

## Skylar Buck

Each question is separated by screenshots. We start with question 1.

```
synth > app > Program.cs > {} app > app.Program > Main(string[] args)
37     writer.Write(0x20746D66); // = encoding.GetBytes("fmt ")
38     writer.Write(formatChunkSize);
39     writer.Write(formatType);
40     writer.Write(tracks);
41     writer.Write(samplesPerSecond);
42     writer.Write(bytesPerSecond);
43     writer.Write(frameSize);
44     writer.Write(bitsPerSample);
45     writer.Write(0x61746164); // = encoding.GetBytes("data")
46     writer.Write(dataChunkSize);
47     {
48         double theta = frequency * TAU / (double)samplesPerSecond;
49         double amp = volume >> 2;
50         for (int step = 0; step < samples; step++)
51         {
52             short s = (short)(amp * Math.Sin(theta * (double)step));
53             writer.Write(s);
54         }
55     }
56
57
58     writer.Close();
59     writer.Dispose();
60     SoundPlayer sp = new SoundPlayer("test.wav");
61     sp.Play();
62     // public static void PlayBeep(UInt16 frequency, int sDuration, UInt16 volume = 16383)
63 }
64 }
```

```
synth > app > Program.cs > {} app > app.Program > Main(string[] args)
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Media;
6
7  namespace app{
8      0 references
9      class Program{
10          0 references
11          static void Main(string[] args)
12          {
13              // if (args.Length!=0)
14              //     Console.WriteLine(args[0]);
15              // else
16              //     Console.WriteLine("nothing here");
17              UInt16 frequency=440;
18              int sDuration=3;
19              UInt16 volume = 16383;
20              File.Delete("test.wav");
21              FileStream f = new FileStream("test.wav", FileMode.Create);
22              BinaryWriter writer = new BinaryWriter(f);
23              const double TAU = 2 * Math.PI;
24              int formatChunkSize = 16;
25              int headerSize = 8;
26              short formatType = 1;
27              short tracks = 1;
28              int samplesPerSecond = 44100;
29              short bitsPerSample = 16;
30              short frameSize = (short)(tracks * ((bitsPerSample + 7) / 8));
31              int bytesPerSecond = samplesPerSecond * frameSize;
```

For question 1 we had to create audio for 3 seconds of a 440Hz sinusoid with a sampling rate of 44100 samples per second. The first thing i had to do was create a wav file so that I could play the sound from it. In order to do this in c# you have to write each individual thing. This includes the file size, file type, format, how many tracks there are, samples per second, bits per sample, ect.. I named this file test.wav and once all the writing to the file to make the skeleton is

complete we use formulas to compute theta and amp. Then using those we generate shorts that are written to the file. Each short is calculated using theta, amp, math.sin, and step(from 0 to number of samples). Once all of this is written to the file we can use SoundPlayer to run the audio for the file.

```

synth > app > C\Program.cs > {} app > app.Program > JingleBells()
    20     info=new string[] {"74",".3"};
    21     sinewave(info);
    22     info=new string[] {"74",".3"};
    23     sinewave(info);
    24     info=new string[] {"74",".6"};
    25     sinewave(info);
    26     info=new string[] {"69",".3"};
    27     sinewave(info);
    28     info=new string[] {"71",".3"};
    29     sinewave(info);
    30     info=new string[] {"74",".6"};
    31     sinewave(info);
    32     info=new string[] {"69",".15"};
    33     sinewave(info);
    34     info=new string[] {"71",".15"};
    35     sinewave(info);
    36     Thread.Sleep(500);
    37     info=new string[] {"71",".3"};
    38     sinewave(info);
    39     info=new string[] {"71",".3"};
    40     sinewave(info);
    41     info=new string[] {"71",".3"};
    42     sinewave(info);
    43     info=new string[] {"70",".15"};
    44     sinewave(info);
    45     Thread.Sleep(100);
    46     info=new string[] {"70",".15"};
    47     sinewave(info);
    48     Thread.Sleep(100);
    49     info=new string[] {"70",".15"};

```

```

synth > app > C\Program.cs > {} app > app.Program > JingleBells()
    70     info=new string[] {"69",".3"};
    71     sinewave(info);
    72 }
    73
    25 references
    74     static void sinewave(string[] args1)
    75     {
    76         UInt16 frequency=440;
    77         double sDuration=3;
    78         UInt16 volume = 50000;
    79         double TAU = 2 * Math.PI;
    80         if (args1.Length!=0){
    81             if(args1[0]=="JingleBells"){
    82                 Console.WriteLine("Playing JingleBells");
    83                 JingleBells();
    84                 return;
    85             }
    86             Console.WriteLine("MIDI note # is "+args1[0]);
    87             Console.WriteLine("Duration in seconds is "+args1[1]);
    88             sDuration=Convert.ToDouble(args1[1]);
    89             double power=Convert.ToDouble(args1[0]);
    90             TAU = Math.Pow(2, ((power-69)/12));
    91         }
    92         File.Delete("test.wav");
    93         FileStream f = new FileStream("test.wav", FileMode.Create);
    94         BinaryWriter writer = new BinaryWriter(f);
    95         int formatChunkSize = 16;
    96         int headerSize = 8;
    97         short formatType = 1;
    98         short tracks = 1;
    99         int samplesPerSecond = 44100;

```

For question 2 we had to create a function that takes as arguments a MIDI note number and a duration and similarly generates a sinusoidal sound. We also had to make a song. For the song

I chose to do the famous christmas carol “jingle bells” which is the function shown in the first screenshot. In the second screenshot we are using the command line inputs to either run the jingle bells song or alter frequency/duration based on what is put into the arguments. The formula is used to calculate TAU by changing the midi note into something we can use in order to generate the correct sound.

```

File Edit Selection View Go Run Terminal Help
Program.cs - csc - Visual Studio Code
Program.cs 9+ ×
synth > app > Program.cs > {} app > app.Program > sinewave(string[] args1)
0 references
1 reference
10 class Program{
11     static void score(string[] args2){
12     {
13         string[] info1={"69","",".3"};
14         for (int i=1; i<args2.Length ; i+=2){
15             if(args2[i]=="rest"){
16                 Thread.Sleep(Int32.Parse(args2[i+1]));
17                 Console.WriteLine("sleeping");
18             }else{
19                 info1=new string[] {args2[i], args2[i+1]};
20                 sinewave(info1);
21             }
22         }
23     }
1 reference
24     static void JingleBells()
25 }

PROBLEMS 18 OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
dotnet + ×
Time Elapsed 00:00:02.37
PS C:\Users\skyla\Downloads\csc\synth\app> dotnet run score 69 1 rest 500 69 2 rest 1000 79 1 rest 200 rest 300 61 1 70 1
Playing Score
MIDI note # is 69
Duration in seconds is 1
0.06268937721449433
sleeping
MIDI note # is 69
Duration in seconds is 2
0.06268937721449433
sleeping

```

For question 3 we can now run multiple notes in succession and include rests as well. For command line input it reads score, goes to this function and skips the first input(“score”). The loop that takes the command line as input will take the first input and will register that as either a note or as a rest with the second input as amount of time in seconds. Rest will sleep for the amount of seconds allotted while the note will play for the amount of seconds allotted. Then the loop will reset and look at the next two inputs until there are no more inputs to look at.

