) [4]
                0.02    0.00       2/2           shortToDouble(WAVEFile*, double*) [5]
                0.02    0.00       1/1           adjustOutputSignal(WAVEFile*, double*, int) [8]
                0.00    0.00 2758613/2758617     fwriteShortLSB(short, _IO_FILE*) [15]
                0.00    0.00       1/1           writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[17]
-----------------------------------------------
                9.88    0.00       3/3           createOutputFile(char*) [2]
[3]     97.8    9.88    0.00       3         four1(double*, unsigned long, int) [3]
-----------------------------------------------
                0.05    0.00       1/1           createOutputFile(char*) [2]
[4]      0.5    0.05    0.00       1         convolve(double*, double*, double*, int) [4]
-----------------------------------------------
                0.02    0.00       2/2           createOutputFile(char*) [2]
[5]      0.2    0.02    0.00       2         shortToDouble(WAVEFile*, double*) [5]
-----------------------------------------------
                0.02    0.00       2/2           WAVEFile::readWAVEFile(char const*) [7]
[6]      0.2    0.02    0.00       2         WAVEFile::dataToSignal() [6]
-----------------------------------------------
                0.00    0.02       2/2           main [1]
[7]      0.2    0.00    0.02       2         WAVEFile::readWAVEFile(char const*) [7]
                0.02    0.00       2/2           WAVEFile::dataToSignal() [6]
```

```
-----------------------------------------------
            0.02    0.00       1/1          createOutputFile(char*) [2]
[8]      0.2    0.02    0.00       1        adjustOutputSignal(WAVEFile*, double*, int) [8]
-----------------------------------------------
                  0.00      0.00        4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [17]
            0.00    0.00 2758613/2758617     createOutputFile(char*) [2]
[15]     0.0    0.00    0.00 2758617        fwriteShortLSB(short, _IO_FILE*) [15]
-----------------------------------------------
             0.00    0.00       5/5          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[17]
[16]     0.0    0.00    0.00       5        fwriteIntLSB(int, _IO_FILE*) [16]
-----------------------------------------------
            0.00    0.00       1/1          createOutputFile(char*) [2]
[17]      0.0     0.00    0.00       1        writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[17]
            0.00    0.00       5/5          fwriteIntLSB(int, _IO_FILE*) [16]
            0.00    0.00       4/2758617     fwriteShortLSB(short, _IO_FILE*) [15]
-----------------------------------------------
            0.00    0.00       1/1          _GLOBAL__sub_I_inputfile [19]
[18]     0.0    0.00    0.00       1        __static_initialization_and_destruction_0() [18]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
                The index number is printed next to every function name so
                it is easier to look up where the function is in the table.

     % time     This is the percentage of the `total' time that was spent
                in this function and its children.  Note that due to
                different viewpoints, functions excluded by options, etc,
                these numbers will NOT add up to 100%.

     self This is the total amount of time spent in this function.

     children   This is the total amount of time propagated into this
                function by its children.

     called     This is the number of times the function was called.
                If the function called itself recursively, the number
                only includes non-recursive calls, and is followed by
                a `+' and the number of recursive calls.

     name       The name of the current function.  The index number is
                printed after it.  If the function is a member of a
                cycle, the cycle number is printed between the
                function's name and the index number.


 For the function's parents, the fields have the following meanings:
```

self This is the amount of time that was propagated directly
from the function into this parent.

children This is the amount of time that was propagated from
the function's children into this parent.

called This is the number of times this parent called the
function `/' the total number of times the function
was called. Recursive calls to the function are not
included in the number after the `/'.

name This is the name of the parent. The parent's index
number is printed after it. If the parent is a
member of a cycle, the cycle number is printed between
the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly
from the child into the function.

children This is the amount of time that was propagated from the
child's children to the function.

called This is the number of times the function called
this child `/' the total number of times the child
was called. Recursive calls by the child are not
listed in the number after the `/'.

name This is the name of the child. The child's index
number is printed after it. If the child is a
member of a cycle, the cycle number is printed
between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The `+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Index by function name

[16] fwriteIntLSB(int, _IO_FILE*) [8] adjustOutputSignal(WAVEFile*, double*, int) [4] convolve(double*, double*, double*, int)
[5] shortToDouble(WAVEFile*, double*) [17] writeWAVEFileHeader(int, int, int, int, _IO_FILE*) [6] WAVEFile::dataToSignal()
[15] fwriteShortLSB(short, _IO_FILE*) [18] __static_initialization_and_destruction_0() [7] WAVEFile::readWAVEFile(char const*)
[2] createOutputFile(char*) [3] four1(double*, unsigned long, int)

3. First Optimization Program - Loop Unrolling

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 97.51      8.24     8.24        3     2.75     2.75  four1(double*, unsigned long, int)
  0.95      8.32     0.08        1     0.08     8.44  createOutputFile(char*)
  0.59      8.37     0.05        1     0.05     0.05  convolve(double*, double*, double*, int)
  0.30      8.39     0.03  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.30      8.42     0.03        1     0.03     0.03  adjustOutputSignal(WAVEFile*, double*,
int)
  0.24      8.44     0.02        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.12      8.45     0.01        2     0.01     0.01  WAVEFile::dataToSignal()
  0.00      8.45     0.00        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00      8.45     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00      8.45     0.00        1     0.00     0.00  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)
  0.00      8.45     0.00        1     0.00     0.00
__static_initialization_and_destruction_0()

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.12% of 8.45 seconds

index % time    self  children    called     name
```

```
                                           <spontaneous>
[1]    100.0    0.00    8.45              main [1]
                0.08    8.36       1/1          createOutputFile(char*) [2]
                0.00    0.01       2/2          WAVEFile::readWAVEFile(char const*) [9]
-----------------------------------------------
                0.08    8.36       1/1          main [1]
[2]     99.9    0.08    8.36       1        createOutputFile(char*) [2]
                8.24    0.00       3/3          four1(double*, unsigned long, int) [3]
                0.05    0.00       1/1          convolve(double*, double*, double*, int) [4]
                0.03    0.00       1/1          adjustOutputSignal(WAVEFile*, double*, int) [6]
                0.02    0.00 2758613/2758617     fwriteShortLSB(short, _IO_FILE*) [5]
                0.02    0.00       2/2          shortToDouble(WAVEFile*, double*) [7]
                 0.00    0.00       1/1          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
-----------------------------------------------
                8.24    0.00       3/3          createOutputFile(char*) [2]
[3]     97.5    8.24    0.00       3        four1(double*, unsigned long, int) [3]
-----------------------------------------------
                0.05    0.00       1/1          createOutputFile(char*) [2]
[4]      0.6    0.05    0.00       1       convolve(double*, double*, double*, int) [4]
-----------------------------------------------
                        0.00      0.00        4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [10]
                0.02    0.00 2758613/2758617     createOutputFile(char*) [2]
[5]      0.3    0.03    0.00 2758617        fwriteShortLSB(short, _IO_FILE*) [5]
-----------------------------------------------
                0.03    0.00       1/1          createOutputFile(char*) [2]
[6]      0.3    0.03    0.00       1       adjustOutputSignal(WAVEFile*, double*, int) [6]
-----------------------------------------------
                0.02    0.00       2/2          createOutputFile(char*) [2]
[7]      0.2    0.02    0.00       2       shortToDouble(WAVEFile*, double*) [7]
-----------------------------------------------
                0.01    0.00       2/2          WAVEFile::readWAVEFile(char const*) [9]
[8]      0.1    0.01    0.00       2       WAVEFile::dataToSignal() [8]
-----------------------------------------------
                0.00    0.01       2/2          main [1]
[9]      0.1    0.00    0.01       2       WAVEFile::readWAVEFile(char const*) [9]
                0.01    0.00       2/2          WAVEFile::dataToSignal() [8]
-----------------------------------------------
                0.00    0.00       1/1          createOutputFile(char*) [2]
[10]      0.0    0.00    0.00       1        writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
                0.00    0.00       4/2758617     fwriteShortLSB(short, _IO_FILE*) [5]
                0.00    0.00       5/5          fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
                 0.00    0.00       5/5          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
[17]     0.0    0.00    0.00       5        fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
                0.00    0.00       1/1          _GLOBAL__sub_I_inputfile [19]
[18]     0.0    0.00    0.00       1        __static_initialization_and_destruction_0() [18]
-----------------------------------------------
```

This table describes the call tree of the program, and was sorted by the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the index number at the left hand margin lists the current function. The lines above it list the functions that called this function,

and the lines below it list the functions this one called.
This line lists:

index A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called

```
                this child `/' the total number of times the child
                was called.  Recursive calls by the child are not
                listed in the number after the `/'.

    name        This is the name of the child.  The child's index
                number is printed after it.  If the child is a
                member of a cycle, the cycle number is printed
                between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
Index by function name

  [17] fwriteIntLSB(int, _IO_FILE*) [6] adjustOutputSignal(WAVEFile*, double*, int) [4]
convolve(double*, double*, double*, int)
   [7] shortToDouble(WAVEFile*, double*) [10] writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [8] WAVEFile::dataToSignal()
                          [5]         fwriteShortLSB(short,          _IO_FILE*)          [18]
__static_initialization_and_destruction_0() [9] WAVEFile::readWAVEFile(char const*)
  [2] createOutputFile(char*) [3] four1(double*, unsigned long, int)
```

4. **<u>Second Optimization Program - Inline Functions</u>**

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 96.75      7.44     7.44        3     2.48     2.48  four1(double*, unsigned long, int)
  1.82      7.58     0.14        1     0.14     7.68  createOutputFile(char*)
  0.65      7.63     0.05        1     0.05     0.05  convolve(double*, double*, double*, int)
  0.26      7.65     0.02        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.20      7.67     0.01  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.13      7.67     0.01        2     0.01     0.01  WAVEFile::dataToSignal()
  0.13      7.68     0.01        1     0.01     0.01  adjustOutputSignal(WAVEFile*, double*,
int)
  0.07      7.69     0.01        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00      7.69     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00      7.69     0.00        1     0.00     0.01  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.
 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
          listing.

calls      the number of times this function was invoked, if
          this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
             else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
             function is profiled, else blank.

name       the name of the function.  This is the minor sort
          for this listing. The index shows the location of
             the function in the gprof listing. If the index is
             in parenthesis it shows where it would appear in
             the gprof listing if it were to be printed.
```

```
                        Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.13% of 7.69 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00    7.69                main [1]
             0.14    7.54       1/1          createOutputFile(char*) [2]
             0.00    0.01       2/2          WAVEFile::readWAVEFile(char const*) [8]
-----------------------------------------------
             0.14    7.54       1/1          main [1]
[2]     99.9    0.14    7.54       1        createOutputFile(char*) [2]
             7.44    0.00       3/3          four1(double*, unsigned long, int) [3]
             0.05    0.00       1/1          convolve(double*, double*, double*, int) [4]
             0.02    0.00       2/2          shortToDouble(WAVEFile*, double*) [5]
             0.01    0.00 2758613/2758617      fwriteShortLSB(short, _IO_FILE*) [6]
             0.01    0.00       1/1          adjustOutputSignal(WAVEFile*, double*, int) [9]
              0.00    0.01       1/1          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
-----------------------------------------------
             7.44    0.00       3/3          createOutputFile(char*) [2]
[3]     96.7    7.44    0.00       3        four1(double*, unsigned long, int) [3]
-----------------------------------------------
             0.05    0.00       1/1          createOutputFile(char*) [2]
```

```
[4]     0.7    0.05    0.00       1         convolve(double*, double*, double*, int) [4]
-----------------------------------------------
                0.02    0.00       2/2          createOutputFile(char*) [2]
[5]     0.3    0.02    0.00       2         shortToDouble(WAVEFile*, double*) [5]
-----------------------------------------------
                        0.00      0.00        4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [10]
                0.01    0.00 2758613/2758617     createOutputFile(char*) [2]
[6]     0.2    0.01    0.00 2758617         fwriteShortLSB(short, _IO_FILE*) [6]
-----------------------------------------------
                0.01    0.00       2/2          WAVEFile::readWAVEFile(char const*) [8]
[7]     0.1    0.01    0.00       2         WAVEFile::dataToSignal() [7]
-----------------------------------------------
                0.00    0.01       2/2          main [1]
[8]     0.1    0.00    0.01       2         WAVEFile::readWAVEFile(char const*) [8]
                0.01    0.00       2/2          WAVEFile::dataToSignal() [7]
-----------------------------------------------
                0.01    0.00       1/1          createOutputFile(char*) [2]
[9]     0.1    0.01    0.00       1         adjustOutputSignal(WAVEFile*, double*, int) [9]
-----------------------------------------------
                0.00    0.01       1/1          createOutputFile(char*) [2]
[10]     0.1    0.00    0.01       1         writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
                0.01    0.00       5/5          fwriteIntLSB(int, _IO_FILE*) [11]
                0.00    0.00       4/2758617      fwriteShortLSB(short, _IO_FILE*) [6]
-----------------------------------------------
                 0.01    0.00       5/5          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
[11]     0.1    0.01    0.00       5         fwriteIntLSB(int, _IO_FILE*) [11]
-----------------------------------------------
                                   1           __static_initialization_and_destruction_0() [19]
[19]     0.0    0.00    0.00      0+1       __static_initialization_and_destruction_0() [19]
                                   1           __static_initialization_and_destruction_0() [19]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
                The index number is printed next to every function name so
                it is easier to look up where the function is in the table.

