```
}



ChucK – Harmonic Progression Logic
//Author - Mo Washburn for EBP 2019
 SndBuf pad_root => dac;
 SndBuf pad_third => dac;
 SndBuf pad_fifth => dac;
 SndBuf pad_seventh => dac;
 SndBuf pad_octave => dac;
 SndBuf pad_ninth => dac;
 SndBuf pad_extension => dac;
 Gain bal[2];
 pad_root.chan(0) => bal[0] => dac.left;
 pad_third.chan(0) => bal[0] => dac.left;
 pad_fifth.chan(0) => bal[0] => dac.left;
 pad_seventh.chan(0) => bal[0] => dac.left;
 pad_octave.chan(0) => bal[0] => dac.left;
 pad_ninth.chan(0) => bal[0] => dac.left;
 pad_extension(0) => bal[0] => dac.left;

 pad_root.chan(1) => bal[1] => dac.right;
 pad_third.chan(1) => bal[1] => dac.right;
 pad_fifth.chan(1) => bal[1] => dac.right;
 pad_seventh.chan(1) => bal[1] => dac.right;
 pad_octave.chan(1) => bal[1] => dac.right;
 pad_ninth.chan(1) => bal[1] => dac.right;
 pad_extension(1) => bal[1] => dac.right;




 /*
 ChuckSubInstance myChuck;
 Chuck.IntCallback myIntCallback;
 long latestGottenInt;

 void Start()
 {
    myChuck = GetComponent<ChuckSubInstance>();
    myChuck.RunCode(@"
        global int myGlobalInt;
    ");
    myIntCallback = myChuck.CreateGetIntCallback( MyGetIntCallbackFunction );

    myChuck.SetInt( "myGlobalInt", 0 );
 }

 void Update()
```

```
{
   myChuck.GetInt( "myGlobalInt", myIntCallback );
   // DO SOMETHING WITH THE GOTTEN VALUE

}

void MyGetIntCallbackFunction( long newValue )
{
   latestGottenInt = newValue;
}
*/
// READ IN DATA
FileIO pressure_fio;
"C:/data/pressure_data_output.csv" => string pressure_file;
pressure_fio.open( pressure_file, FileIO.READ );

// verify file loaded correctly
if( !pressure_fio.good() )
{
   cherr <= "Failed to read file(s)."
         <= IO.newline();
   me.exit();
}

float pressure[937];

0 => int index;
while( pressure_fio.good() )
{
   Std.atof(pressure_fio.readLine()) => float datum_pressure;
   datum_pressure => pressure[index];

   1 +=> index;
   if( pressure_fio.eof() ) break;
}
```

```
// INSTRUMENT HANDLING
// instantiate directories for accessing samples
me.dir() => string base_dir;
// <<< base_dir >>>;
if (base_dir.charAt(base_dir.length()-1) != '/')
{
    "/" +=> base_dir;
}

// for accessing soundfonts form Github repository
"ChucK_Audio/" => string subfolder;
// pick your instrument
"organ/" => string instrument;

base_dir + subfolder + instrument => string file_path;

["C", "Db", "D", "Eb", "E", "F", "Gb", "G", "Ab", "A", "Bb", "B"] @=> string notes[];
string lower_scale[12];
string upper_scale[12];
for(0 => int i; i<24; i++)
{
    if(i < 12)
    {
        "2" => string octave;
        file_path + notes[i] + octave + ".wav" => lower_scale[i];
    }
    if(i >= 12)
    {
        "3" => string octave;
        file_path + notes[i-12] + octave + ".wav" => upper_scale[i-12];
    }
}


// SONG EXECUTION

8 => float length;
1.0 => float volume;
/*
bool SetInt( "deceptive_cadence", long 0);
bool SetInt( "tritone_sub", long 0);
bool SetInt( "substitution", long 0);
bool SetInt( "matrix", long 0);
*/
```

```
int deceptive_cadence;
int tritone_sub;
int substitution;
int matrix;
int deceptive_cadence_2;
int tritone_sub_2;
int substitution_2;
int matrix_2;
int deceptive_cadence_3;
int tritone_sub_3;
int substitution_3;
int matrix_3;
int deceptive_cadence_4;
int tritone_sub_4;
int substitution_4;
int matrix_4;

0 => int tick;
for(1 => int i; i<5; i++)
{

    for(tick; tick < (936/8)*i; tick++)
    {
        if(pressure[tick] > 0.75 && pressure[tick] <= 1.0)
        {
            <<< "one" >>>;
            0 => int deceptive_cadence;
            0 => int tritone_sub;
            0 => int substitution;
            0 => int matrix;
        }
        if(pressure[tick] > 0.5 && pressure[tick] <= 0.75)
        {
            1 => int deceptive_cadence_2;
            <<< deceptive_cadence_2 >>>;
            if(deceptive_cadence_2 == 1)
            {
                <<< "Deceptive activated" >>>;
            }
            0 => int tritone_sub_2;
            0 => int substitution_2;
            0 => int matrix_2;
        }
        if(pressure[tick] > 0.25 && pressure[tick] <= 0.5)