The screenshot shows the Visual Studio Code interface with the file 'Program.cs' open. The code implements a sine wave generator. It starts by writing header information to a file, including the number of bits per sample (4), the TAU constant (6.28318530718), and the data chunk size. It then enters a loop where it calculates a theta value based on frequency and step count, and selects an oscillator type (noise, pulse, triangle, sawtooth) using command line arguments. The selected oscillator is scaled by an amplitude and written to the file. Finally, the writer is closed.

```
synth > app > Program.cs > {} app > app.Program > sinewave(string[] args1)
243     writer.Write(bitsPerSample);
244     writer.Write(0x61746164); // = encoding.GetBytes("data")
245     writer.Write(dataChunkSize);
246     {
247         double theta = frequency * TAU / (double)samplesPerSecond;
248         if(args1.Length!=0)
249             | theta*= 6.28318530718;
250             | Console.WriteLine(theta);
251         double amp = volume >> 2;
252         for (int step = 0; step < samples; step++)
253         {
254             short s;
255             if(noise1==1){
256                 s = (short)(amp * noise(theta * (double)step));
257             }else if(pulse1==1){
258                 s = (short)(amp * pulse(theta * (double)step));
259             }else if(triangle1==1){
260                 s = (short)(amp * triangle(theta * (double)step));
261             }else if(sawtooth1==1){
262                 s = (short)(amp * sawtooth(theta * (double)step));
263             }else{
264                 s = (short)(amp * Math.Sin(theta * (double)step));
265             }
266
267             writer.Write(s);
268         }
269     }
270
271     writer.Close();
272
```

The screenshot shows the Visual Studio Code interface with the file 'Program.cs' open. It contains implementations for four oscillators: sawtooth, pulse, triangle, and noise. The sawtooth function returns the fractional part of the input x divided by PI. The pulse function returns the sign of the sawtooth(x) value. The triangle function returns 1 minus twice the absolute value of the sawtooth(x) value. The noise function generates a random double between 0 and 1 using a Random class.

```
8
9  namespace app{
10    0 references
11    class Program{
12        3 references
13        static double sawtooth(double x)
14        {
15            return (x - Math.PI)/Math.PI;
16        }
17        1 reference
18        static double pulse(double x)
19        {
20            return Math.Sign(sawtooth(x));
21        }
22        1 reference
23        static double triangle(double x)
24        {
25            return 1 - 2 * Math.Abs(sawtooth(x));
26        }
27        1 reference
28        static double noise(double x)
29        {
30            Random rnd = new Random();
31            double num = rnd.NextDouble();
32            return num;
33        }
34        1 reference
35        static void sample(string[] args1, string samplesnum)
36        {
37            UInt16 frequency=440;
38            double sDuration=3;
39        }
40    }
41}
```

For question 4 we had to Implement the following oscillators: Sawtooh, Pulse, Triangle and Noise. The 2nd screenshot shows the formulas for each of these oscillators as their own function that gets called in the step loop that gets written into the file. This step loop is shown in the first screenshot, the oscillator that gets chosen depends on what is determined in the command line arguments which declares which one to use and causes the variables to get ticked.

File Edit Selection View Go Run Terminal Help

Program.cs - csc - Visual Studio Code

Program.cs

```
synth > app > Program.cs > {} app > app.Program > sample(string[] args1, string samplesnum)

185     if(noise1==1){
186         s = (short)(amp * noise(theta * (double)step));
187     }else if(pulse1==1){
188         s = (short)(amp * pulse(theta * (double)step));
189     }else if(triangle1==1){
190         s = (short)(amp * triangle(theta * (double)step));
191     }else if(sawtooth1==1){
192         s = (short)(amp * sawtooth(theta * (double)step));
193     }else{
194         s = (short)(amp * Math.Sin(theta * (double)step));
195     }

196     writer.Write(s);
197 }
198
199 writer.Close();
200 writer.Dispose();
201 using (SoundPlayer player = new SoundPlayer("test.wav"))
202 {
203     Stopwatch timer = new Stopwatch();
204     timer.Start();
205     //Console.WriteLine("playing at " + (timer.ElapsedMilliseconds/1000) + " seconds");
206     //player.LoadAsync();
207     player.PlayLooping();
208     Thread.Sleep((int)(trueDuration*1000));
209     Console.WriteLine("played for " + (timer.ElapsedMilliseconds/1000) + " seconds");
210     timer.Stop();
211 }
212
213 }
214 }
```

Ln 105, Col 2 Spaces: 4 UTF-8 with BOM CRLF C# F5 D

0 0 0 Connect csc Server not selected

203 AM 2/6/2023

File Edit Selection View Go Run Terminal Help

Program.cs - csc - Visual Studio Code

Program.cs

```
synth > app > Program.cs > {} app > app.Program > sample(string[] args1, string samplesnum)
26 references
104 static void sample(string[] args1, string samplesnum)
105 {
106     double samplesnum1=Convert.ToDouble(samplesnum);
107     UInt16 frequency=440;
108     double sDuration=1;
109     double trueDuration=3;
110     UInt16 volume = 50000;
111     double TAU = 2 * Math.PI;
112     int pulse1=0;
113     int sawtooth1=0;
114     int triangle1=0;
115     int noise1=0;
116     int samplesPerSecond = (int)samplesnum1;
117     if (args1.Length!=0){
118         if(args1[0]=="pulse"){
119             pulse1=1;
120             args1 = args1.Skip(1).ToArray();
121         }
122         if(args1[0]=="sawtooth"){
123             sawtooth1=1;
124             args1 = args1.Skip(1).ToArray();
125         }
126         if(args1[0]=="triangle"){
127             triangle1=1;
128             args1 = args1.Skip(1).ToArray();
129         }
130         if(args1[0]=="noise"){
131             noise1=1;
132             args1 = args1.Skip(1).ToArray();
133         }
134     }
```