     % time     This is the percentage of the `total' time that was spent
                in this function and its children.  Note that due to
                different viewpoints, functions excluded by options, etc,
```

these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.

name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The `+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Index by function name

```
  [11] fwriteIntLSB(int, _IO_FILE*) [9] adjustOutputSignal(WAVEFile*, double*, int) [7]
WAVEFile::dataToSignal()
    [5] shortToDouble(WAVEFile*, double*) [10] writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [8] WAVEFile::readWAVEFile(char const*)
   [6] fwriteShortLSB(short, _IO_FILE*) [3] four1(double*, unsigned long, int)
   [2] createOutputFile(char*) [4] convolve(double*, double*, double*, int)
```

5. Third Optimization Program - Avoiding Recomputation

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 96.56      6.18     6.18        3     2.06     2.06  four1(double*, unsigned long, int)
  1.56      6.28     0.10        1     0.10     6.39  createOutputFile(char*)
  0.78      6.33     0.05        1     0.05     0.05  convolve(double*, double*, double*, int)
  0.47      6.36     0.03  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.31      6.38     0.02        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.16      6.39     0.01        2     0.01     0.01  WAVEFile::dataToSignal()
  0.16      6.40     0.01        1     0.01     0.01  adjustOutputSignal(WAVEFile*, double*,
int)
  0.00      6.40     0.00        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00      6.40     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00      6.40     0.00        1     0.00     0.00  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)
  0.00      6.40     0.00        1     0.00     0.00
__static_initialization_and_destruction_0()
```

 %   the percentage of the total running time of the
time   program used by this function.

```
cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
               else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
               function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
               the function in the gprof listing. If the index is
               in parenthesis it shows where it would appear in
               the gprof listing if it were to be printed.
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
                    Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.16% of 6.40 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00    6.40                 main [1]
                0.10    6.29       1/1          createOutputFile(char*) [2]
                0.00    0.01       2/2          WAVEFile::readWAVEFile(char const*) [8]
-----------------------------------------------
                0.10    6.29       1/1          main [1]
[2]     99.8    0.10    6.29       1         createOutputFile(char*) [2]
                6.18    0.00       3/3          four1(double*, unsigned long, int) [3]
                0.05    0.00       1/1          convolve(double*, double*, double*, int) [4]
                0.03    0.00 2758613/2758617     fwriteShortLSB(short, _IO_FILE*) [5]
                0.02    0.00       2/2          shortToDouble(WAVEFile*, double*) [6]
                0.01    0.00       1/1          adjustOutputSignal(WAVEFile*, double*, int) [9]
                 0.00    0.00       1/1          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
-----------------------------------------------
                6.18    0.00       3/3          createOutputFile(char*) [2]
```

```
[3]    96.6    6.18    0.00       3         four1(double*, unsigned long, int) [3]
-----------------------------------------------
                0.05    0.00       1/1          createOutputFile(char*) [2]
[4]     0.8    0.05    0.00       1         convolve(double*, double*, double*, int) [4]
-----------------------------------------------
                       0.00      0.00        4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [10]
                0.03    0.00 2758613/2758617     createOutputFile(char*) [2]
[5]     0.5    0.03    0.00 2758617        fwriteShortLSB(short, _IO_FILE*) [5]
-----------------------------------------------
                0.02    0.00       2/2          createOutputFile(char*) [2]
[6]     0.3    0.02    0.00       2         shortToDouble(WAVEFile*, double*) [6]
-----------------------------------------------
                0.01    0.00       2/2          WAVEFile::readWAVEFile(char const*) [8]
[7]     0.2    0.01    0.00       2         WAVEFile::dataToSignal() [7]
-----------------------------------------------
                0.00    0.01       2/2          main [1]
[8]     0.2    0.00    0.01       2         WAVEFile::readWAVEFile(char const*) [8]
                0.01    0.00       2/2          WAVEFile::dataToSignal() [7]
-----------------------------------------------
                0.01    0.00       1/1          createOutputFile(char*) [2]
[9]     0.2    0.01    0.00       1         adjustOutputSignal(WAVEFile*, double*, int) [9]
-----------------------------------------------
                0.00    0.00       1/1          createOutputFile(char*) [2]
[10]     0.0     0.00    0.00       1         writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
                0.00    0.00       4/2758617      fwriteShortLSB(short, _IO_FILE*) [5]
                0.00    0.00       5/5          fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
                 0.00    0.00       5/5          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
[17]     0.0    0.00    0.00       5         fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
                0.00    0.00       1/1          _GLOBAL__sub_I_inputfile [19]
[18]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0() [18]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
                The index number is printed next to every function name so
                it is easier to look up where the function is in the table.

     % time     This is the percentage of the `total' time that was spent
```

in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called

```
                 this child `/' the total number of times the child
                 was called.  Recursive calls by the child are not
                 listed in the number after the `/'.

     name        This is the name of the child.  The child's index
                 number is printed after it.  If the child is a
                 member of a cycle, the cycle number is printed
                 between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
Index by function name

  [17] fwriteIntLSB(int, _IO_FILE*) [9] adjustOutputSignal(WAVEFile*, double*, int) [4]
convolve(double*, double*, double*, int)
   [6] shortToDouble(WAVEFile*, double*) [10] writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [7] WAVEFile::dataToSignal()
                   [5]         fwriteShortLSB(short,          _IO_FILE*)          [18]
__static_initialization_and_destruction_0() [8] WAVEFile::readWAVEFile(char const*)
  [2] createOutputFile(char*) [3] four1(double*, unsigned long, int)
```

6. Fourth Optimization Program - Optimizing Memory Access

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 96.51      5.81     5.81        3     1.94     1.94  four1(double*, unsigned long, int)
  1.83      5.92     0.11        1     0.11     6.00  createOutputFile(char*)
  0.83      5.97     0.05        1     0.05     0.05  convolve(double*, double*, double*, int)
  0.33      5.99     0.02        2     0.01     0.01  WAVEFile::dataToSignal()
  0.17      6.00     0.01  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.17      6.01     0.01        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.17      6.02     0.01        1     0.01     0.01  adjustOutputSignal(WAVEFile*, double*,
int)
  0.00      6.02     0.00        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00      6.02     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00      6.02     0.00        1     0.00     0.00  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)
    0.00            6.02          0.00              1          0.00         0.00
__static_initialization_and_destruction_0()
```

```
 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
              else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
              function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
              the function in the gprof listing. If the index is
              in parenthesis it shows where it would appear in
              the gprof listing if it were to be printed.
```

```
                         Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.17% of 6.02 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00    6.02                  main [1]
                0.11    5.89       1/1           createOutputFile(char*) [2]
                0.00    0.02       2/2           WAVEFile::readWAVEFile(char const*) [6]
-----------------------------------------------
                0.11    5.89       1/1           main [1]
[2]     99.7    0.11    5.89       1         createOutputFile(char*) [2]
                5.81    0.00       3/3           four1(double*, unsigned long, int) [3]
                0.05    0.00       1/1           convolve(double*, double*, double*, int) [4]
                0.01    0.00       2/2           shortToDouble(WAVEFile*, double*) [8]
                0.01    0.00       1/1           adjustOutputSignal(WAVEFile*, double*, int) [9]
                0.01    0.00 2758613/2758617     fwriteShortLSB(short, _IO_FILE*) [7]
                0.00    0.00       1/1          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
-----------------------------------------------
               5.81    0.00       3/3          createOutputFile(char*) [2]
[3]     96.5    5.81    0.00       3         four1(double*, unsigned long, int) [3]
-----------------------------------------------
               0.05    0.00       1/1          createOutputFile(char*) [2]
[4]      0.8    0.05    0.00       1         convolve(double*, double*, double*, int) [4]
-----------------------------------------------
               0.02    0.00       2/2          WAVEFile::readWAVEFile(char const*) [6]
[5]      0.3    0.02    0.00       2         WAVEFile::dataToSignal() [5]
-----------------------------------------------
               0.00    0.02       2/2          main [1]
[6]      0.3    0.00    0.02       2         WAVEFile::readWAVEFile(char const*) [6]
               0.02    0.00       2/2          WAVEFile::dataToSignal() [5]
-----------------------------------------------
                      0.00      0.00        4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [10]
               0.01    0.00 2758613/2758617     createOutputFile(char*) [2]
[7]      0.2    0.01    0.00 2758617         fwriteShortLSB(short, _IO_FILE*) [7]
-----------------------------------------------
               0.01    0.00       2/2          createOutputFile(char*) [2]
[8]      0.2    0.01    0.00       2         shortToDouble(WAVEFile*, double*) [8]
-----------------------------------------------
               0.01    0.00       1/1          createOutputFile(char*) [2]
[9]      0.2    0.01    0.00       1         adjustOutputSignal(WAVEFile*, double*, int) [9]
-----------------------------------------------
               0.00    0.00       1/1          createOutputFile(char*) [2]
[10]      0.0     0.00    0.00       1         writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
               0.00    0.00       4/2758617     fwriteShortLSB(short, _IO_FILE*) [7]
               0.00    0.00       5/5          fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
                0.00    0.00       5/5          writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
[17]     0.0    0.00    0.00       5         fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
               0.00    0.00       1/1          _GLOBAL__sub_I_inputfile [19]
[18]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0() [18]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
```

The index number is printed next to every function name so it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

```
children    This is the amount of time that was propagated from the
            child's children to the function.

called      This is the number of times the function called
            this child `/' the total number of times the child
            was called.  Recursive calls by the child are not
            listed in the number after the `/'.

name        This is the name of the child.  The child's index
            number is printed after it.  If the child is a
            member of a cycle, the cycle number is printed
            between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
```

Index by function name

```
  [17] fwriteIntLSB(int, _IO_FILE*) [9] adjustOutputSignal(WAVEFile*, double*, int) [4]
convolve(double*, double*, double*, int)
   [8] shortToDouble(WAVEFile*, double*) [10] writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [5] WAVEFile::dataToSignal()
                      [7]        fwriteShortLSB(short,        _IO_FILE*)        [18]
__static_initialization_and_destruction_0() [6] WAVEFile::readWAVEFile(char const*)
  [2] createOutputFile(char*) [3] four1(double*, unsigned long, int)
```

7. Fifth Optimization Program - Optimizing Conditional Statements

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 96.38      5.33     5.33        3     1.78     1.78  four1(double*, unsigned long, int)
  1.81      5.43     0.10        1     0.10     5.52  createOutputFile(char*)
  0.90      5.48     0.05        1     0.05     0.05  convolve(double*, double*, double*, int)
  0.36      5.50     0.02        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.18      5.51     0.01  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.18      5.52     0.01        2     0.01     0.01  WAVEFile::dataToSignal()
  0.18      5.53     0.01        1     0.01     0.01  adjustOutputSignal(WAVEFile*, double*,
int)
  0.00      5.53     0.00        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00      5.53     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00      5.53     0.00        1     0.00     0.00  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)
  0.00      5.53     0.00        1     0.00     0.00
__static_initialization_and_destruction_0()

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.18% of 5.53 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    5.53                 main [1]
                0.10    5.42       1/1           createOutputFile(char*) [2]
                0.00    0.01       2/2           WAVEFile::readWAVEFile(char const*) [8]
-----------------------------------------------
                0.10    5.42       1/1           main [1]
[2]     99.8    0.10    5.42       1         createOutputFile(char*) [2]
                5.33    0.00       3/3           four1(double*, unsigned long, int) [3]
                0.05    0.00       1/1           convolve(double*, double*, double*, int) [4]
                0.02    0.00       2/2           shortToDouble(WAVEFile*, double*) [5]
                0.01    0.00       1/1           adjustOutputSignal(WAVEFile*, double*, int) [9]
                0.01    0.00 2758613/2758617     fwriteShortLSB(short, _IO_FILE*) [6]
                 0.00    0.00       1/1           writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
-----------------------------------------------
                5.33    0.00       3/3           createOutputFile(char*) [2]
[3]     96.4    5.33    0.00       3         four1(double*, unsigned long, int) [3]
-----------------------------------------------
                0.05    0.00       1/1           createOutputFile(char*) [2]
[4]      0.9    0.05    0.00       1         convolve(double*, double*, double*, int) [4]
-----------------------------------------------
                0.02    0.00       2/2           createOutputFile(char*) [2]
[5]      0.4    0.02    0.00       2         shortToDouble(WAVEFile*, double*) [5]
-----------------------------------------------
                        0.00      0.00       4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [10]
                0.01    0.00 2758613/2758617     createOutputFile(char*) [2]
[6]      0.2    0.01    0.00 2758617         fwriteShortLSB(short, _IO_FILE*) [6]
-----------------------------------------------
                0.01    0.00       2/2           WAVEFile::readWAVEFile(char const*) [8]
[7]      0.2    0.01    0.00       2         WAVEFile::dataToSignal() [7]
-----------------------------------------------
                0.00    0.01       2/2           main [1]
[8]      0.2    0.00    0.01       2         WAVEFile::readWAVEFile(char const*) [8]
                0.01    0.00       2/2           WAVEFile::dataToSignal() [7]
-----------------------------------------------
                0.01    0.00       1/1           createOutputFile(char*) [2]
[9]      0.2    0.01    0.00       1         adjustOutputSignal(WAVEFile*, double*, int) [9]
-----------------------------------------------
                0.00    0.00       1/1           createOutputFile(char*) [2]
[10]     0.0    0.00    0.00       1         writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
                0.00    0.00       4/2758617     fwriteShortLSB(short, _IO_FILE*) [6]
                0.00    0.00       5/5           fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
                 0.00    0.00       5/5           writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[10]
[17]     0.0    0.00    0.00       5         fwriteIntLSB(int, _IO_FILE*) [17]
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I_inputfile [19]
[18]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0() [18]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
```

This line lists:

index A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

```
    self  This is the amount of time that was propagated directly
                from the child into the function.

    children    This is the amount of time that was propagated from the
                child's children to the function.

    called      This is the number of times the function called
                this child `/' the total number of times the child
                was called.  Recursive calls by the child are not
                listed in the number after the `/'.

    name        This is the name of the child.  The child's index
                number is printed after it.  If the child is a
                member of a cycle, the cycle number is printed
                between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
```

```
Index by function name

  [17] fwriteIntLSB(int, _IO_FILE*) [9] adjustOutputSignal(WAVEFile*, double*, int) [4]
convolve(double*, double*, double*, int)
   [5] shortToDouble(WAVEFile*, double*) [10] writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [7] WAVEFile::dataToSignal()
                          [6]          fwriteShortLSB(short,          _IO_FILE*)          [18]
__static_initialization_and_destruction_0() [8] WAVEFile::readWAVEFile(char const*)
   [2] createOutputFile(char*) [3] four1(double*, unsigned long, int)
```

8. Sixth Optimization Program - Strength Reduction

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 96.41      5.10     5.10        3     1.70     1.70  four1(double*, unsigned long, int)
  1.89      5.20     0.10        1     0.10     5.28  createOutputFile(char*)
  0.95      5.25     0.05        1     0.05     0.05  convolve(double*, double*, double*, int)
  0.38      5.27     0.02        2     0.01     0.01  shortToDouble(WAVEFile*, double*)
  0.19      5.28     0.01        2     0.01     0.01  WAVEFile::dataToSignal()
  0.19      5.29     0.01        1     0.01     0.01  adjustOutputSignal(WAVEFile*, double*,
int)
```

```
  0.00      5.29     0.00  2758617     0.00     0.00  fwriteShortLSB(short, _IO_FILE*)
  0.00      5.29     0.00        5     0.00     0.00  fwriteIntLSB(int, _IO_FILE*)
  0.00      5.29     0.00        2     0.00     0.01  WAVEFile::readWAVEFile(char const*)
  0.00      5.29     0.00        1     0.00     0.00  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)
    0.00              5.29            0.00                   1            0.00            0.00
__static_initialization_and_destruction_0()

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.19% of 5.29 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00    5.29                 main [1]
                0.10    5.18       1/1          createOutputFile(char*) [2]
                0.00    0.01       2/2          WAVEFile::readWAVEFile(char const*) [7]
-----------------------------------------------
                0.10    5.18       1/1          main [1]
```

```
[2]     99.8    0.10    5.18       1         createOutputFile(char*) [2]
                5.10    0.00       3/3           four1(double*, unsigned long, int) [3]
                0.05    0.00       1/1           convolve(double*, double*, double*, int) [4]
                0.02    0.00       2/2           shortToDouble(WAVEFile*, double*) [5]
                0.01    0.00       1/1           adjustOutputSignal(WAVEFile*, double*, int) [8]
                0.00    0.00 2758613/2758617     fwriteShortLSB(short, _IO_FILE*) [15]
                 0.00    0.00       1/1           writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[17]
-----------------------------------------------
                5.10    0.00       3/3           createOutputFile(char*) [2]
[3]     96.4    5.10    0.00       3         four1(double*, unsigned long, int) [3]
-----------------------------------------------
                0.05    0.00       1/1           createOutputFile(char*) [2]
[4]      0.9    0.05    0.00       1         convolve(double*, double*, double*, int) [4]
-----------------------------------------------
                0.02    0.00       2/2           createOutputFile(char*) [2]
[5]      0.4    0.02    0.00       2         shortToDouble(WAVEFile*, double*) [5]
-----------------------------------------------
                0.01    0.00       2/2           WAVEFile::readWAVEFile(char const*) [7]
[6]      0.2    0.01    0.00       2         WAVEFile::dataToSignal() [6]
-----------------------------------------------
                0.00    0.01       2/2           main [1]
[7]      0.2    0.00    0.01       2         WAVEFile::readWAVEFile(char const*) [7]
                0.01    0.00       2/2           WAVEFile::dataToSignal() [6]
-----------------------------------------------
                0.01    0.00       1/1           createOutputFile(char*) [2]
[8]      0.2    0.01    0.00       1         adjustOutputSignal(WAVEFile*, double*, int) [8]
-----------------------------------------------
                    0.00      0.00        4/2758617      writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [17]
                0.00    0.00 2758613/2758617     createOutputFile(char*) [2]
[15]     0.0    0.00    0.00 2758617         fwriteShortLSB(short, _IO_FILE*) [15]
-----------------------------------------------
                 0.00    0.00       5/5           writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[17]
[16]     0.0    0.00    0.00       5         fwriteIntLSB(int, _IO_FILE*) [16]
-----------------------------------------------
                0.00    0.00       1/1           createOutputFile(char*) [2]
[17]     0.0    0.00    0.00       1         writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[17]
                0.00    0.00       5/5           fwriteIntLSB(int, _IO_FILE*) [16]
                0.00    0.00       4/2758617     fwriteShortLSB(short, _IO_FILE*) [15]
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I_inputfile [19]
[18]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0() [18]
-----------------------------------------------
```

This table describes the call tree of the program, and was sorted by the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the

```
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.
This line lists:
    index       A unique number given to each element of the table.
                Index numbers are sorted numerically.
                The index number is printed next to every function name so
                it is easier to look up where the function is in the table.