        {
            <<< "three" >>>;
            1 => int deceptive_cadence_3;
            1 => int tritone_sub_3;
            1 => int substitution_3;
            0 => int matrix_3;
        }
        if(pressure[tick] > 0 && pressure[tick] <= 0.25)
        {
            <<< "four" >>>;
            1 => int deceptive_cadence_4;
            1 => int tritone_sub_4;
            1 => int substitution_4;
            1 => int matrix_4;
        }
    }
    <<< deceptive_cadence_2 >>>;

// UPDATE CHORD PROGRESSION DECISION LOGIC
    for(1 => int a; a < 5; a++)
    {
        0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_octave.pos;
        volume => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_octave.gain;
        0 => pad_seventh.gain => pad_ninth.gain => pad_extension.gain;
        lower_scale[0] => pad_root.read;
        lower_scale[4] => pad_third.read;
        lower_scale[7] => pad_fifth.read;
        if(a == 3 && matrix == 0)
        {
            lower_scale[11] => pad_octave.read;
        }
        if(a == 3 && matrix == 1)
        {
            four_six(length, volume, substitution, tritone_sub, deceptive_cadence);
        }
        else
        {
            upper_scale[0] => pad_octave.read;
            2 :: second => now;
        }
        if(a == 2)
        {
            three_six(length, volume, substitution, substitution_3, tritone_sub,
deceptive_cadence_2, tritone_sub_3);
        }
    }
}


// CHORD PROGRESSION FUNCTIONS
// length parameter should be in seconds
// how length is determined is tbd
// vol parameter is volume(gain)



// ii-V7 PROGRESSION
// TRITONE SUB OPTION
// ["C", "Db", "D", "Eb", "E", "F", "Gb", "G", "Ab", "A", "Bb", "B"]
fun void two_five( float length, float vol, int tritone, int deceptive_2, int tritone_3)
{
    <<< "Two-five reached." >>>;
    vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain =>
pad_octave.gain => pad_ninth.gain => pad_extension.gain;
    for(1 => int i; i <= (length/4); i++)
    {
        <<< "d-" >>>;
        0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_octave.pos =>
pad_extension.pos;
        vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_octave.gain =>
pad_extension.gain;
        0 => pad_seventh.gain => pad_ninth.gain;
        lower_scale[2] => pad_root.read;
        lower_scale[5] => pad_third.read;
        lower_scale[9] => pad_fifth.read;
        upper_scale[9] => pad_extension.read;
        upper_scale[2] => pad_octave.read;

        2 :: second => now;
    }
    //TRITONE SUB DECISION
    if(tritone == 0){
        for(1 => int t; t <= (length/4); t++)
        {
            <<< "V7 reached." >>>;
            0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos;
            vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain;
            0 => pad_octave.gain => pad_ninth.gain => pad_extension.gain;
            lower_scale[7] => pad_root.read;
            lower_scale[11] => pad_third.read;
            upper_scale[2] => pad_fifth.read;
            lower_scale[5] => pad_seventh.read;

            2 :: second => now;
        }
    }
    if(tritone_3 == 1)
    {
        <<< "Tritone sub reached." >>>;
        for(1 => int t; t <= (length/4); t++)
        {
            <<< "C#7" >>>;
            0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos =>
pad_octave.pos;
            vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain =>
pad_octave.gain;
            0 => pad_ninth.gain => pad_extension.gain;
            lower_scale[1] => pad_root.read;
            lower_scale[11] => pad_third.read;
            lower_scale[8] => pad_fifth.read;
            lower_scale[5] => pad_seventh.read;
            upper_scale[8] => pad_octave.read;

            2 :: second => now;
        }
    }
    // DECEPTIVE CADENCE DECISION
    <<< deceptive_2 >>>;
    <<< tritone >>>;
    if(deceptive_2 == 1 && tritone == 0)
    {
        <<< "Deceptive cadence reached." >>>;
        for(1 => int d; d < 3; d++)
        {
            0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos =>