Ln 105, Col 2 Spaces: 4 UTF-8 with BOM CRLF C# F5 D

0 0 0 Connect csc Server not selected

202 AM 2/6/2023

File Edit Selection View Go Run Terminal Help

Program.cs - csc - Visual Studio Code

```
synth > app > Program.cs > {} app > app.Program > sample(string[] args1, string samplesnum)
28     return num;
29 }
1 reference
30 static void JingleBells1(string samplesnum)
31 {
32     string[] info={"69",".3"};
33     sample(info, samplesnum);
34     info=new string[] {"69", ".3"};
35     sample(info, samplesnum);
36     info=new string[] {"69",".6"};
37     sample(info, samplesnum);
38     info=new string[] {"74",".3"};
39     sample(info, samplesnum);
40     info=new string[] {"74",".3"};
41     sample(info, samplesnum);
42     info=new string[] {"74",".6"};
43     sample(info, samplesnum);
44     info=new string[] {"69",".3"};
45     sample(info, samplesnum);
46     info=new string[] {"71",".3"};
47     sample(info, samplesnum);
48     info=new string[] {"74",".6"};
49     sample(info, samplesnum);
50     info=new string[] {"69",".15"};
51     sample(info, samplesnum);
52     info=new string[] {"71",".15"};
53     sample(info, samplesnum);
54     Thread.Sleep(500);
55     info=new string[] {"71",".3"};
56     sample(info, samplesnum);
57     info=new string[] {"71",".3"};

```

Ln 119, Col 22 Spaces: 4 UTF-8 with BOM CRLF C# F5 D Server not selected

File Edit Selection View Go Run Terminal Help

Program.cs - csc - Visual Studio Code

```
synth > app > Program.cs > {} app > app.Program > sample(string[] args1, string samplesnum)
135         Console.WriteLine("Playing Score");
136         score1(args1, samplesnum);
137         return;
138     }
139     if(args1[0]=="JingleBells"){
140         Console.WriteLine("Playing JingleBells");
141         JingleBells1(samplesnum);
142         return;
143     }
144     Console.WriteLine("MIDI note # is "+args1[0]);
145     Console.WriteLine("Duration in seconds is "+args1[1]);
146     sDuration=Convert.ToDouble(args1[1]);
147     double power=Convert.ToDouble(args1[0]);
148     TAU = Math.Pow(2, ((power-69)/12));
149 }
File.Delete("test.wav");
FileStream f = new FileStream("test.wav", FileMode.Create);
BinaryWriter writer = new BinaryWriter(f);
int formatChunkSize = 16;
int headerSize = 8;
short formatType = 1;
short tracks = 1;
short bitsPerSample = 16;
short frameSize = (short)(tracks * ((bitsPerSample + 7) / 8));
int bytesPerSecond = samplesPerSecond * frameSize;
int waveSize = 4;
int samples = (int)(samplesPerSecond * sDuration);
int dataChunkSize = samples * frameSize;
int fileSize = waveSize + headerSize + formatChunkSize + headerSize + dataChunkSize;
writer.Write(0x46454952); // = encoding.GetBytes("RIFF")

```

Ln 119, Col 22 Spaces: 4 UTF-8 with BOM CRLF C# F5 D Server not selected

File Edit Selection View Go Run Terminal Help

Program.cs - Visual Studio Code

```
synth > app > Program.cs > {} app > app.Program > sample(string[] args1, string samplesnum)
    info=new string[] {"73",".15"};
    sample(info, samplesnum);
    Thread.Sleep(100);
    info=new string[] {"71",".15"};
    sample(info, samplesnum);
    Thread.Sleep(100);
    info=new string[] {"69",".3"};
    sample(info, samplesnum);
}
1 reference
static void score1(string[] args2, string samplesnum)
{
    string[] info1={"69","");
    for (int i=1; i<args2.Length ; i+=2){
        if(args2[i]=="rest"){
            Thread.Sleep(Int32.Parse(args2[i+1]));
            Console.WriteLine("sleeping");
        }else{
            info1=new string[] {args2[i], args2[i+1]};
            sample(info1, samplesnum);
        }
    }
}
26 references
static void sample(string[] args1, string samplesnum)
{
    double samplesnum1=Convert.ToDouble(samplesnum);
    UInt16 frequency=440;
    double sDuration=1;
    double trueDuration=3;
    UInt16 volume = 50000;
}
```

Ln 119, Col 22 Spaces: 4 UTF-8 with BOM CRLF C# F# D 2:08 AM 2/6/2023

File Edit Selection View Go Run Terminal Help

Program.cs - Visual Studio Code

```
synth > app > Program.cs > {} app > app.Program > sample(string[] args1, string samplesnum)
writer.Write(frameSize);
writer.Write(bitsPerSample);
writer.Write(0x61746164); // = encoding.GetBytes("data")
writer.Write(dataChunkSize);
{
    double theta = frequency * TAU / (double)samplesPerSecond;
    if(args1.Length!=0)
        theta*= 6.28318530718;
    Console.WriteLine(theta);
    double amp = volume >> 2;
    for (int step = 0; step < samples; step++)
    {
        short s;
        if(noise1==1){
            s = (short)(amp * noise(theta * (double)step));
        }else if(pulse1==1){
            s = (short)(amp * pulse(theta * (double)step));
        }else if(triangle1==1){
            s = (short)(amp * triangle(theta * (double)step));
        }else if(sawtooth1==1){
            s = (short)(amp * sawtooth(theta * (double)step));
        }else{
            s = (short)(amp * Math.Sin(theta * (double)step));
        }

        writer.Write(s);
    }
    writer.Close();
    writer.Dispose();
}
```