    % time      This is the percentage of the `total' time that was spent
                in this function and its children.  Note that due to
                different viewpoints, functions excluded by options, etc,
                these numbers will NOT add up to 100%.

    self This is the total amount of time spent in this function.

    children    This is the total amount of time propagated into this
                function by its children.

    called      This is the number of times the function was called.
                If the function called itself recursively, the number
                only includes non-recursive calls, and is followed by
                a `+' and the number of recursive calls.

    name        The name of the current function.  The index number is
                printed after it.  If the function is a member of a
                cycle, the cycle number is printed between the
                function's name and the index number.


For the function's parents, the fields have the following meanings:

    self This is the amount of time that was propagated directly
                from the function into this parent.

    children    This is the amount of time that was propagated from
                the function's children into this parent.

    called      This is the number of times this parent called the
                function `/' the total number of times the function
                was called.  Recursive calls to the function are not
                included in the number after the `/'.

    name        This is the name of the parent.  The parent's index
                number is printed after it.  If the parent is a
                member of a cycle, the cycle number is printed between
                the name and the index number.

If the parents of the function cannot be determined, the word
`<spontaneous>' is printed in the `name' field, and all the other
fields are blank.
```

```
 For the function's children, the fields have the following meanings:

    self This is the amount of time that was propagated directly
                    from the child into the function.

    children     This is the amount of time that was propagated from the
                    child's children to the function.

    called       This is the number of times the function called
                    this child `/' the total number of times the child
                    was called.  Recursive calls by the child are not
                    listed in the number after the `/'.

    name         This is the name of the child.  The child's index
                    number is printed after it.  If the child is a
                    member of a cycle, the cycle number is printed
                    between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
```

```
Index by function name

  [16] fwriteIntLSB(int, _IO_FILE*) [8] adjustOutputSignal(WAVEFile*, double*, int) [4]
convolve(double*, double*, double*, int)
   [5] shortToDouble(WAVEFile*, double*) [17] writeWAVEFileHeader(int, int, int, int,
_IO_FILE*) [6] WAVEFile::dataToSignal()
                    [15]          fwriteShortLSB(short,          _IO_FILE*)          [18]
__static_initialization_and_destruction_0() [7] WAVEFile::readWAVEFile(char const*)
  [2] createOutputFile(char*) [3] four1(double*, unsigned long, int)
```

9. Compiler Optimization - Compiler Flag -O3

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 92.26      2.86     2.86        3     0.95     0.95  four1(double*, unsigned long, int)
  7.10      3.08     0.22        1     0.22     3.09  createOutputFile(char*)
  0.32      3.09     0.01        2     0.01     0.01  WAVEFile::readWAVEFile(char const*)
  0.32      3.10     0.01        1     0.01     0.01  adjustOutputSignal(WAVEFile*, double*,
int)
```

```
  0.00      3.10     0.00        1     0.00     0.00  writeWAVEFileHeader(int, int, int, int,
_IO_FILE*)

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
               else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
               function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
               the function in the gprof listing. If the index is
               in parenthesis it shows where it would appear in
               the gprof listing if it were to be printed.
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
                          Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.32% of 3.10 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00    3.10                 main [1]
                0.22    2.87      1/1           createOutputFile(char*) [2]
                0.01    0.00      2/2           WAVEFile::readWAVEFile(char const*) [4]
-----------------------------------------------
                0.22    2.87      1/1           main [1]
[2]     99.7    0.22    2.87      1          createOutputFile(char*) [2]
                2.86    0.00      3/3           four1(double*, unsigned long, int) [3]
                0.01    0.00      1/1           adjustOutputSignal(WAVEFile*, double*, int) [5]
                0.00    0.00      1/1           writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[12]
-----------------------------------------------
                2.86    0.00      3/3           createOutputFile(char*) [2]
[3]     92.3    2.86    0.00      3          four1(double*, unsigned long, int) [3]
-----------------------------------------------
                0.01    0.00      2/2           main [1]
[4]      0.3    0.01    0.00      2         WAVEFile::readWAVEFile(char const*) [4]
-----------------------------------------------
                0.01    0.00      1/1           createOutputFile(char*) [2]
```

```
[5]     0.3    0.01    0.00       1         adjustOutputSignal(WAVEFile*, double*, int) [5]
-----------------------------------------------
                0.00    0.00       1/1          createOutputFile(char*) [2]
[12]     0.0    0.00    0.00       1         writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[12]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
                The index number is printed next to every function name so
                it is easier to look up where the function is in the table.

     % time     This is the percentage of the `total' time that was spent
                in this function and its children.  Note that due to
                different viewpoints, functions excluded by options, etc,
                these numbers will NOT add up to 100%.

     self       This is the total amount of time spent in this function.

     children   This is the total amount of time propagated into this
                function by its children.

     called     This is the number of times the function was called.
                If the function called itself recursively, the number
                only includes non-recursive calls, and is followed by
                a `+' and the number of recursive calls.

     name       The name of the current function.  The index number is
                printed after it.  If the function is a member of a
                cycle, the cycle number is printed between the
                function's name and the index number.


 For the function's parents, the fields have the following meanings:

     self       This is the amount of time that was propagated directly
                from the function into this parent.

     children   This is the amount of time that was propagated from
                the function's children into this parent.

     called     This is the number of times this parent called the
                function `/' the total number of times the function
                was called.  Recursive calls to the function are not
                included in the number after the `/'.

     name       This is the name of the parent.  The parent's index
                number is printed after it.  If the parent is a
                member of a cycle, the cycle number is printed between
                the name and the index number.
```

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly
            from the child into the function.

children   This is the amount of time that was propagated from the
            child's children to the function.

called   This is the number of times the function called
            this child `/' the total number of times the child
            was called. Recursive calls by the child are not
            listed in the number after the `/'.

name   This is the name of the child. The child's index
            number is printed after it. If the child is a
            member of a cycle, the cycle number is printed
            between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The `+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Index by function name

[2] createOutputFile(char*) [12] writeWAVEFileHeader(int, int, int, int, _IO_FILE*)
[4] WAVEFile::readWAVEFile(char const*)
[5] adjustOutputSignal(WAVEFile*, double*, int) [3] four1(double*, unsigned long, int)

Version control log report:

```
commit 9215fed598926ced6aef84d3a36c06fe130fe469
Merge: c721042 d91537b
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 22:14:29 2023 -0700

    Merge branch 'main' of https://github.com/viet-ho/CPSC501-A4

commit c721042060af689b65156b50c9fc3524f2f7b1cc
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 22:14:10 2023 -0700

    Added name, ucid and comments to file

commit d91537b125ec17fb21c74ae71d9e83b0fe22d8ab
Author: Viet <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 22:09:33 2023 -0700

    Deleted OutputFileFFT3.wav which is not neccessary to be in Git repo

commit 33f1ab3a6609c68f5f16a4fcf921b6c93f9a6ad3
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 22:07:14 2023 -0700

    Added name, ucid and comments to files

    Remove Regresssion Test file, use diff file1 file2 instead

commit f1dc56c5b5665602b987213856d25e08d51d9903
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 18:52:31 2023 -0700

    Created FFT-Convolve-6.cpp file for the sixth optimization - Strength Reduction

commit ea10adf7791abad5ab812913ae12d1eda07c0e36
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 17:59:42 2023 -0700

    Created FFT-Convolve-5.cpp for the fifth optimization - Optimizing Conditional Statements

commit d5189686551a0019d933e9437def2cd8c3bb1517
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 17:30:26 2023 -0700

    Created FFT-Convolve-4.cpp for the fourth optimization - Optimizing Memory Access

commit 510edb5e310c177a1f299bf79ae59c84adb2e743
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 16:56:32 2023 -0700

    Created FFT-Convolve-3.cpp file for the third optimization - Avoiding Recomputation

commit 5de26e7a114b7e15d1c05ed19fb0191f3cc33fd7
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 16:28:35 2023 -0700

    Updated the way to calculate the process time
```

```
commit 7b41a1dc1ce8e113be0effdc7b2e93ce9f4e98e5
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 15:47:53 2023 -0700

    Created FFT-Convolve-2.cpp for the second optimization - Inline Functions

commit 203870085ce93c672877c19987b375e96969b4d7
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Tue Dec 5 15:14:05 2023 -0700

    Create FFT-Convolve-1.cpp for the first optimization - Loop Unrolling

commit 0e82a73bab420c41e9eeb4416d050434ba4ef897
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sun Dec 3 00:40:02 2023 -0700

    Created Regression Test file

     Created two main functions compareHeaders() and compareData() for comparing WAVE
headers and signal data between two WAVE files.

commit 77dca0a3662443dfb8648d6241718adba4dc6dbe
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sat Dec 2 20:54:29 2023 -0700

    Applied Extract Method for createOutputFile function

commit 3032ba57db76ed230f6842703a5ec2493cf2a1e1
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sat Dec 2 20:15:21 2023 -0700

    Implemented writing header and signal data to output file

    Applied Extract Method for the main method

commit 2a3b0514b01b27fe9f88fe09d30edd0446ee419b
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sat Dec 2 20:04:56 2023 -0700

    Minor change for cleaner code

commit 21c599c7ddb06982b62cd92af322a52d8add663d
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sat Dec 2 17:45:38 2023 -0700

    Added a convolve part to convolve the frequency signals

commit c25b1e6d02e080663bd9401ec5cb9b41a30fd1a8
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sat Dec 2 17:35:21 2023 -0700

    Updated time measurement

commit dce7fffcecd11bd619f86cf8c49508427b3c2d17
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sat Dec 2 12:11:32 2023 -0700

    Minor changed from array to pointer
```

```
commit 4bec6cee346bff602dc49ec37c068b7c057f834d
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sat Dec 2 01:44:23 2023 -0700

    Created FFT-Convolve.cpp file

     Modified code from convolve.cpp file and added four1 function which is a FFT convolution
algorithm.

commit fdc7b9c1abb5f52cf750be09390f793c08084add
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Fri Dec 1 17:09:24 2023 -0700

    Added a function to calculate the convolution time and display to the screen

commit 5fa9609866bf9c41f95f6199c23bbca8877d465c
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Fri Dec 1 11:47:29 2023 -0700

    Added writeWAVEFileHeader function to write the header to output file

     Added fwriteIntLSB and fwriteShortLSB functions to write 4-bit and 2-bit integer to the file
stream

commit 02c9147b248464e005c121555b4a33f02414f910
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Fri Dec 1 01:23:36 2023 -0700

    Added adjustOutputSignal function to adjust the output signal to prevent overflows

commit 13be61eb269a7c13fc7823697d526c198ebf3e1e
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Fri Dec 1 00:14:15 2023 -0700

    Added a function to convert short array to double array

commit 5d1657d964cfcc5c6583fb346ff5ce861d7411bd
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Thu Nov 30 22:11:24 2023 -0700

    Added a convolve function to convolve two signals, producing an output signal.

commit d78f491ad52fa180eed6fa89c15c95a6ee7ec885
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Thu Nov 30 19:20:32 2023 -0700

    Created convolve.cpp

      Take users input from terminal for inputfile, IRfile and outputfile names. Read data in
inputfile and IRfile using readWAVEFile function in WAVEFile.h

commit 7d772ca6200893f5d5f13a1d70244a3578d058a6
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Wed Nov 29 21:00:00 2023 -0700

    Created functions for reading WAVE file and converting data file to signal

commit b3500b218152fd366ecc1da479f66b770df6a488
```

```
Author: viet-ho <103388731+viet-ho@users.noreply.github.com>
Date:   Sun Nov 26 18:03:30 2023 -0700

    Created header file for WAVE file
```

All versions of the program code:

WAVEFile.h:

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: Header file for WAVE file and reading data from file.
*/

#ifndef HEADERFILE_H
#define HEADERFILE_H

#include <iostream>
#include <fstream>
#include <cstring>

class WAVEFile
{

// WAVE file structure
public:
   char chunkID[4];
   int chunkSize;
   char format[4];
   char subChunk1ID[4];
   int subChunk1Size;
   short audioFormat;
   short numChannels;
   int sampleRate;
   int byteRate;
   short blockAlign;
   short bitsPerSample;
   char subChunk2ID[4];
   int subChunk2Size;
   char *audioData;
   short *signal;
   int signalSize;

   // Method to read data from WAVE file
   void readWAVEFile(const char *fileName)
   {
       std::ifstream file(fileName, std::ios::binary);

       file.read(chunkID, sizeof(chunkID));
       file.read(reinterpret_cast<char *>(&chunkSize), sizeof(chunkSize));
       file.read(format, sizeof(format));

       file.read(subChunk1ID, sizeof(subChunk1ID));
```

```
        file.read(reinterpret_cast<char *>(&subChunk1Size), sizeof(subChunk1Size));
        file.read(reinterpret_cast<char *>(&audioFormat), sizeof(audioFormat));
        file.read(reinterpret_cast<char *>(&numChannels), sizeof(numChannels));
        file.read(reinterpret_cast<char *>(&sampleRate), sizeof(sampleRate));
        file.read(reinterpret_cast<char *>(&byteRate), sizeof(byteRate));
        file.read(reinterpret_cast<char *>(&blockAlign), sizeof(blockAlign));
        file.read(reinterpret_cast<char *>(&bitsPerSample), sizeof(bitsPerSample));

        if (subChunk1Size == 18)
        {
            file.ignore(sizeof(short));
        }

        file.read(subChunk2ID, sizeof(subChunk2ID));
        file.read(reinterpret_cast<char *>(&subChunk2Size), sizeof(subChunk2Size));

        audioData = new char[subChunk2Size];
        file.read(audioData, subChunk2Size);

        file.close();

        dataToSignal();
    }

private:
    // Method to convert data to signal based on bitsPerSample
    void dataToSignal()
    {
        signal = nullptr;
        if (bitsPerSample == 8)
        {
            signalSize = subChunk2Size;
            signal = new short[signalSize];
            for (int i = 0; i < subChunk2Size; i++)
            {
                signal[i] = static_cast<short>(static_cast<unsigned char>(audioData[i]));
            }
        }
        else
        {
            signalSize = subChunk2Size / 2;
            signal = new short[signalSize];
            for (int i = 0; i < subChunk2Size; i += 2)
            {
                short tempSignal = static_cast<short>(static_cast<unsigned char>(audioData[i]));
                tempSignal |= static_cast<short>(static_cast<unsigned char>(audioData[i + 1])) << 8;
                signal[i / 2] = tempSignal;
            }
        }
    }
};

#endif
```

1. **Baseline Program (convolve.cpp):**

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: A baseline program where the convolution is implemented
directly in the
    time domain (use the input-side convolution algorithm found on p. 112-115
in the Smith text).
    The program should be invoked from the command line as follows: convolve
inputfile IRfile outputfile
*/

#include <iostream>
#include <fstream>
#include <memory>
#include <ctime>
#include "WAVEFile.h"

using namespace std;

void convolve(double *x, int N, double *h, int M, double *y, int P);
void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double *doubleArray);
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int
output_size);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
size_t fwriteIntLSB(int data, FILE *outputFile);
size_t fwriteShortLSB(short data, FILE *outputFile);

WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
    // Using clock() to calculate the processing time
    clock_t startTime;
    clock_t endTime;
    startTime = clock();
    if (argc != 4)
    {
                cerr << "Usage: ./convolve <inputfile.wav> <IRfile.wav>
<outputfile.wav>\n";
        return EXIT_FAILURE;
    }

    char *inputFileName = argv[1];
    char *IRFileName = argv[2];
    char *outputFileName = argv[3];

    // Read input files
    inputfile->readWAVEFile(inputFileName);
    IRfile->readWAVEFile(IRFileName);

     cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';

    createOutputFile(outputFileName);

    endTime = clock();
    double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("The process was done in %.2f seconds!\n", time);

    return 0;
}

// Method to convolve two signals, producing an output signal
// The convolution was done in the time domain using the "Input Side
Algorithm" (Smith text, p. 112 -115)
void convolve(double *x, int N, double *h, int M, double *y, int P)
{
    int n, m;

    if (P != (N + M - 1))
    {
        printf("Output signal vector is the wrong size\n");
        printf("It is %-d, but should be %-d\n", P, (N + M - 1));
        printf("Aborting convolution\n");
        return;
    }

    for (n = 0; n < P; n++)
    {
        y[n] = 0.0;
    }

    for (n = 0; n < N; n++)
    {
        for (m = 0; m < M; m++)
        {
            y[n + m] += x[n] * h[m];
        }
    }
}



// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];

    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);

    printf("Start convolution...\n");
              convolve(input_signal,    inputfile->signalSize,    IR_signal,
IRfile->signalSize, output_signal, output_size);
    printf("End convolution!\n");

    adjustOutputSignal(inputfile, output_signal, output_size);

    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
                writeWAVEFileHeader(inputfile->numChannels,     output_size,
inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
    for (int i = 0; i < output_size; i++)
    {
        fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
    }
    printf("End writing!\n");

    fclose(outputfile);
}

// Mehthod to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double *doubleArray)
```

```
{
    for (int i = 0; i < (waveFile->signalSize); i++)
    {
        doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
    }
}

// Method to adjust output signal to avoid overflow
void  adjustOutputSignal(WAVEFile  *waveFile,  double  *output_signal,  int
output_size)
{
    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        if (signalInputMax < waveFile->signal[i])
        {
            signalInputMax = waveFile->signal[i];
        }

        if (signalOutputMax < output_signal[i])
        {
            signalOutputMax = output_signal[i];
        }
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{

    int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
    int chunkSize = subChunk2Size + 36;
    short blockAlign = numChannels * (bitsPerSample / 8);
    int byteRate = static_cast<int>(sampleRate * blockAlign);

    fputs("RIFF", outputFile);
    fwriteIntLSB(chunkSize, outputFile);
    fputs("WAVE", outputFile);
    fputs("fmt ", outputFile);
    fwriteIntLSB(16, outputFile);
    fwriteShortLSB(1, outputFile);
    fwriteShortLSB(numChannels, outputFile);
    fwriteIntLSB(sampleRate, outputFile);
    fwriteIntLSB(byteRate, outputFile);
    fwriteShortLSB(blockAlign, outputFile);
    fwriteShortLSB(bitsPerSample, outputFile);
    fputs("data", outputFile);
    fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
size_t fwriteIntLSB(int data, FILE *outputFile)
{

    unsigned char array[4];
    array[3] = (unsigned char)((data >> 24) & 0xFF);
    array[2] = (unsigned char)((data >> 16) & 0xFF);
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to write a 2-byte integer to the file stream
size_t fwriteShortLSB(short data, FILE *outputFile)
{

    unsigned char array[2];
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}
```

2. Algorithm-Based Optimization Program (FFT-Convolve.cpp):

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: A program based on the baseline program (convolve),
with re-implementing the convolution using a
    frequency-domain convolution algorithm via Fast Fourier Transform (FFT).
       The program should be invoked from the command line as follows:
FFT-Convolve inputfile IRfile outputfile
*/

#include <iostream>
#include <fstream>
#include <memory>
```

```
#include <ctime>
#include <math.h>
#include "WAVEFile.h"
#define SWAP(a, b) \
   tempr = (a);   \
   (a) = (b);     \
   (b) = tempr

using namespace std;

void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double doubleArray[]);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
size_t fwriteIntLSB(int data, FILE *outputFile);
size_t fwriteShortLSB(short data, FILE *outputFile);
void four1(double data[], unsigned long nn, int isign);
void  adjustOutputSignal(WAVEFile  *waveFile,  double  *output_signal,  int
output_size);
void  convolve(double  *freq_input_signal,  double  *freq_IR_signal,  double
*freq_output_signal, int length);

WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
   // Using clock() to calculate the processing time
   clock_t startTime;
   clock_t endTime;
   startTime = clock();

   if (argc != 4)
   {
                cerr  <<  "Usage:  ./convolve  <inputfile.wav>  <IRfile.wav>
<outputfile.wav>\n";
       return EXIT_FAILURE;
   }

   char *inputFileName = argv[1];
   char *IRFileName = argv[2];
   char *outputFileName = argv[3];

   // Read input files
   inputfile->readWAVEFile(inputFileName);
   IRfile->readWAVEFile(IRFileName);

    cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';
```

```
    createOutputFile(outputFileName);

    endTime = clock();
    double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("The process was done in %.3f seconds!\n", time);

    return 0;
}

// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    // Apply frequency-domain transformation
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);

        int  maxFileSize  =  (inputfile->signalSize  <=  IRfile->signalSize)  ?
IRfile->signalSize : inputfile->signalSize;
    int powerOfTwo = 1;
    while (powerOfTwo < maxFileSize)
    {
        powerOfTwo *= 2;
    }

    double *freq_input_signal = new double[powerOfTwo * 2];
    double *freq_IR_signal = new double[powerOfTwo * 2];
    double *freq_output_signal = new double[powerOfTwo * 2];

    for (int i = 0; i < (powerOfTwo * 2); i++)
    {
        freq_input_signal[i] = 0.0;
        freq_IR_signal[i] = 0.0;
    }

    for (int i = 0; i < (inputfile->signalSize); i++)
    {
        freq_input_signal[i * 2] = input_signal[i];
    }

    for (int i = 0; i < (IRfile->signalSize); i++)
    {
        freq_IR_signal[i * 2] = IR_signal[i];
    }
    four1(freq_input_signal - 1, powerOfTwo, 1);
    four1(freq_IR_signal - 1, powerOfTwo, 1);

    // Apply frequency-domain convolution
            convolve(freq_input_signal, freq_IR_signal, freq_output_signal,
powerOfTwo * 2);

    four1(freq_output_signal - 1, powerOfTwo, -1);
    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] = freq_output_signal[i * 2];
    }
    adjustOutputSignal(inputfile, output_signal, output_size);

    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
                writeWAVEFileHeader(inputfile->numChannels, output_size,
inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
    for (int i = 0; i < output_size; i++)
    {
        fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
    }
    printf("End writing!\n");

    fclose(outputfile);
}

// Method to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double doubleArray[])
{
    for (int i = 0; i < (waveFile->signalSize); i++)
    {
        doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
    }
}

// Method to convolve frequency-domain signals, producing an output signal
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length)
{
    printf("Start convolution...\n");
    for (int i = 0; i < length; i += 2)
```

```
   {
        freq_output_signal[i] = (freq_input_signal[i] * freq_IR_signal[i]) -
(freq_input_signal[i + 1] * freq_IR_signal[i + 1]);
               freq_output_signal[i   +   1]   =   (freq_input_signal[i   +   1]   *
freq_IR_signal[i]) + (freq_input_signal[i] * freq_IR_signal[i + 1]);
   }
   printf("End convolution!\n");
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{
   int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
   int chunkSize = subChunk2Size + 36;
   short blockAlign = numChannels * (bitsPerSample / 8);
   int byteRate = static_cast<int>(sampleRate * blockAlign);

   fputs("RIFF", outputFile);
   fwriteIntLSB(chunkSize, outputFile);
   fputs("WAVE", outputFile);
   fputs("fmt ", outputFile);
   fwriteIntLSB(16, outputFile);
   fwriteShortLSB(1, outputFile);
   fwriteShortLSB(numChannels, outputFile);
   fwriteIntLSB(sampleRate, outputFile);
   fwriteIntLSB(byteRate, outputFile);
   fwriteShortLSB(blockAlign, outputFile);
   fwriteShortLSB(bitsPerSample, outputFile);
   fputs("data", outputFile);
   fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
size_t fwriteIntLSB(int data, FILE *outputFile)
{
   unsigned char array[4];
   array[3] = (unsigned char)((data >> 24) & 0xFF);
   array[2] = (unsigned char)((data >> 16) & 0xFF);
   array[1] = (unsigned char)((data >> 8) & 0xFF);
   array[0] = (unsigned char)(data & 0xFF);
   return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to write a 2-byte integer to the file stream
size_t fwriteShortLSB(short data, FILE *outputFile)
{
   unsigned char array[2];
   array[1] = (unsigned char)((data >> 8) & 0xFF);
```

```
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}

// Method to adjust output signal to avoid overflow
void  adjustOutputSignal(WAVEFile  *waveFile,  double  *output_signal,  int
output_size)
{

    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        if (signalInputMax < waveFile->signal[i])
        {
            signalInputMax = waveFile->signal[i];
        }

        if (signalOutputMax < output_signal[i])
        {
            signalOutputMax = output_signal[i];
        }
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// The four1 method from Numerical Recipes in C p. 507 - 508.
void four1(double data[], unsigned long 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;
    }
}
```

3. **First Optimization Program - Loop Unrolling (FFT-Convolve-1.cpp):**

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: A program based on the FFT-Convolve program, with
applying the first code-tuning technique optimization.
     The program should be invoked from the command line as follows:
FFT-Convolve-1 inputfile IRfile outputfile
```

```
*/

#include <iostream>
#include <fstream>
#include <memory>
#include <ctime>
#include <math.h>
#include "WAVEFile.h"
#define SWAP(a, b) \
   tempr = (a);   \
   (a) = (b);     \
   (b) = tempr

using namespace std;

void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double doubleArray[]);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
size_t fwriteIntLSB(int data, FILE *outputFile);
size_t fwriteShortLSB(short data, FILE *outputFile);
void four1(double data[], unsigned long nn, int isign);
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int
output_size);
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length);

WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
   // Using clock() to calculate the processing time
   clock_t startTime;
   clock_t endTime;
   startTime = clock();

   if (argc != 4)
   {
               cerr << "Usage: ./convolve <inputfile.wav> <IRfile.wav>
<outputfile.wav>\n";
       return EXIT_FAILURE;
   }

   char *inputFileName = argv[1];
   char *IRFileName = argv[2];
   char *outputFileName = argv[3];

   // Read input files
```

```
    inputfile->readWAVEFile(inputFileName);
    IRfile->readWAVEFile(IRFileName);

    cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';

    createOutputFile(outputFileName);

    endTime = clock();
    double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("The process was done in %.3f seconds!\n", time);

    return 0;
}

// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    // Apply frequency-domain transformation
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);

    int maxFileSize = (inputfile->signalSize <= IRfile->signalSize) ?
IRfile->signalSize : inputfile->signalSize;
    int powerOfTwo = 1;
    while (powerOfTwo < maxFileSize)
    {
        powerOfTwo *= 2;
    }

    double *freq_input_signal = new double[powerOfTwo * 2];
    double *freq_IR_signal = new double[powerOfTwo * 2];
    double *freq_output_signal = new double[powerOfTwo * 2];

    // Loop Unrolling - Optimization 1
    for (int i = 0; i < (powerOfTwo * 2); i += 2)
    {
        freq_input_signal[i] = 0.0;
        freq_IR_signal[i] = 0.0;

        freq_input_signal[i + 1] = 0.0;
        freq_IR_signal[i + 1] = 0.0;
    }

    for (int i = 0; i < (inputfile->signalSize); i++)
    {
        freq_input_signal[i * 2] = input_signal[i];
    }

    for (int i = 0; i < (IRfile->signalSize); i++)
    {
        freq_IR_signal[i * 2] = IR_signal[i];
    }

    four1(freq_input_signal - 1, powerOfTwo, 1);
    four1(freq_IR_signal - 1, powerOfTwo, 1);

    // Apply frequency-domain convolution
    convolve(freq_input_signal, freq_IR_signal, freq_output_signal, powerOfTwo * 2);

    four1(freq_output_signal - 1, powerOfTwo, -1);
    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] = freq_output_signal[i * 2];
    }
    adjustOutputSignal(inputfile, output_signal, output_size);

    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
    writeWAVEFileHeader(inputfile->numChannels, output_size, inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
    for (int i = 0; i < output_size; i++)
    {
        fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
    }
    printf("End writing!\n");

    fclose(outputfile);
}

// Method to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double doubleArray[])
{
    for (int i = 0; i < (waveFile->signalSize); i++)
    {
        doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
```