Ln 119, Col 22 Spaces: 4 UTF-8 with BOM CRLF C# F# D 2:09 AM 2/6/2023

```
synth > app > Program.cs > {} app > app.Program > sample(string[] args1, string samplesnum)
114     int triangle1=0;
115     int noise1=0;
116     int samplesPerSecond = (int)samplesnum1;
117     if(args1.Length!=0){
118         if(args1[0]=="pulse"){
119             pulse1=1;
120             args1 = args1.Skip(1).ToArray();
121         }
122         if(args1[0]=="sawtooth"){
123             sawtooth1=1;
124             args1 = args1.Skip(1).ToArray();
125         }
126         if(args1[0]=="triangle"){
127             triangle1=1;
128             args1 = args1.Skip(1).ToArray();
129         }
130         if(args1[0]=="noise"){
131             noise1=1;
132             args1 = args1.Skip(1).ToArray();
133         }
134         if(args1[0]=="score"){
135             Console.WriteLine("Playing Score");
136             score1(args1, samplesnum);
137             return;
138         }
139         if(args1[0]=="JingleBells"){
140             Console.WriteLine("Playing JingleBells");
141             JingleBells1(samplesnum);
142             return;
143     }
```

```
writer.Dispose();
using (SoundPlayer player = new SoundPlayer("test.wav"))
{
    player.PlaySync();
}

static void Main(string[] args)
{
    if (args.Length!=0){
        if(args[0]=="sample"){
            Console.WriteLine("Playing "+args[1]+" samples.");
            //Console.WriteLine(args[0]);
            args = args.Skip(1).ToArray();
            string samplesnum = args[0];
            args = args.Skip(1).ToArray();
            sample(args, samplesnum);
            return;
        }
    }
    sinewave(args);
}
```

For question 5 we had to re-implement questions 1,2,3,4 but using buffers. I decided to create a sinusoid for 1 second then my oscillator function will be called multiple times to generate a user-provided number of samples. To do this i essentially copied the whole file and altered it to make only a 1 second buffer of sound then I would replay the sound for the amount of time that is needed, even if the sound is need to be 0.5 seconds or 2.5 seconds of duration. For command line arguments by default it takes 2 which is the word sample followed by how many samples they would like to play. Any command line

arguments past that go into the questions as MIDI note number and a duration as well as jingle bells for question 2, score(string of many notes and rests) for question 3, and different oscillators for question 4. Question 1 is re-implemented in screenshot 8, screenshot 1, screenshot 2, and screenshot 4. Question 2 is re-implemented in screenshot 3 and screenshot 4. Question 3 is re-implemented in screenshot 4 and screenshot 5. Question 4 is re-implemented in screenshot 7, screenshot 6, and screenshot 2.

```

File Edit Selection View Go Run Terminal Help Program.cs - csc - Visual Studio Code
Program.cs X app.csproj
synth > app > Program.cs > {} app > app.Program > secondsound(string[] args1)
446 | writer.Dispose();
447 }
1 reference
448 static void firstsound(string[] args1)
449 {
450     UInt16 frequency=440;
451     double sDuration=3;
452     UInt16 volume = 50000;
453     double TAU = 2 * Math.PI;
454     int pulse1=0;
455     int sawtooth1=0;
456     int triangle1=0;
457     int noise1=0;
458     if (args1.Length!=0){
459         if(args1[0]=="pulse"){
460             pulse1=1;
461             args1 = args1.Skip(1).ToArray();
462         }
463         if(args1[0]=="sawtooth"){
464             sawtooth1=1;
465             args1 = args1.Skip(1).ToArray();
466         }
467         if(args1[0]=="triangle"){
468             triangle1=1;
469             args1 = args1.Skip(1).ToArray();
470         }
471         if(args1[0]=="noise"){
472             noise1=1;
473             args1 = args1.Skip(1).ToArray();
474         }
475     }
476     if(args1[0]=="score"){
477         sinewave(info);
478     }
479 }
Ln 392, Col 47 Spaces: 4 UTF-8 with BOM CRLF 4:41 AM 2/6/2023
.NET Core Launch (console) (csc) Connect csc Server not selected

```

```

File Edit Selection View Go Run Terminal Help Program.cs - csc - Visual Studio Code
Program.cs X app.csproj
synth > app > Program.cs > {} app > app.Program > secondsound(string[] args1)
346 | sinewave(info);
347 }
1 reference
348 static void secondsound(string[] args1)
349 {
350     UInt16 frequency=440;
351     double sDuration=3;
352     UInt16 volume = 50000;
353     double TAU = 2 * Math.PI;
354     int pulse1=0;
355     int sawtooth1=0;
356     int triangle1=0;
357     int noise1=0;
358     if (args1.Length!=0){
359         if(args1[0]=="pulse"){
360             pulse1=1;
361             args1 = args1.Skip(1).ToArray();
362         }
363         if(args1[0]=="sawtooth"){
364             sawtooth1=1;
365             args1 = args1.Skip(1).ToArray();
366         }
367         if(args1[0]=="triangle"){
368             triangle1=1;
369             args1 = args1.Skip(1).ToArray();
370         }
371         if(args1[0]=="noise"){
372             noise1=1;
373             args1 = args1.Skip(1).ToArray();
374         }
375     }
376     if(args1[0]=="score"){
377         sinewave(info);
378     }
379 }
Ln 392, Col 47 Spaces: 4 UTF-8 with BOM CRLF 4:41 AM 2/6/2023
.NET Core Launch (console) (csc) Connect csc Server not selected

```

The screenshot shows the Visual Studio Code interface with a C# file named `Program.cs` open. The code implements two methods for playing audio files using `WaveStream` and `WaveOutEvent`.

```
synth > app > Program.cs > {} app > app.Program > secondsound(string[] args1)

10
11     namespace app{
12         0 references
13         class Program{
14             1 reference
15             static void twosounds(){
16                 WaveStream stream1= new AudioFileReader("firstsound.wav");
17                 WaveOutEvent out1 = new();
18                 out1.Init(stream1);
19                 stream1.CurrentTime = new TimeSpan(0L);
20                 out1.Play();
21
22                 WaveStream stream2= new AudioFileReader("secondsound.wav");
23                 WaveOutEvent out2 = new();
24                 out2.Init(stream2);
25                 stream2.CurrentTime = new TimeSpan(0L);
26                 out2.Play();
27
28                 Console.ReadKey();
29             }
29
30             1 reference
31             static void howmany(){
32                 WaveStream stream1= new AudioFileReader("howmany.wav");
33                 WaveOutEvent out1 = new();
34                 out1.Init(stream1);
35                 stream1.CurrentTime = new TimeSpan(0L);
36                 out1.Play();
37
}
37
```

The status bar at the bottom indicates the file is `csc`, the encoding is `UTF-8 with BOM`, and the line and column numbers are `Ln 392, Col 47`. It also shows the date and time as `2/6/2023 4:41 AM`.