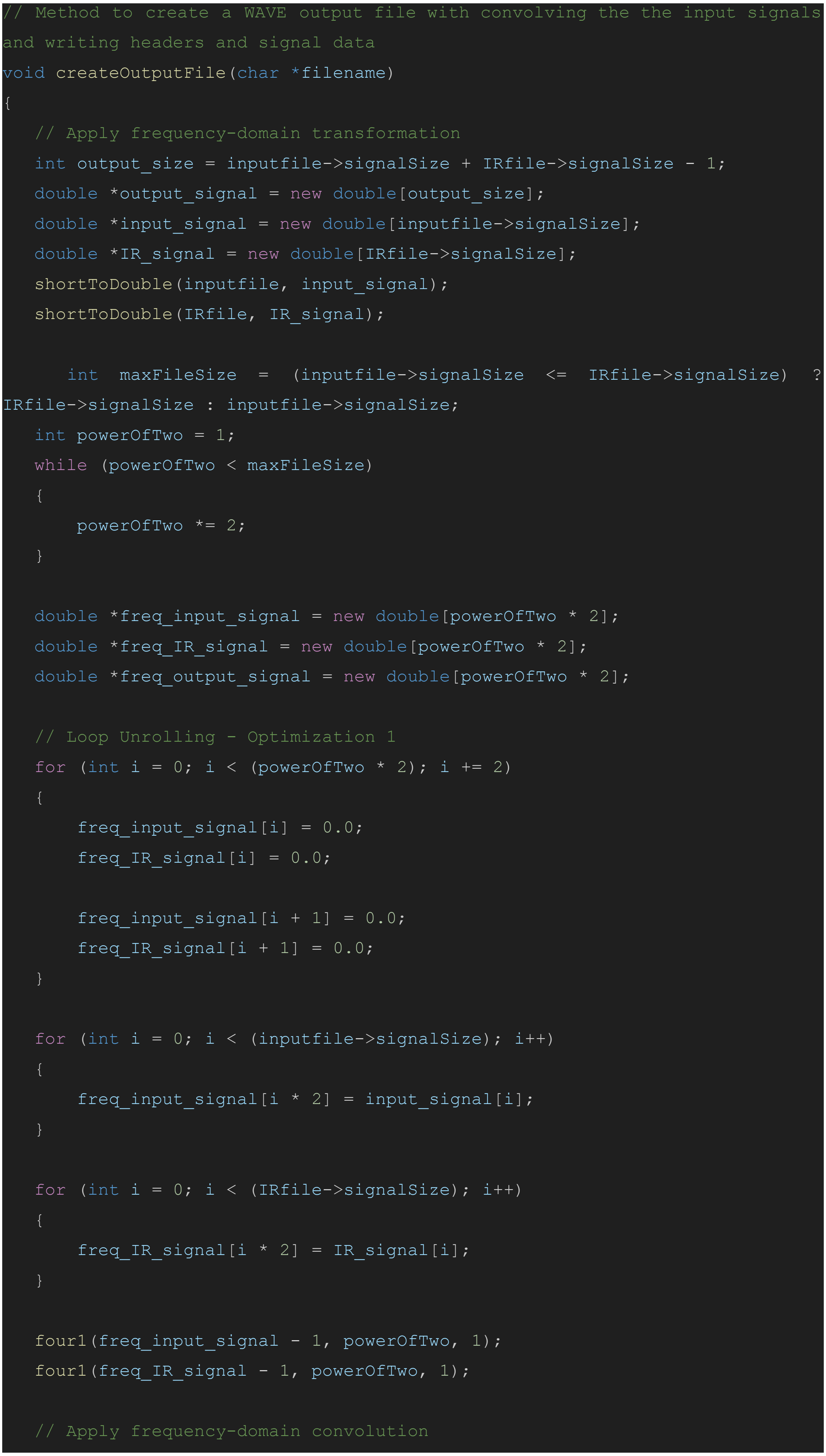
```
   }
}

// Method to convolve frequency-domain signals, producing an output signal
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length)
{
   printf("Start convolution...\n");
   for (int i = 0; i < length; i += 2)
   {
        freq_output_signal[i] = (freq_input_signal[i] * freq_IR_signal[i]) -
(freq_input_signal[i + 1] * freq_IR_signal[i + 1]);
               freq_output_signal[i + 1] = (freq_input_signal[i + 1] *
freq_IR_signal[i]) + (freq_input_signal[i] * freq_IR_signal[i + 1]);
   }
   printf("End convolution!\n");
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{
   int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
   int chunkSize = subChunk2Size + 36;
   short blockAlign = numChannels * (bitsPerSample / 8);
   int byteRate = static_cast<int>(sampleRate * blockAlign);

   fputs("RIFF", outputFile);
   fwriteIntLSB(chunkSize, outputFile);
   fputs("WAVE", outputFile);
   fputs("fmt ", outputFile);
   fwriteIntLSB(16, outputFile);
   fwriteShortLSB(1, outputFile);
   fwriteShortLSB(numChannels, outputFile);
   fwriteIntLSB(sampleRate, outputFile);
   fwriteIntLSB(byteRate, outputFile);
   fwriteShortLSB(blockAlign, outputFile);
   fwriteShortLSB(bitsPerSample, outputFile);
   fputs("data", outputFile);
   fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
size_t fwriteIntLSB(int data, FILE *outputFile)
{
   unsigned char array[4];
   array[3] = (unsigned char)((data >> 24) & 0xFF);
   array[2] = (unsigned char)((data >> 16) & 0xFF);
   array[1] = (unsigned char)((data >> 8) & 0xFF);
```

```
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to write a 2-byte integer to the file stream
size_t fwriteShortLSB(short data, FILE *outputFile)
{
    unsigned char array[2];
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}

// Method to adjust output signal to avoid overflow
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int
output_size)
{

    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        if (signalInputMax < waveFile->signal[i])
        {
            signalInputMax = waveFile->signal[i];
        }

        if (signalOutputMax < output_signal[i])
        {
            signalOutputMax = output_signal[i];
        }
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    // Loop Unrolling - Optimization 1
    int i = 0;
    for (; i <= output_size - 4; i += 4)
    {
        output_signal[i] *= adjustFactor;
        output_signal[i + 1] *= adjustFactor;
        output_signal[i + 2] *= adjustFactor;
        output_signal[i + 3] *= adjustFactor;
    }

    for (; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// The four1 method from Numerical Recipes in C p. 507 - 508.
void four1(double data[], unsigned long 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)
        {
            // Loop Unrolling - Optimization 1
            for (i = m; i <= n; i += istep * 2)
            {
                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;

                if (i + istep <= n)
                {
                    int j_unrolled = i + istep + mmax;
                         double tempr_unrolled = wr * data[j_unrolled] - wi *
data[j_unrolled + 1];
                      double tempi_unrolled = wr * data[j_unrolled + 1] + wi *
data[j_unrolled];
                    data[j_unrolled] = data[i + istep] - tempr_unrolled;
                               data[j_unrolled + 1] = data[i + istep + 1] -
tempi_unrolled;
                    data[i + istep] += tempr_unrolled;
                    data[i + istep + 1] += tempi_unrolled;
                }
            }
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        mmax = istep;
    }
}
```

4. **Second Optimization Program - Inline Functions (FFT-Convolve-2.cpp):**

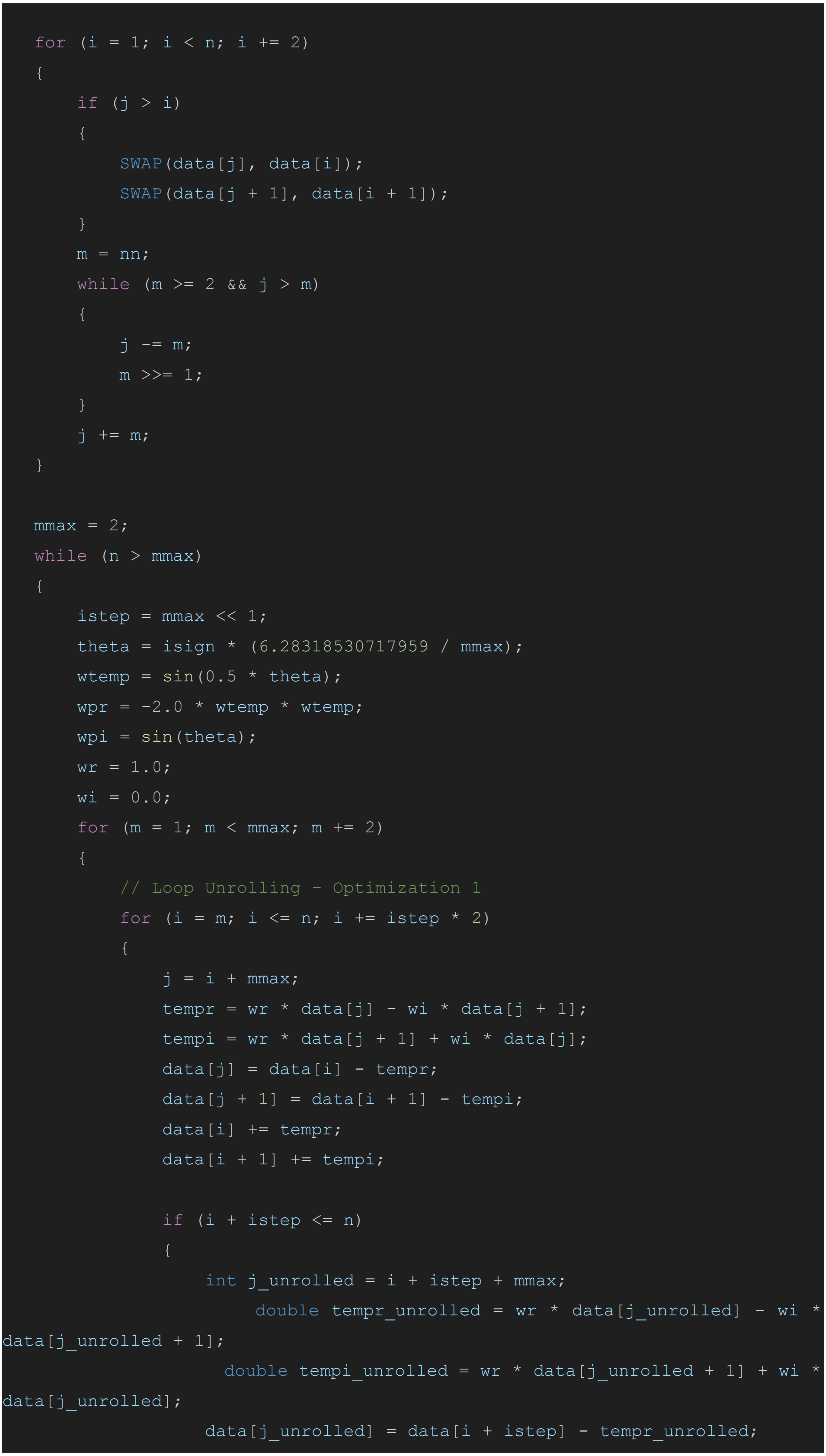
```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: A program based on the FFT-Convolve-1 program, with
applying the second code-tuning technique optimization.
      The program should be invoked from the command line as follows:
FFT-Convolve-2 inputfile IRfile outputfile
*/

#include <iostream>
#include <fstream>
#include <memory>
#include <ctime>
#include <math.h>
#include "WAVEFile.h"
#define SWAP(a, b) \
   tempr = (a);   \
   (a) = (b);     \
   (b) = tempr

using namespace std;
```

```
void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double doubleArray[]);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
inline size_t fwriteIntLSB(int data, FILE *outputFile);
inline size_t fwriteShortLSB(short data, FILE *outputFile);
void four1(double data[], unsigned long nn, int isign);
void  adjustOutputSignal(WAVEFile  *waveFile,  double  *output_signal,  int
output_size);
void  convolve(double  *freq_input_signal,  double  *freq_IR_signal,  double
*freq_output_signal, int length);

WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
   // Using clock() to calculate the processing time
   clock_t startTime;
   clock_t endTime;
   startTime = clock();

   if (argc != 4)
   {
               cerr  <<  "Usage:  ./convolve  <inputfile.wav>  <IRfile.wav>
<outputfile.wav>\n";
       return EXIT_FAILURE;
   }

   char *inputFileName = argv[1];
   char *IRFileName = argv[2];
   char *outputFileName = argv[3];

   // Read input files
   inputfile->readWAVEFile(inputFileName);
   IRfile->readWAVEFile(IRFileName);

    cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';

   createOutputFile(outputFileName);

   endTime = clock();
   double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
   printf("The process was done in %.3f seconds!\n", time);

   return 0;
}
```



```
// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    // Apply frequency-domain transformation
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);

        int maxFileSize = (inputfile->signalSize <= IRfile->signalSize) ?
IRfile->signalSize : inputfile->signalSize;
    int powerOfTwo = 1;
    while (powerOfTwo < maxFileSize)
    {
        powerOfTwo *= 2;
    }

    double *freq_input_signal = new double[powerOfTwo * 2];
    double *freq_IR_signal = new double[powerOfTwo * 2];
    double *freq_output_signal = new double[powerOfTwo * 2];

    // Loop Unrolling - Optimization 1
    for (int i = 0; i < (powerOfTwo * 2); i += 2)
    {
        freq_input_signal[i] = 0.0;
        freq_IR_signal[i] = 0.0;

        freq_input_signal[i + 1] = 0.0;
        freq_IR_signal[i + 1] = 0.0;
    }

    for (int i = 0; i < (inputfile->signalSize); i++)
    {
        freq_input_signal[i * 2] = input_signal[i];
    }

    for (int i = 0; i < (IRfile->signalSize); i++)
    {
        freq_IR_signal[i * 2] = IR_signal[i];
    }

    four1(freq_input_signal - 1, powerOfTwo, 1);
    four1(freq_IR_signal - 1, powerOfTwo, 1);

    // Apply frequency-domain convolution
    convolve(freq_input_signal, freq_IR_signal, freq_output_signal, powerOfTwo * 2);

    four1(freq_output_signal - 1, powerOfTwo, -1);
    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] = freq_output_signal[i * 2];
    }
    adjustOutputSignal(inputfile, output_signal, output_size);

    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
    writeWAVEFileHeader(inputfile->numChannels, output_size, inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
    for (int i = 0; i < output_size; i++)
    {
        fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
    }
    printf("End writing!\n");

    fclose(outputfile);
}

// Method to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double doubleArray[])
{
    for (int i = 0; i < (waveFile->signalSize); i++)
    {
        doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
    }
}

// Method to convolve frequency-domain signals, producing an output signal
void convolve(double *freq_input_signal, double *freq_IR_signal, double *freq_output_signal, int length)
{
    printf("Start convolution...\n");
    for (int i = 0; i < length; i += 2)
    {
        freq_output_signal[i] = (freq_input_signal[i] * freq_IR_signal[i]) - (freq_input_signal[i + 1] * freq_IR_signal[i + 1]);
        freq_output_signal[i + 1] = (freq_input_signal[i + 1] * freq_IR_signal[i]) + (freq_input_signal[i] * freq_IR_signal[i + 1]);
    }
    printf("End convolution!\n");
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{
    int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
    int chunkSize = subChunk2Size + 36;
    short blockAlign = numChannels * (bitsPerSample / 8);
    int byteRate = static_cast<int>(sampleRate * blockAlign);

    fputs("RIFF", outputFile);
    fwriteIntLSB(chunkSize, outputFile);
    fputs("WAVE", outputFile);
    fputs("fmt ", outputFile);
    fwriteIntLSB(16, outputFile);
    fwriteShortLSB(1, outputFile);
    fwriteShortLSB(numChannels, outputFile);
    fwriteIntLSB(sampleRate, outputFile);
    fwriteIntLSB(byteRate, outputFile);
    fwriteShortLSB(blockAlign, outputFile);
    fwriteShortLSB(bitsPerSample, outputFile);
    fputs("data", outputFile);
    fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteIntLSB(int data, FILE *outputFile)
{
    unsigned char array[4];
    array[3] = (unsigned char)((data >> 24) & 0xFF);
    array[2] = (unsigned char)((data >> 16) & 0xFF);
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to write a 2-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteShortLSB(short data, FILE *outputFile)
{
    unsigned char array[2];
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}
```

```
// Method to adjust output signal to avoid overflow
void  adjustOutputSignal(WAVEFile  *waveFile,  double  *output_signal,  int
output_size)
{

    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        if (signalInputMax < waveFile->signal[i])
        {
            signalInputMax = waveFile->signal[i];
        }

        if (signalOutputMax < output_signal[i])
        {
            signalOutputMax = output_signal[i];
        }
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    // Loop Unrolling - Optimization 1
    int i = 0;
    for (; i <= output_size - 4; i += 4)
    {
        output_signal[i] *= adjustFactor;
        output_signal[i + 1] *= adjustFactor;
        output_signal[i + 2] *= adjustFactor;
        output_signal[i + 3] *= adjustFactor;
    }

    for (; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// The four1 method from Numerical Recipes in C p. 507 - 508.
void four1(double data[], unsigned long 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)
       {
           // Loop Unrolling - Optimization 1
           for (i = m; i <= n; i += istep * 2)
           {
               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;

               if (i + istep <= n)
               {
                   int j_unrolled = i + istep + mmax;
                        double tempr_unrolled = wr * data[j_unrolled] - wi *
data[j_unrolled + 1];
                     double tempi_unrolled = wr * data[j_unrolled + 1] + wi *
data[j_unrolled];
                   data[j_unrolled] = data[i + istep] - tempr_unrolled;
```

```
                             data[j_unrolled + 1] = data[i + istep + 1] -
tempi_unrolled;
                   data[i + istep] += tempr_unrolled;
                   data[i + istep + 1] += tempi_unrolled;
               }
           }
           wr = (wtemp = wr) * wpr - wi * wpi + wr;
           wi = wi * wpr + wtemp * wpi + wi;
       }
       mmax = istep;
   }
}
```