Program.cs - csc - Visual Studio Code

```
synth > app > Program.cs > {} app > app.Program > howmany()
```

```
13 static void howmany(){
```

```
14 |
```

```
15     WaveStream stream1= new AudioFileReader("howmany.wav");
```

```
16     WaveOutEvent out1 = new();
```

```
17     out1.Init(stream1);
```

```
18     stream1.CurrentTime = new TimeSpan(0L);
```

```
19     out1.Play();
```

```
20
```

```
21     WaveStream stream2= new AudioFileReader("howmany.wav");
```

```
22     WaveOutEvent out2 = new();
```

```
23     out2.Init(stream2);
```

```
24     stream2.CurrentTime = new TimeSpan(0L);
```

```
25     out2.Play();
```

```
26
```

```
27     WaveStream stream3= new AudioFileReader("howmany.wav");
```

```
28     WaveOutEvent out3 = new();
```

```
29     out3.Init(stream3);
```

```
30     stream3.CurrentTime = new TimeSpan(0L);
```

```
31     out3.Play();
```

```
32
```

```
33     WaveStream stream4= new AudioFileReader("howmany.wav");
```

```
34     WaveOutEvent out4 = new();
```

```
35     out4.Init(stream4);
```

```
36     stream4.CurrentTime = new TimeSpan(0L);
```

```
37     out4.Play();
```

```
38
```

```
39 // WaveStream stream5= new AudioFileReader("howmany.wav");
```

```
40 // WaveOutEvent out5 = new();
```

```
41 // out5.Init(stream5);
```

```
42 // stream5.CurrentTime = new TimeSpan(0L);
```

Ln 42, Col 5 Spaces: 4 UTF-8 with BOM CRLF C# F D

Program.cs - csc - Visual Studio Code

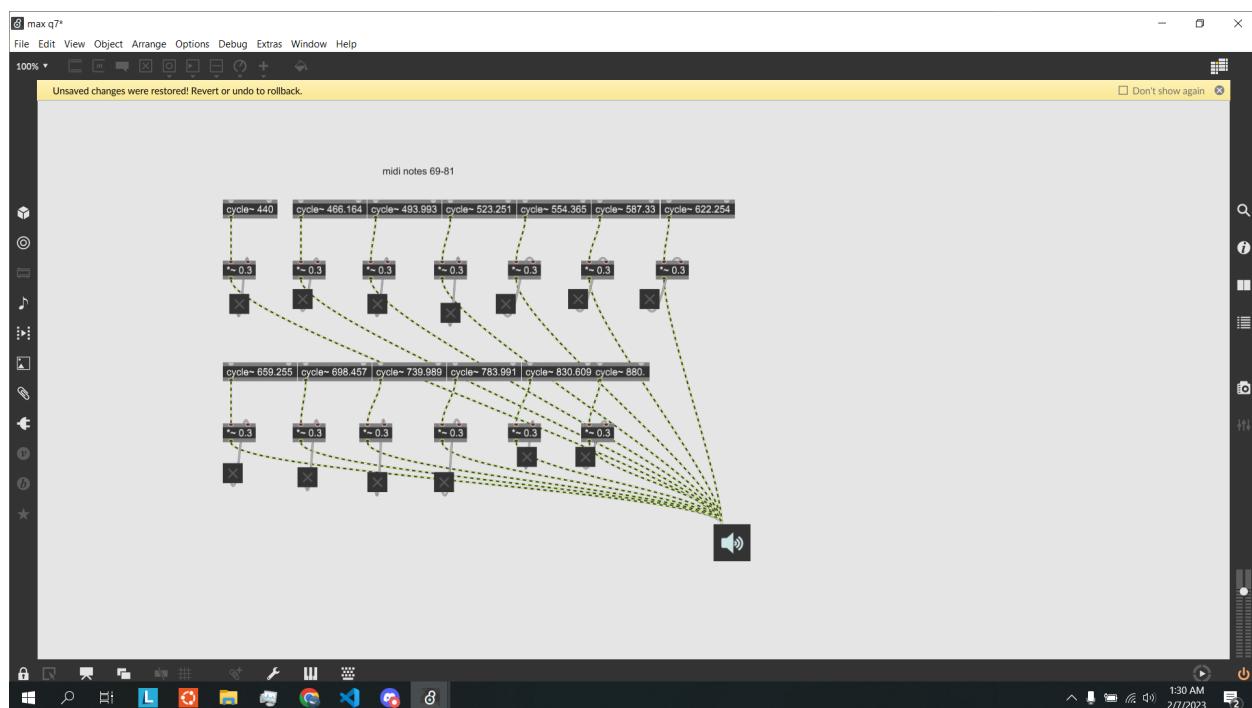
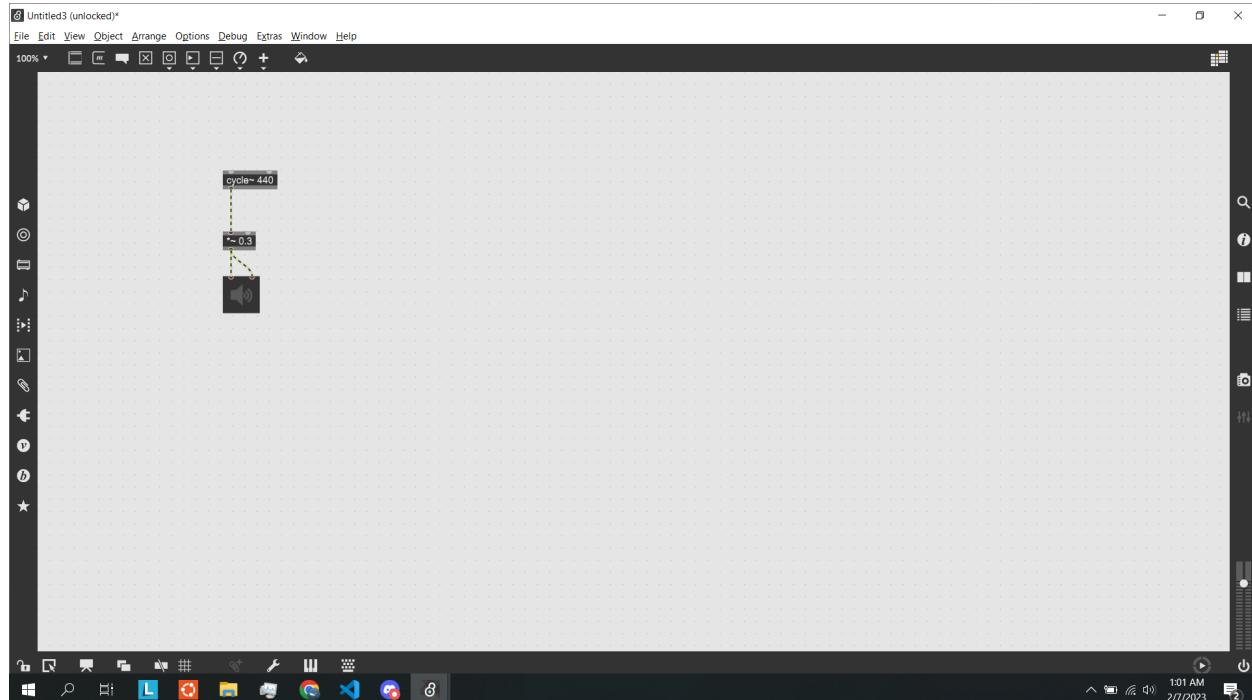
```
synth > app > Program.cs > {} app > app.Program > secondsound(string[] args)
```