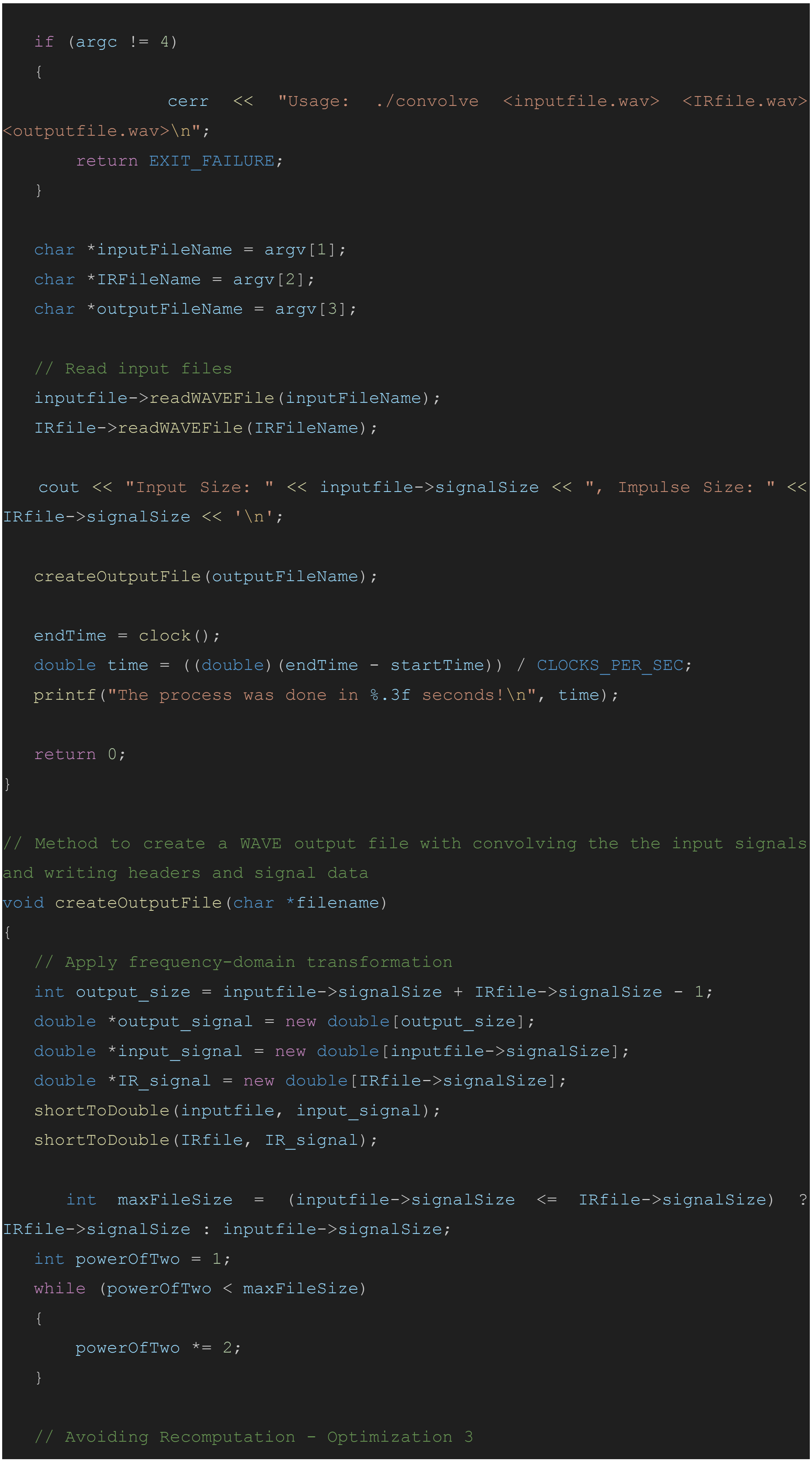
5. Third Optimization Program - Avoiding Recomputation (FFT-Convolve-3.cpp):

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: A program based on the FFT-Convolve-2 program, with
applying the third code-tuning technique optimization.
     The program should be invoked from the command line as follows:
FFT-Convolve-3 inputfile IRfile outputfile
*/

#include <iostream>
#include <fstream>
#include <memory>
#include <ctime>
#include <math.h>
#include "WAVEFile.h"
#define SWAP(a, b) \
   tempr = (a);    \
   (a) = (b);      \
   (b) = tempr

using namespace std;

void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double doubleArray[]);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
inline size_t fwriteIntLSB(int data, FILE *outputFile);
inline size_t fwriteShortLSB(short data, FILE *outputFile);
void four1(double data[], unsigned long nn, int isign);
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int
output_size);
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length);
```

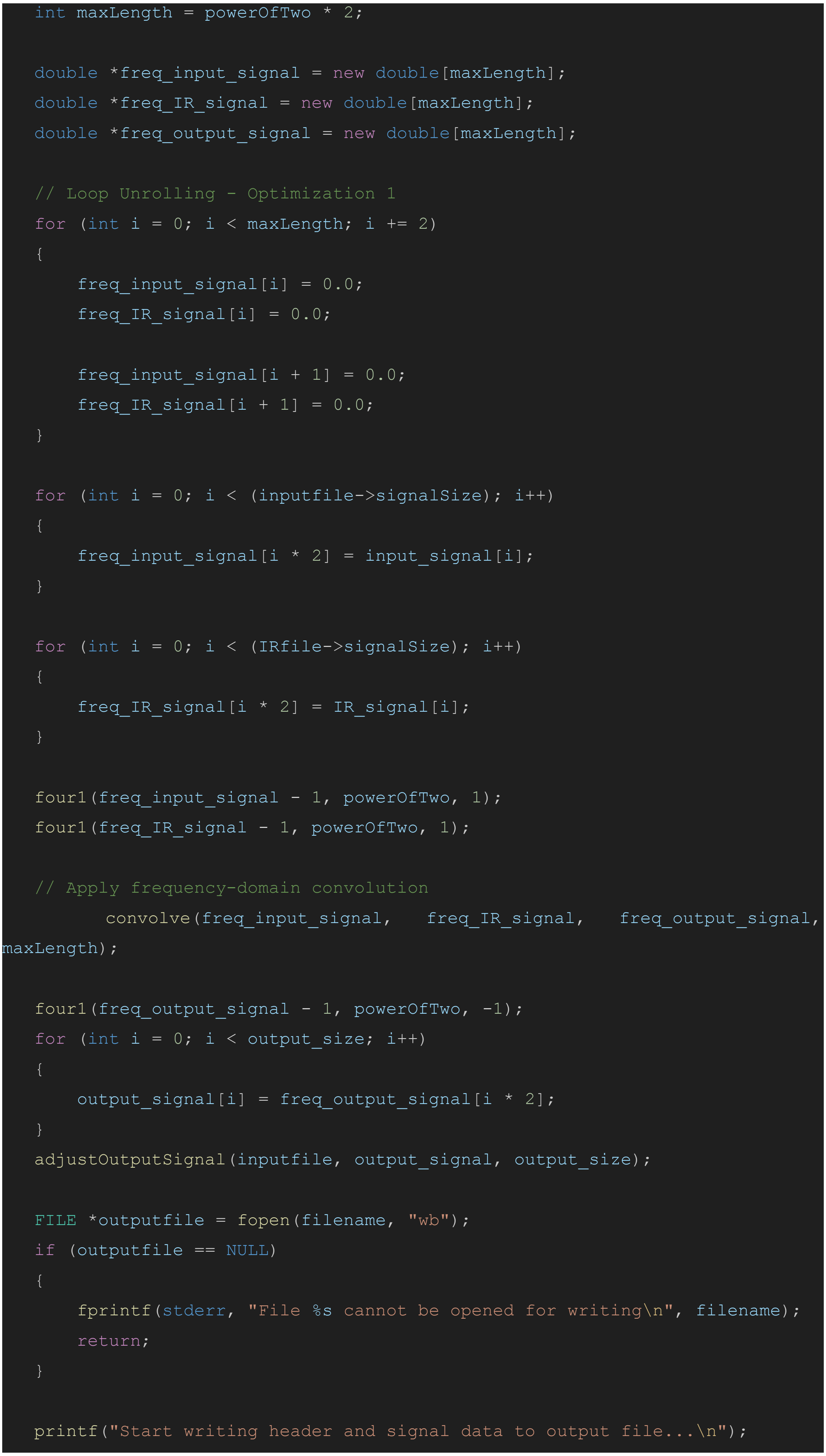
```
WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
    // Using clock() to calculate the processing time
    clock_t startTime;
    clock_t endTime;
    startTime = clock();

    if (argc != 4)
    {
        cerr << "Usage: ./convolve <inputfile.wav> <IRfile.wav>
<outputfile.wav>\n";
        return EXIT_FAILURE;
    }

    char *inputFileName = argv[1];
    char *IRFileName = argv[2];
    char *outputFileName = argv[3];

    // Read input files
    inputfile->readWAVEFile(inputFileName);
    IRfile->readWAVEFile(IRFileName);

    cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';

    createOutputFile(outputFileName);

    endTime = clock();
    double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("The process was done in %.3f seconds!\n", time);

    return 0;
}

// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    // Apply frequency-domain transformation
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);
    int maxFileSize = (inputfile->signalSize <= IRfile->signalSize) ?
IRfile->signalSize : inputfile->signalSize;
    int powerOfTwo = 1;
    while (powerOfTwo < maxFileSize)
    {
        powerOfTwo *= 2;
    }

    // Avoiding Recomputation - Optimization 3
    int maxLength = powerOfTwo * 2;

    double *freq_input_signal = new double[maxLength];
    double *freq_IR_signal = new double[maxLength];
    double *freq_output_signal = new double[maxLength];

    // Loop Unrolling - Optimization 1
    for (int i = 0; i < maxLength; i += 2)
    {
        freq_input_signal[i] = 0.0;
        freq_IR_signal[i] = 0.0;

        freq_input_signal[i + 1] = 0.0;
        freq_IR_signal[i + 1] = 0.0;
    }

    for (int i = 0; i < (inputfile->signalSize); i++)
    {
        freq_input_signal[i * 2] = input_signal[i];
    }

    for (int i = 0; i < (IRfile->signalSize); i++)
    {
        freq_IR_signal[i * 2] = IR_signal[i];
    }

    four1(freq_input_signal - 1, powerOfTwo, 1);
    four1(freq_IR_signal - 1, powerOfTwo, 1);

    // Apply frequency-domain convolution
    convolve(freq_input_signal, freq_IR_signal, freq_output_signal,
maxLength);

    four1(freq_output_signal - 1, powerOfTwo, -1);
    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] = freq_output_signal[i * 2];
    }
    adjustOutputSignal(inputfile, output_signal, output_size);
    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
                writeWAVEFileHeader(inputfile->numChannels,      output_size,
inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
    for (int i = 0; i < output_size; i++)
    {
        fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
    }
    printf("End writing!\n");

    fclose(outputfile);
}

// Method to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double doubleArray[])
{
    for (int i = 0; i < (waveFile->signalSize); i++)
    {
        doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
    }
}

// Method to convolve frequency-domain signals, producing an output signal
void  convolve(double  *freq_input_signal,  double  *freq_IR_signal,  double
*freq_output_signal, int length)
{
    printf("Start convolution...\n");
    for (int i = 0; i < length; i += 2)
    {
         freq_output_signal[i] = (freq_input_signal[i] * freq_IR_signal[i]) -
(freq_input_signal[i + 1] * freq_IR_signal[i + 1]);
                freq_output_signal[i   +   1]   =   (freq_input_signal[i   +   1]   *
freq_IR_signal[i]) + (freq_input_signal[i] * freq_IR_signal[i + 1]);
    }
    printf("End convolution!\n");
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{
    int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
    int chunkSize = subChunk2Size + 36;
    short blockAlign = numChannels * (bitsPerSample / 8);
    int byteRate = static_cast<int>(sampleRate * blockAlign);

    fputs("RIFF", outputFile);
    fwriteIntLSB(chunkSize, outputFile);
    fputs("WAVE", outputFile);
    fputs("fmt ", outputFile);
    fwriteIntLSB(16, outputFile);
    fwriteShortLSB(1, outputFile);
    fwriteShortLSB(numChannels, outputFile);
    fwriteIntLSB(sampleRate, outputFile);
    fwriteIntLSB(byteRate, outputFile);
    fwriteShortLSB(blockAlign, outputFile);
    fwriteShortLSB(bitsPerSample, outputFile);
    fputs("data", outputFile);
    fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteIntLSB(int data, FILE *outputFile)
{
    unsigned char array[4];
    array[3] = (unsigned char)((data >> 24) & 0xFF);
    array[2] = (unsigned char)((data >> 16) & 0xFF);
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to writes a 2-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteShortLSB(short data, FILE *outputFile)
{
    unsigned char array[2];
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}

// Method to adjust output signal to avoid overflow
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int output_size)
{