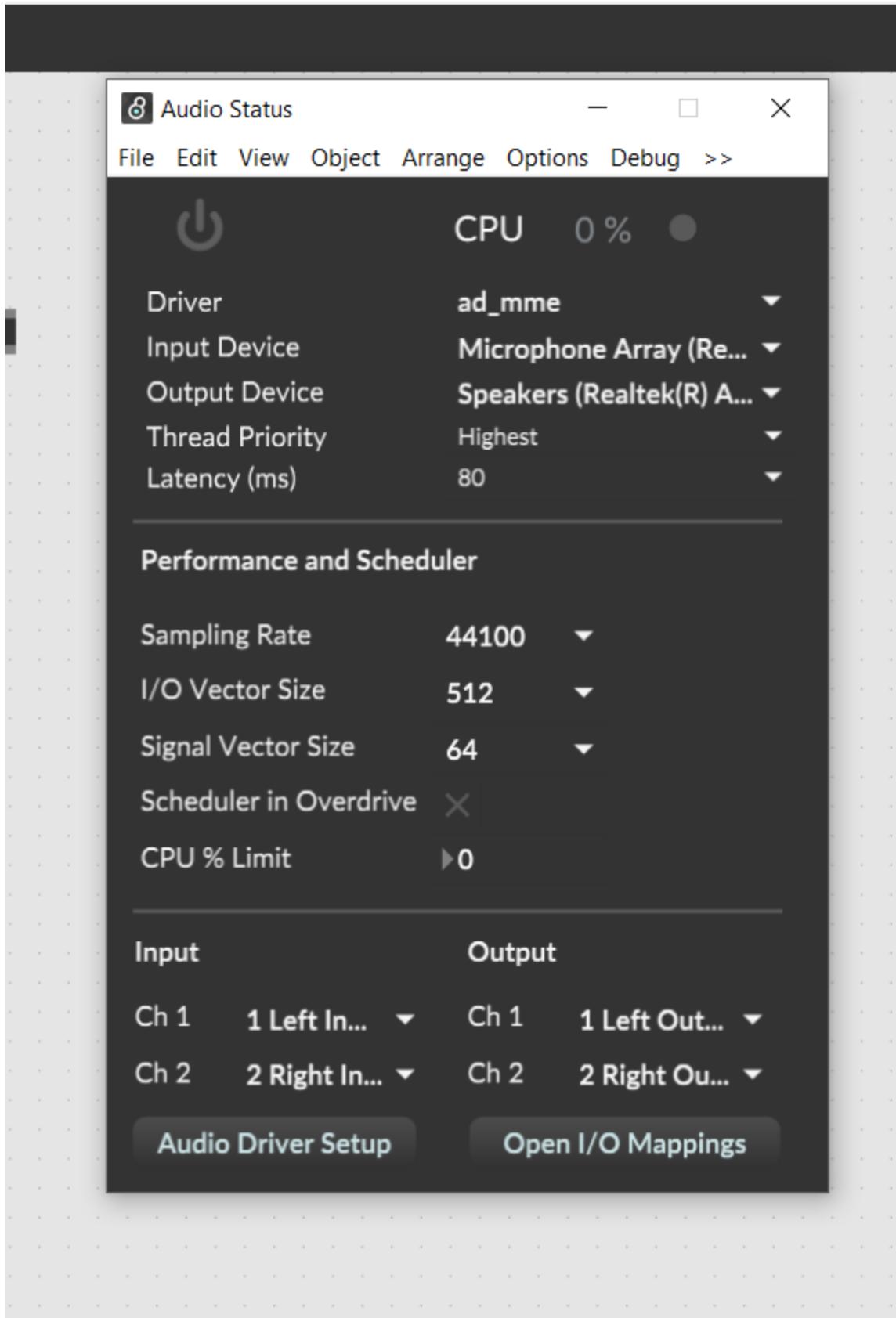
```
655     if(args[0]=="sample"){
656         Console.WriteLine("Playing "+args[1]+" samples.");
657         //Console.WriteLine(args[0]);
658         args = args.Skip(1).ToArray();
659         string samplesnum = args[0];
660         args = args.Skip(1).ToArray();
661         sample(args, samplesnum);
662         return;
663     }
664     if(args[0]=="howmany"){
665         howmany();
666         return;
667     }
668     if(args[0]=="poly"){
669         string[] first={args[1], args[2]};
670         string[] second={args[3], args[4]};
671         firstsound(first);
672         secondsound(second);
673         twosounds();
674         return;
675     }
676
677     sinewave(args);
678 }
680 }
```

Ln 680, Col 47 Spaces: 4 UTF-8 with BOM CRLF C# F D

For question 6 i had to make my program support polyphony and multiple oscillators. I implemented polyphony using a fixed number of oscillators (one for each voice - 2). Then I checked to see what level of polyphony I can support with your implementation on my laptop until it could not keep up with real time and the sound did not sound as one. I created functions for outputting different files using individual notes/durations as input in order to play them at the same time and get different notes combined together. Then I have a different function that will play the sounds together for their possibly

different durations. To see how many oscillators I could play at once without distorting the thought of real time I used a file that would beep once every other second so that I could register the distortion in sound. I was able to run 3 files at the same time without noticing but once I ran the 4th I noticed a small sound difference, it sounded like the beeps were vibrating a little bit which is a bad description but im not sure how to describe it.





Pitch	Frequency [Hz]	MIDI
A4	440.000	69
A#4	466.164	70
B4	493.883	71
C5	523.251	72
C#5	554.365	73
D5	587.330	74
D#5	<b>622.254</b>	75
E5	659.255	76
F5	698.457	77
F#5	739.989	78
G5	783.991	79
G#5	830.609	80
A5	880.000	81

For question 7 I had to do questions 1 and 2 using max/msp. The first question is shown in the first screenshot where i used cycle~ at a frequency of 440 in order to play a sound. This sound was loud so I lowered the volume using \*~ .3. The sample rate is by default 44100 as shown in screenshot 3. For question 2 i used 13 different notes shown in screenshot 4 with their corresponding frequencies they call connect to 1 audio output so I used a toggle button that gives either 0 or 1 to the volume essentially muting the sound until i press the buttons to play notes individually or together. I am able to play a simple melody(jingle bells) by switching through different toggle buttons turning the sound on and off.

```
synth > chuck > chuck.ck
176     <<<"midi:","midi," sec:" ,sec>>>;
177     SinOsc s => dac;
178     Std.mtof(midi)>= s.freq;
179     0.5>s.gain;
180     // allow 2 seconds to pass
181     //automatically at 44100Hz
182     sec::second => now;
183     i++;
184   }
185 }
186 }else{
187   // connect sine oscillator to D/A convertor (sound card)
188   int midi;
189   69=>midi;
190   float sec;
191   3=>sec;
192   SinOsc s => dac;
193   Std.mtof(midi)>= s.freq;
194   0.5>s.gain;
195   // allow 2 seconds to pass
196   //automatically at 44100Hz
197   sec::second => now;
198 }
199 ]
```

```
synth > chuck > chuck.ck
167     0.5>s.gain;
168     0.0>s.gain;
169     .1::second => now;
170     0.5>s.gain;
171     //sleep
172     Std.mtof(69)>= s.freq;
173     .3::second => now;
174 }else{
175   for( int i; i < me.args(); i++ )
176   {
177     Std.atoi(me.arg(i))=>midi;
178     Std.atof(me.arg(i+1))=>sec;
179     <<<"midi:","midi," sec:" ,sec>>>;
180     SinOsc s => dac;
181     Std.mtof(midi)>= s.freq;
182     0.5>s.gain;
183     // allow 2 seconds to pass
184     //automatically at 44100Hz
185     sec::second => now;
186     i++;
187   }
188 }
189 }else{
190   // connect sine oscillator to D/A convertor (sound card)
191   SinOsc s => dac;
192   Std.mtof(midi)>= s.freq;
193   0.5>s.gain;
194   // allow 2 seconds to pass
195   //automatically at 44100Hz
196   sec::second => now;
197 }
```

A screenshot of Visual Studio Code interface showing a Chuck C# project. The left sidebar contains icons for file operations like Open, Save, Find, and Run. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar shows "chuck.cs - csc - Visual Studio Code". The main editor area displays the code for "chuck.cs", which includes a C# class definition and several methods. The status bar at the bottom shows "Ln 182, Col 25" and "Spaces: 4" along with other system status indicators.

```
synth > chuck >  chuck.cs
8  3=>s;
9  if(me.args()>0)
10 {
11     <<< "number of arguments:", me.args() >>>;
12     if(me.arg(0)=="jinglebells"){
13         SinOsc s => dac;
14         0.5=>s.gain;
15         Std.mtof(69)>> s.freq;
16         .3::second => now;
17         0.0=>s.gain;
18         .1::second => now;
19         0.5=>s.gain;
20         Std.mtof(69)>> s.freq;
21         .3::second => now;
22         0.0=>s.gain;
23         .1::second => now;
24         0.5=>s.gain;
25         Std.mtof(69)>> s.freq;
26         .6::second => now;
27         0.0=>s.gain;
28         .1::second => now;
29         0.5=>s.gain;
30         Std.mtof(74)>> s.freq;
31         .3::second => now;
32         0.0=>s.gain;
33         .1::second => now;
34         0.5=>s.gain;
35         Std.mtof(74)>> s.freq;
36         .3::second => now;
37         0.0=>s.gain;
```

Screenshot 1 shows the Visual Studio Code interface with the file 'chuck.ck' open. The code defines a function that takes command-line arguments. It uses `Std.mtof` to convert strings to floats. It then initializes a `SinOsc` object named `s` with `dac` as its output. A `for` loop iterates through the arguments, setting the frequency (`s.freq`) and gain (`s.gain`) based on the argument value. If the argument is "rest", it sets the frequency to 0.0 and gain to 0.0. If the argument is a MIDI note, it converts it to a frequency using `Std.atof` and sets the frequency and gain. The loop continues until all arguments are processed. Finally, it connects the oscillator to the DAC.

```
synth > chuck > //sleep
        Std.mtof(69)> s.freq;
        .3::second => now;
    }else{
        SinOsc s => dac;
        for( int i; i < me.args(); i++ )
        {

            if(me.arg(i)=="rest"){
                Std.atof(me.arg(i+1))>sec;
                Std.mtof(0)> s.freq;
                <<<"midi: rest sec:",sec>>>;
                0.0=>s.gain;
            }else{
                Std atoi(me.arg(i))>midi;
                Std.atof(me.arg(i+1))>sec;
                <<<"midi:",midi," sec:",sec>>>;
                Std.mtof(midi)> s.freq;
                0.5=>s.gain;
            }
            // allow 2 seconds to pass
            //automatically at 44100Hz
            sec::second => now;
            i++;
        }
    }else{
        // connect sine oscillator to D/A convertor (sound card)
        SinOsc s => dac;
        Std.mtof(midi)> s.freq;
    }
}