    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        if (signalInputMax < waveFile->signal[i])
        {
            signalInputMax = waveFile->signal[i];
        }

        if (signalOutputMax < output_signal[i])
        {
            signalOutputMax = output_signal[i];
        }
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    // Loop Unrolling - Optimization 1
    int i = 0;
    for (; i <= output_size - 4; i += 4)
    {
        output_signal[i] *= adjustFactor;
        output_signal[i + 1] *= adjustFactor;
        output_signal[i + 2] *= adjustFactor;
        output_signal[i + 3] *= adjustFactor;
    }

    for (; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// The four1 method from Numerical Recipes in C p. 507 - 508.
void four1(double data[], unsigned long 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)
        {
            // Loop Unrolling - Optimization 1
            for (i = m; i <= n; i += istep * 2)
            {
                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;

                if (i + istep <= n)
                {
                    int j_unrolled = i + istep + mmax;
                         double tempr_unrolled = wr * data[j_unrolled] - wi *
data[j_unrolled + 1];
                      double tempi_unrolled = wr * data[j_unrolled + 1] + wi *
data[j_unrolled];
                    data[j_unrolled] = data[i + istep] - tempr_unrolled;
                              data[j_unrolled + 1] = data[i + istep + 1] -
tempi_unrolled;
                    data[i + istep] += tempr_unrolled;
                    data[i + istep + 1] += tempi_unrolled;
                }
            }
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        mmax = istep;
    }
}
```

6. Fourth Optimization Program - Optimizing Memory Access (FFT-Convolve-4.cpp):

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: A program based on the FFT-Convolve-3 program, with
applying the fourth code-tuning technique optimization.
      The program should be invoked from the command line as follows:
FFT-Convolve-4 inputfile IRfile outputfile
*/

#include <iostream>
#include <fstream>
#include <memory>
#include <ctime>
#include <math.h>
#include "WAVEFile.h"
#define SWAP(a, b) \
    tempr = (a);    \
    (a) = (b);      \
    (b) = tempr

using namespace std;

void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double doubleArray[]);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
inline size_t fwriteIntLSB(int data, FILE *outputFile);
inline size_t fwriteShortLSB(short data, FILE *outputFile);
void four1(double data[], unsigned long nn, int isign);
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int
output_size);
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length);

WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
    // Using clock() to calculate the processing time
    clock_t startTime;
    clock_t endTime;
    startTime = clock();
```



```
    if (argc != 4)
    {
                cerr << "Usage: ./convolve <inputfile.wav> <IRfile.wav>
<outputfile.wav>\n";
        return EXIT_FAILURE;
    }

    char *inputFileName = argv[1];
    char *IRFileName = argv[2];
    char *outputFileName = argv[3];

    // Read input files
    inputfile->readWAVEFile(inputFileName);
    IRfile->readWAVEFile(IRFileName);

     cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';

    createOutputFile(outputFileName);

    endTime = clock();
    double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("The process was done in %.3f seconds!\n", time);

    return 0;
}

// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    // Apply frequency-domain transformation
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);

       int maxFileSize = (inputfile->signalSize <= IRfile->signalSize) ?
IRfile->signalSize : inputfile->signalSize;
    int powerOfTwo = 1;
    while (powerOfTwo < maxFileSize)
    {
        powerOfTwo *= 2;
    }

    // Avoiding Recomputation - Optimization 3
```



```
    int maxLength = powerOfTwo * 2;

    double *freq_input_signal = new double[maxLength];
    double *freq_IR_signal = new double[maxLength];
    double *freq_output_signal = new double[maxLength];

    // Loop Unrolling - Optimization 1
    for (int i = 0; i < maxLength; i += 2)
    {
        freq_input_signal[i] = 0.0;
        freq_IR_signal[i] = 0.0;

        freq_input_signal[i + 1] = 0.0;
        freq_IR_signal[i + 1] = 0.0;
    }

    for (int i = 0; i < (inputfile->signalSize); i++)
    {
        freq_input_signal[i * 2] = input_signal[i];
    }

    for (int i = 0; i < (IRfile->signalSize); i++)
    {
        freq_IR_signal[i * 2] = IR_signal[i];
    }

    four1(freq_input_signal - 1, powerOfTwo, 1);
    four1(freq_IR_signal - 1, powerOfTwo, 1);

    // Apply frequency-domain convolution
          convolve(freq_input_signal,   freq_IR_signal,   freq_output_signal,
maxLength);

    four1(freq_output_signal - 1, powerOfTwo, -1);
    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] = freq_output_signal[i * 2];
    }
    adjustOutputSignal(inputfile, output_signal, output_size);

    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
```

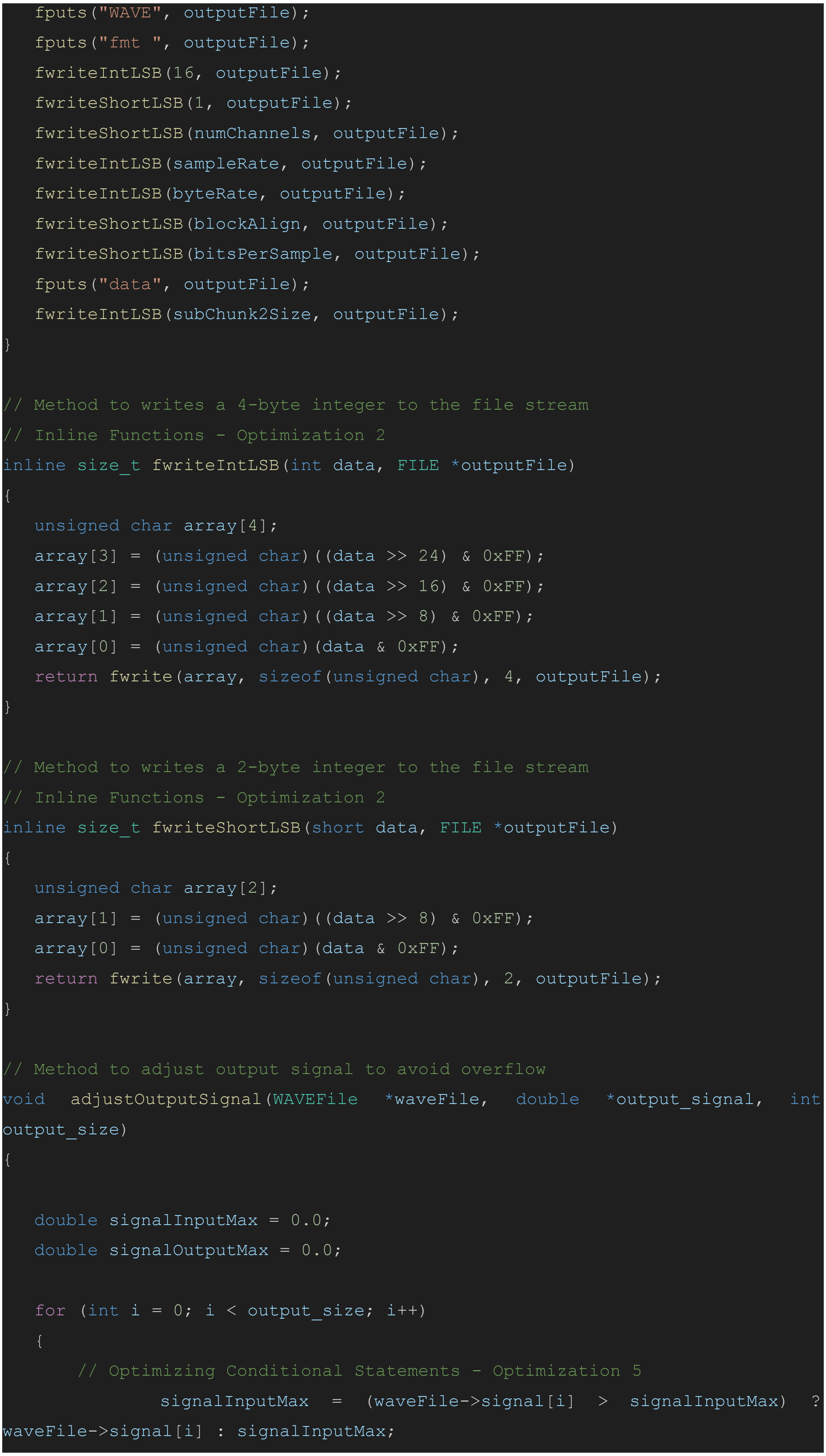
```
                writeWAVEFileHeader(inputfile->numChannels,      output_size,
inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
   for (int i = 0; i < output_size; i++)
   {
       fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
   }
   printf("End writing!\n");

   fclose(outputfile);
}

// Method to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double doubleArray[])
{
   for (int i = 0; i < (waveFile->signalSize); i++)
   {
       doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
   }
}

// Method to convolve frequency-domain signals, producing an output signal
void  convolve(double  *freq_input_signal,  double  *freq_IR_signal,  double
*freq_output_signal, int length)
{
   printf("Start convolution...\n");

   // Optimizing Memory Access - Optimization 4
   for (int i = 0; i < length; i += 2)
   {
       double inputReal = freq_input_signal[i];
       double inputImag = freq_input_signal[i + 1];
       double irReal = freq_IR_signal[i];
       double irImag = freq_IR_signal[i + 1];

       freq_output_signal[i] = (inputReal * irReal) - (inputImag * irImag);
          freq_output_signal[i  + 1]  =  (inputImag  *  irReal)  +  (inputReal  *
irImag);
   }
   printf("End convolution!\n");
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{
   int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
   int chunkSize = subChunk2Size + 36;
   short blockAlign = numChannels * (bitsPerSample / 8);
   int byteRate = static_cast<int>(sampleRate * blockAlign);
```

```
    fputs("RIFF", outputFile);
    fwriteIntLSB(chunkSize, outputFile);
    fputs("WAVE", outputFile);
    fputs("fmt ", outputFile);
    fwriteIntLSB(16, outputFile);
    fwriteShortLSB(1, outputFile);
    fwriteShortLSB(numChannels, outputFile);
    fwriteIntLSB(sampleRate, outputFile);
    fwriteIntLSB(byteRate, outputFile);
    fwriteShortLSB(blockAlign, outputFile);
    fwriteShortLSB(bitsPerSample, outputFile);
    fputs("data", outputFile);
    fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteIntLSB(int data, FILE *outputFile)
{
    unsigned char array[4];
    array[3] = (unsigned char)((data >> 24) & 0xFF);
    array[2] = (unsigned char)((data >> 16) & 0xFF);
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to writes a 2-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteShortLSB(short data, FILE *outputFile)
{
    unsigned char array[2];
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}

// Method to adjust output signal to avoid overflow
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int output_size)
{

    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        if (signalInputMax < waveFile->signal[i])
        {
            signalInputMax = waveFile->signal[i];
        }

        if (signalOutputMax < output_signal[i])
        {
            signalOutputMax = output_signal[i];
        }
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    // Loop Unrolling - Optimization 1
    int i = 0;
    for (; i <= output_size - 4; i += 4)
    {
        output_signal[i] *= adjustFactor;
        output_signal[i + 1] *= adjustFactor;
        output_signal[i + 2] *= adjustFactor;
        output_signal[i + 3] *= adjustFactor;
    }

    for (; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// The four1 method from Numerical Recipes in C p. 507 - 508.
void four1(double data[], unsigned long 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)
        {
            // Loop Unrolling - Optimization 1
            for (i = m; i <= n; i += istep * 2)
            {
                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;

                if (i + istep <= n)
                {
                    int j_unrolled = i + istep + mmax;
                    double tempr_unrolled = wr * data[j_unrolled] - wi * data[j_unrolled + 1];
                    double tempi_unrolled = wr * data[j_unrolled + 1] + wi * data[j_unrolled];
                    data[j_unrolled] = data[i + istep] - tempr_unrolled;
                    data[j_unrolled + 1] = data[i + istep + 1] - tempi_unrolled;
                    data[i + istep] += tempr_unrolled;
                    data[i + istep + 1] += tempi_unrolled;
                }
            }
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        mmax = istep;
    }
}
```

7. **Fifth Optimization Program - Optimizing Conditional Statements (FFT-Convolve-5.cpp):**

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
*  Class Description: A program based on the FFT-Convolve-4 program, with
applying the fifth code-tuning technique optimization.
     The program should be invoked from the command line as follows:
FFT-Convolve-5 inputfile IRfile outputfile
*/

#include <iostream>
#include <fstream>
#include <memory>
#include <ctime>
#include <math.h>
#include "WAVEFile.h"
#define SWAP(a, b) \
   tempr = (a);   \
   (a) = (b);     \
   (b) = tempr

using namespace std;

void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double doubleArray[]);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
inline size_t fwriteIntLSB(int data, FILE *outputFile);
inline size_t fwriteShortLSB(short data, FILE *outputFile);
void four1(double data[], unsigned long nn, int isign);
void  adjustOutputSignal(WAVEFile  *waveFile,  double  *output_signal,  int
output_size);
void  convolve(double  *freq_input_signal,  double  *freq_IR_signal,  double
*freq_output_signal, int length);

WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
   // Using clock() to calculate the processing time
   clock_t startTime;
   clock_t endTime;
   startTime = clock();

   if (argc != 4)
```

```
    {
        cerr << "Usage: ./convolve <inputfile.wav> <IRfile.wav>
<outputfile.wav>\n";
        return EXIT_FAILURE;
    }

    char *inputFileName = argv[1];
    char *IRFileName = argv[2];
    char *outputFileName = argv[3];

    // Read input files
    inputfile->readWAVEFile(inputFileName);
    IRfile->readWAVEFile(IRFileName);

    cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';

    createOutputFile(outputFileName);

    endTime = clock();
    double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("The process was done in %.3f seconds!\n", time);

    return 0;
}

// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    // Apply frequency-domain transformation
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);

    int maxFileSize = (inputfile->signalSize <= IRfile->signalSize) ?
IRfile->signalSize : inputfile->signalSize;
    int powerOfTwo = 1;
    while (powerOfTwo < maxFileSize)
    {
        powerOfTwo *= 2;
    }

    // Avoiding Recomputation - Optimization 3
    int maxLength = powerOfTwo * 2;
    double *freq_input_signal = new double[maxLength];
    double *freq_IR_signal = new double[maxLength];
    double *freq_output_signal = new double[maxLength];

    // Loop Unrolling - Optimization 1
    for (int i = 0; i < maxLength; i += 2)
    {
        freq_input_signal[i] = 0.0;
        freq_IR_signal[i] = 0.0;

        freq_input_signal[i + 1] = 0.0;
        freq_IR_signal[i + 1] = 0.0;
    }

    for (int i = 0; i < (inputfile->signalSize); i++)
    {
        freq_input_signal[i * 2] = input_signal[i];
    }

    for (int i = 0; i < (IRfile->signalSize); i++)
    {
        freq_IR_signal[i * 2] = IR_signal[i];
    }

    four1(freq_input_signal - 1, powerOfTwo, 1);
    four1(freq_IR_signal - 1, powerOfTwo, 1);

    // Apply frequency-domain convolution
           convolve(freq_input_signal,   freq_IR_signal,   freq_output_signal,
maxLength);

    four1(freq_output_signal - 1, powerOfTwo, -1);
    for (int i = 0; i < output_size; i++)
    {
        output_signal[i] = freq_output_signal[i * 2];
    }
    adjustOutputSignal(inputfile, output_signal, output_size);

    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
                writeWAVEFileHeader(inputfile->numChannels,      output_size,
inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
    for (int i = 0; i < output_size; i++)
    {
        fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
    }
    printf("End writing!\n");

    fclose(outputfile);
}

// Method to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double doubleArray[])
{
    for (int i = 0; i < (waveFile->signalSize); i++)
    {
        doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
    }
}

// Method to convolve frequency-domain signals, producing an output signal
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length)
{
    printf("Start convolution...\n");