```

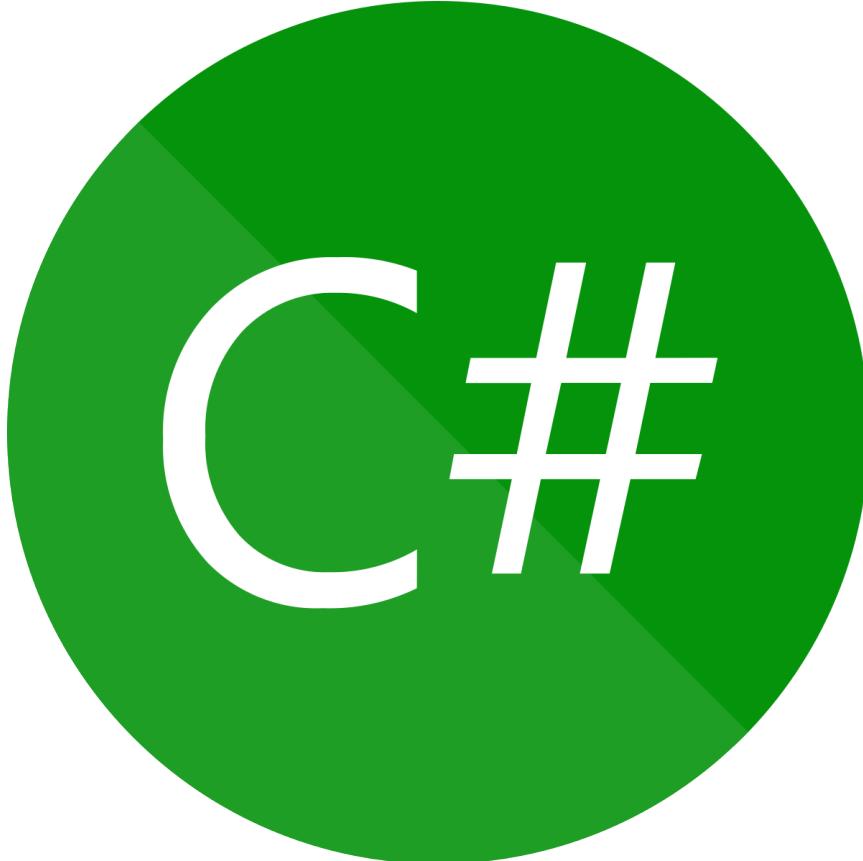
Screenshot 2 shows the Visual Studio Code interface with the file 'chuck.ck' open. The code defines a function that takes command-line arguments. It checks if the first argument is "sawtooth". If true, it initializes a `SawOsc` object named `s` with `dac` as its output. It then allows 3 seconds to pass automatically at 44100Hz. It then checks if the second argument is "triangle". If true, it initializes a `TriOsc` object named `s` with `dac` as its output. It then allows 3 seconds to pass automatically at 44100Hz. It then checks if the third argument is "pulse". If true, it initializes a `PulseOsc` object named `s` with `dac` as its output. It then allows 3 seconds to pass automatically at 44100Hz. It then checks if the fourth argument is "noise". If true, it initializes a `Noise` object named `s` with `dac` as its output. It then allows 3 seconds to pass automatically at 44100Hz. Finally, it connects the oscillator to the DAC.

```
if(me.arg(0)=="sawtooth"){
    SawOsc s => dac;
    Std.mtof(midi)> s.freq;
    0.5=>s.gain;
    // allow 3 seconds to pass
    //automatically at 44100Hz
    sec::second => now;
}
else if(me.arg(0)=="triangle"){
    TriOsc s => dac;
    Std.mtof(midi)> s.freq;
    0.5=>s.gain;
    // allow 3 seconds to pass
    //automatically at 44100Hz
    sec::second => now;
}
else if(me.arg(0)=="pulse"){
    PulseOsc s => dac;
    Std.mtof(midi)> s.freq;
    0.5=>s.gain;
    // allow 3 seconds to pass
    //automatically at 44100Hz
    sec::second => now;
}
else if(me.arg(0)=="noise"){
    Noise s => dac;
    Std.mtof(midi)> s.freq;
    0.5=>s.gain;
    // allow 3 seconds to pass
    //automatically at 44100Hz
    sec::second => now;
}

```

For question 8 i had to re-implement questions 1,2,3,4 using ChuckK. Question 1 is shown in screenshot 1 using `SinOsc` we can play sound by definition freq and gain then running the `x::second => now` to define how many seconds we want to use. Question 2 is shown in both screenshot 2 and screenshot 3 where i play the jingle bells simple melody and allow the code to take command line argument to play a midi note for a duration. Question 3 is shown in screenshot 4 where it will take command line arguments to play rest for x duration which is just oscillator playing sound for x seconds

on mute and it will also take many notes/rests in succession and will play them in order. Question 4 is shown in screenshot 5 and will play different Oscillators (sawtooth, noise, pulse, triangle) using different Osc variables.



Question 9 - For this assignment I started using haskell which is a language i had never used before and it was very confusing because I was getting errors when other people also using haskell were not getting the same errors i was with the same code. This was frustrating for me so after finishing question 1 in haskell i swapped to c# to do question 9 because even though I had never coded/used before just like haskell, I had used C so i thought it might be similar.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** On the left, showing files sine1.py, sine2.py, and sine3.py.
- Code Editor:** The main area contains a Python script named sine1.py. The code uses the tracemalloc module to measure memory usage. It prints the current time before and after execution, calculates the total time, and prints the total memory used.
- Terminal:** A terminal window on the right shows the output of running the script. The output includes the start and end times, the total time taken, and the total memory used, which is 0.000256061539550781 bytes.
- Status Bar:** At the bottom, it shows the file is 33 lines long, the current column is 2, and the current line is 7. It also indicates the file is saved in UTF-8 encoding and is a Python 3.11.1 64-bit file.

For question 10 I got 3 different ways of calculating sine in python without the math.sin() function. I calculated the time each file took using the time import and the memory using the tracemalloc import. For the first sine file I used the formula of  $2.718281828459045^{**}(x*1j).imag$  to calculate sine of a random number (in this case 42). In the print statement the first value is the true value of sine(42), the second value is the value of the formula(42), the third value is the amount of time it took, and the fourth/fifth line is the amount of memory used. For the second sine file I used the formula of Chebyshev to calculate sine of a random number (in this case 0.1). In the print statement the first value is the true value of sine(0.1), the second value is the value of the ch.eval(0.1), the third value is the amount of time it took, and the fourth/fifth line is the amount of memory used. For the third sine file I used the formula of sine approximation using 2Pi, step, angle, precos, theta, and presin in a combination using a while loop in order to calculate sine of a random number (in this case 1). In the print statement the first value is the true value of SineApproximation(1), the second value is the value of the sine(1), the third value is the amount of time it took, and the fourth/fifth line is the amount of memory used. In terms of time taken for execution the order for least to greatest time taken is third sine file, first sine file, second sine file. In terms of memory used for execution the order for least to greatest memory used is first sine file, third sine file, second sine file.