    // Optimizing Memory Access - Optimization 4
    for (int i = 0; i < length; i += 2)
    {
        double inputReal = freq_input_signal[i];
        double inputImag = freq_input_signal[i + 1];
        double irReal = freq_IR_signal[i];
        double irImag = freq_IR_signal[i + 1];

        freq_output_signal[i] = (inputReal * irReal) - (inputImag * irImag);
        freq_output_signal[i + 1] = (inputImag * irReal) + (inputReal *
irImag);
    }
    printf("End convolution!\n");
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{
    int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
    int chunkSize = subChunk2Size + 36;
    short blockAlign = numChannels * (bitsPerSample / 8);
    int byteRate = static_cast<int>(sampleRate * blockAlign);

    fputs("RIFF", outputFile);
    fwriteIntLSB(chunkSize, outputFile);
```



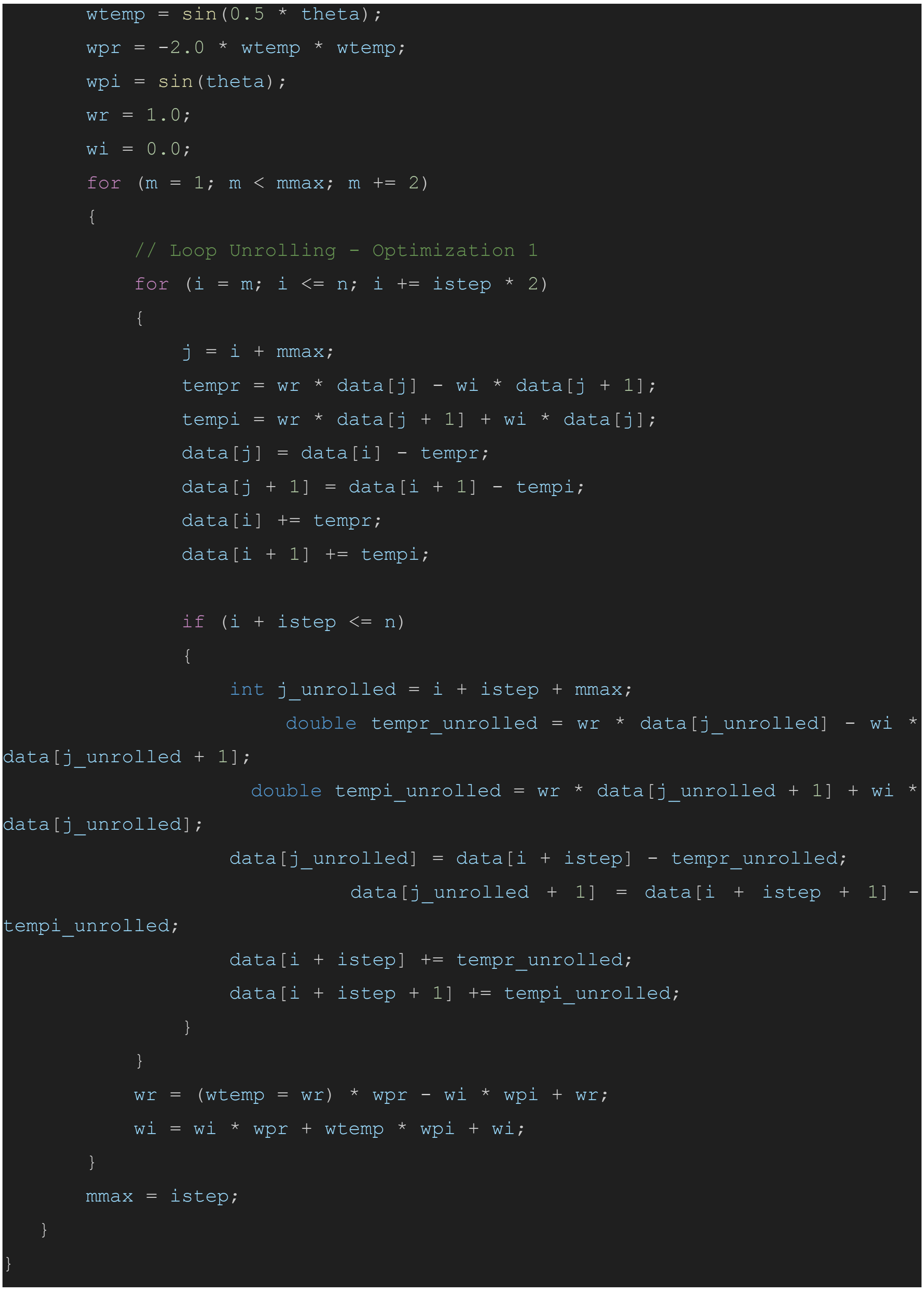
```
    fputs("WAVE", outputFile);
    fputs("fmt ", outputFile);
    fwriteIntLSB(16, outputFile);
    fwriteShortLSB(1, outputFile);
    fwriteShortLSB(numChannels, outputFile);
    fwriteIntLSB(sampleRate, outputFile);
    fwriteIntLSB(byteRate, outputFile);
    fwriteShortLSB(blockAlign, outputFile);
    fwriteShortLSB(bitsPerSample, outputFile);
    fputs("data", outputFile);
    fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteIntLSB(int data, FILE *outputFile)
{
    unsigned char array[4];
    array[3] = (unsigned char)((data >> 24) & 0xFF);
    array[2] = (unsigned char)((data >> 16) & 0xFF);
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to writes a 2-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteShortLSB(short data, FILE *outputFile)
{
    unsigned char array[2];
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}

// Method to adjust output signal to avoid overflow
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int
output_size)
{

    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        // Optimizing Conditional Statements - Optimization 5
                signalInputMax = (waveFile->signal[i] > signalInputMax) ?
waveFile->signal[i] : signalInputMax;
```

```
            signalOutputMax = (output_signal[i] > signalOutputMax) ?
output_signal[i] : signalOutputMax;
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    // Loop Unrolling - Optimization 1
    int i = 0;
    for (; i <= output_size - 4; i += 4)
    {
        output_signal[i] *= adjustFactor;
        output_signal[i + 1] *= adjustFactor;
        output_signal[i + 2] *= adjustFactor;
        output_signal[i + 3] *= adjustFactor;
    }

    for (; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// The four1 method from Numerical Recipes in C p. 507 - 508.
void four1(double data[], unsigned long 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)
        {
            // Loop Unrolling - Optimization 1
            for (i = m; i <= n; i += istep * 2)
            {
                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;

                if (i + istep <= n)
                {
                    int j_unrolled = i + istep + mmax;
                    double tempr_unrolled = wr * data[j_unrolled] - wi * data[j_unrolled + 1];
                    double tempi_unrolled = wr * data[j_unrolled + 1] + wi * data[j_unrolled];
                    data[j_unrolled] = data[i + istep] - tempr_unrolled;
                    data[j_unrolled + 1] = data[i + istep + 1] - tempi_unrolled;
                    data[i + istep] += tempr_unrolled;
                    data[i + istep + 1] += tempi_unrolled;
                }
            }
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        mmax = istep;
    }
}
```

8. **Sixth Optimization Program - Strength Reduction (FFT-Convolve-6.cpp):**

```
/*
* Name: Viet Ho
* UCID: 30122283
* Date: Dec. 5th, 2023
* Class Description: A program based on the FFT-Convolve-5 program, with
applying the fifth code-tuning technique optimization.
     The program should be invoked from the command line as follows:
FFT-Convolve-6 inputfile IRfile outputfile
*/

#include <iostream>
#include <fstream>
#include <memory>
#include <ctime>
#include <math.h>
#include "WAVEFile.h"
#define SWAP(a, b) \
   tempr = (a);   \
   (a) = (b);     \
   (b) = tempr

using namespace std;

void createOutputFile(char *filename);
void shortToDouble(WAVEFile *waveFile, double doubleArray[]);
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile);
inline size_t fwriteIntLSB(int data, FILE *outputFile);
inline size_t fwriteShortLSB(short data, FILE *outputFile);
void four1(double data[], unsigned long nn, int isign);
void adjustOutputSignal(WAVEFile *waveFile, double *output_signal, int
output_size);
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length);

WAVEFile *inputfile = new WAVEFile();
WAVEFile *IRfile = new WAVEFile();

int main(int argc, char *argv[])
{
   // Using clock() to calculate the processing time
   clock_t startTime;
   clock_t endTime;
   startTime = clock();

   if (argc != 4)
   {
                cerr << "Usage: ./convolve <inputfile.wav> <IRfile.wav>
<outputfile.wav>\n";
       return EXIT_FAILURE;
   }

   char *inputFileName = argv[1];
```

```
    char *IRFileName = argv[2];
    char *outputFileName = argv[3];

    // Read input files
    inputfile->readWAVEFile(inputFileName);
    IRfile->readWAVEFile(IRFileName);

    cout << "Input Size: " << inputfile->signalSize << ", Impulse Size: " <<
IRfile->signalSize << '\n';

    createOutputFile(outputFileName);

    endTime = clock();
    double time = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("The process was done in %.3f seconds!\n", time);

    return 0;
}

// Method to create a WAVE output file with convolving the the input signals
and writing headers and signal data
void createOutputFile(char *filename)
{
    // Apply frequency-domain transformation
    int output_size = inputfile->signalSize + IRfile->signalSize - 1;
    double *output_signal = new double[output_size];
    double *input_signal = new double[inputfile->signalSize];
    double *IR_signal = new double[IRfile->signalSize];
    shortToDouble(inputfile, input_signal);
    shortToDouble(IRfile, IR_signal);

    int maxFileSize = (inputfile->signalSize <= IRfile->signalSize) ?
IRfile->signalSize : inputfile->signalSize;
    int powerOfTwo = 1;
    while (powerOfTwo < maxFileSize)
    {
        // Strength Reduction - Optimization 6
        powerOfTwo <<= 1;
    }

    // Avoiding Recomputation - Optimization 3
    int maxLength = powerOfTwo * 2;

    double *freq_input_signal = new double[maxLength];
    double *freq_IR_signal = new double[maxLength];
    double *freq_output_signal = new double[maxLength];

    // Loop Unrolling - Optimization 1
    for (int i = 0; i < maxLength; i += 2)
    {
        freq_input_signal[i] = 0.0;
        freq_IR_signal[i] = 0.0;

        freq_input_signal[i + 1] = 0.0;
        freq_IR_signal[i + 1] = 0.0;
    }

    for (int i = 0; i < (inputfile->signalSize); i++)
    {
        // Strength Reduction - Optimization 6
        freq_input_signal[i<<1] = input_signal[i];
    }

    for (int i = 0; i < (IRfile->signalSize); i++)
    {
        // Strength Reduction - Optimization 6
        freq_IR_signal[i<<1] = IR_signal[i];
    }

    four1(freq_input_signal - 1, powerOfTwo, 1);
    four1(freq_IR_signal - 1, powerOfTwo, 1);

    // Apply frequency-domain convolution
          convolve(freq_input_signal,   freq_IR_signal,   freq_output_signal,
maxLength);

    four1(freq_output_signal - 1, powerOfTwo, -1);
    for (int i = 0; i < output_size; i++)
    {
        // Strength Reduction - Optimization 6
        output_signal[i] = freq_output_signal[i<<1];
    }
    adjustOutputSignal(inputfile, output_signal, output_size);

    FILE *outputfile = fopen(filename, "wb");
    if (outputfile == NULL)
    {
        fprintf(stderr, "File %s cannot be opened for writing\n", filename);
        return;
    }

    printf("Start writing header and signal data to output file...\n");
                  writeWAVEFileHeader(inputfile->numChannels,      output_size,
inputfile->bitsPerSample, inputfile->sampleRate, outputfile);
    for (int i = 0; i < output_size; i++)
    {
        fwriteShortLSB(static_cast<short>(output_signal[i]), outputfile);
    }
    printf("End writing!\n");

    fclose(outputfile);
}

// Method to convert short(signal) to double
void shortToDouble(WAVEFile *waveFile, double doubleArray[])
{
    for (int i = 0; i < (waveFile->signalSize); i++)
    {
        doubleArray[i] = ((double)waveFile->signal[i]) / 32678.0;
    }
}

// Method to convolve frequency-domain signals, producing an output signal
void convolve(double *freq_input_signal, double *freq_IR_signal, double
*freq_output_signal, int length)
{
    printf("Start convolution...\n");

    // Optimizing Memory Access - Optimization 4
    for (int i = 0; i < length; i += 2)
    {
        double inputReal = freq_input_signal[i];
        double inputImag = freq_input_signal[i + 1];
        double irReal = freq_IR_signal[i];
        double irImag = freq_IR_signal[i + 1];

        freq_output_signal[i] = (inputReal * irReal) - (inputImag * irImag);
            freq_output_signal[i + 1] = (inputImag * irReal) + (inputReal *
irImag);
    }
    printf("End convolution!\n");
}

// Method to write the header of the wave file
void writeWAVEFileHeader(int numChannels, int numSamples, int bitsPerSample,
int sampleRate, FILE *outputFile)
{
    int subChunk2Size = numChannels * numSamples * (bitsPerSample / 8);
    int chunkSize = subChunk2Size + 36;
    short blockAlign = numChannels * (bitsPerSample / 8);
    int byteRate = static_cast<int>(sampleRate * blockAlign);

    fputs("RIFF", outputFile);
    fwriteIntLSB(chunkSize, outputFile);
    fputs("WAVE", outputFile);
    fputs("fmt ", outputFile);
    fwriteIntLSB(16, outputFile);
    fwriteShortLSB(1, outputFile);
    fwriteShortLSB(numChannels, outputFile);
    fwriteIntLSB(sampleRate, outputFile);
    fwriteIntLSB(byteRate, outputFile);
    fwriteShortLSB(blockAlign, outputFile);
    fwriteShortLSB(bitsPerSample, outputFile);
    fputs("data", outputFile);
    fwriteIntLSB(subChunk2Size, outputFile);
}

// Method to writes a 4-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteIntLSB(int data, FILE *outputFile)
{
    unsigned char array[4];
    array[3] = (unsigned char)((data >> 24) & 0xFF);
    array[2] = (unsigned char)((data >> 16) & 0xFF);
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 4, outputFile);
}

// Method to writes a 2-byte integer to the file stream
// Inline Functions - Optimization 2
inline size_t fwriteShortLSB(short data, FILE *outputFile)
{
    unsigned char array[2];
    array[1] = (unsigned char)((data >> 8) & 0xFF);
    array[0] = (unsigned char)(data & 0xFF);
    return fwrite(array, sizeof(unsigned char), 2, outputFile);
}

// Method to adjust output signal to avoid overflow
void  adjustOutputSignal(WAVEFile  *waveFile,  double  *output_signal,  int
output_size)
{

    double signalInputMax = 0.0;
    double signalOutputMax = 0.0;

    for (int i = 0; i < output_size; i++)
    {
        // Optimizing Conditional Statements - Optimization 5
                signalInputMax  =  (waveFile->signal[i]  >  signalInputMax)  ?
waveFile->signal[i] : signalInputMax;
                signalOutputMax  =  (output_signal[i]  >  signalOutputMax)  ?
output_signal[i] : signalOutputMax;
    }

    double adjustFactor = signalInputMax / signalOutputMax;

    // Loop Unrolling - Optimization 1
    int i = 0;
    for (; i <= output_size - 4; i += 4)
    {
        output_signal[i] *= adjustFactor;
        output_signal[i + 1] *= adjustFactor;
        output_signal[i + 2] *= adjustFactor;
        output_signal[i + 3] *= adjustFactor;
    }

    for (; i < output_size; i++)
    {
        output_signal[i] *= adjustFactor;
    }
}

// The four1 method from Numerical Recipes in C p. 507 - 508.
void four1(double data[], unsigned long 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)
        {
            // Loop Unrolling - Optimization 1
            for (i = m; i <= n; i += istep * 2)
            {
                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;

                if (i + istep <= n)
                {
                    int j_unrolled = i + istep + mmax;
                         double tempr_unrolled = wr * data[j_unrolled] - wi *
data[j_unrolled + 1];
                        double tempi_unrolled = wr * data[j_unrolled + 1] + wi *
data[j_unrolled];
                    data[j_unrolled] = data[i + istep] - tempr_unrolled;
                              data[j_unrolled + 1] = data[i + istep + 1] -
tempi_unrolled;
                    data[i + istep] += tempr_unrolled;
                    data[i + istep + 1] += tempi_unrolled;
                }
            }
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        mmax = istep;
    }
}
```