pad_octave.pos;
            vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain =>
pad_octave.gain;
            0 => pad_ninth.gain;
            lower_scale[9] => pad_root.read;
            upper_scale[0] => pad_third.read;
            lower_scale[4] => pad_fifth.read;
            lower_scale[7] => pad_seventh.read;
            upper_scale[9] => pad_octave.read;

            2 :: second => now;
        }
    }
    return;
}


// iii-VI7(#9)-ii-V7 PROGRESSION
// ["C", "Db", "D", "Eb", "E", "F", "Gb", "G", "Ab", "A", "Bb", "B"]
fun void two_five_extension( float length, float vol, int tritone, int deceptive_2, int tritone_3)
{
    <<< "Two-five extension reached." >>>;
    vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain =>
pad_octave.gain => pad_ninth.gain;
    for( 1 => int j; j <= (length/4); j++)
    {
        0 => pad_root.pos => pad_third.pos => pad_fifth.pos;
        vol => pad_root.gain => pad_third.gain => pad_fifth.gain;
        0 => pad_seventh.gain => pad_octave.gain => pad_ninth.gain => pad_extension.gain;
        lower_scale[4] => pad_root.read;
        lower_scale[7] => pad_third.read;
        lower_scale[11] => pad_fifth.read;

        2 :: second => now;
    }
    for( length/4 => float j; j < (length/2); j+(length/4) => j)
    {
        0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos =>
pad_extension.pos;
        vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain =>
pad_extension.gain;
        0 => pad_octave.gain => pad_ninth.gain;
        lower_scale[9] => pad_root.read;
        lower_scale[1] => pad_third.read;
        lower_scale[4] => pad_fifth.read;
        lower_scale[7] => pad_seventh.read;
        upper_scale[0] => pad_extension.read;

        2 :: second => now;
    }
    two_five(length, vol, tritone, deceptive_2, tritone_3);
    return;
}


// iii-vi-ii-V7 PROGRESSION
// ALTERED vi OPTION
// ["C", "Db", "D", "Eb", "E", "F", "Gb", "G", "Ab", "A", "Bb", "B"]
fun void three_six( float length, float vol, int substitution, int substitution_3, int tritone, int
deceptive_2, int tritone_3)
{
    <<< "Three-six reached." >>>;
    vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain =>
pad_octave.gain => pad_ninth.gain;
    // iii-7 6/5
    for( 1 => int s; s <= (length/4); s++ )
    {
        0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos;
        vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain;
        0 => pad_octave.gain => pad_ninth.gain => pad_extension.gain;
        upper_scale[4] => pad_root.read;
        lower_scale[7] => pad_third.read;
        lower_scale[11] => pad_fifth.read;
        upper_scale[2] => pad_seventh.read;

        2 :: second => now;
    }
    // vi-M7(#13)
    if( substitution_3 == 1 )
    {
        <<< "vi-M7(#13) reached." >>>;
        for( length/4 => float k; k <= (length/2); k+(length/4) => k)
        {
            0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_extension.pos =>
pad_seventh.pos;
            vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_extension.gain =>
pad_seventh.gain;
            0 => pad_ninth.gain => pad_octave.gain;
            lower_scale[9] => pad_root.read;
            lower_scale[0] => pad_third.read;
            upper_scale[4] => pad_fifth.read;
            lower_scale[6] => pad_extension.read;
            upper_scale[8] => pad_seventh.read;

            2 :: second => now;
        }
    }
    // vi-7 (4/3)
    if( substitution == 0)
    {
        <<< "vi-7 reached." >>>;
        for( length/4 => float k; k <= (length/2); k+(length/4) => k)
        {
            0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos;
            vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain;
            0 => pad_ninth.gain => pad_octave.gain => pad_extension.gain;
            lower_scale[9] => pad_root.read;
            upper_scale[0] => pad_third.read;
            upper_scale[4] => pad_fifth.read;
            lower_scale[7] => pad_seventh.read;

            2 :: second => now;
        }
    }
    two_five(length, vol, tritone, deceptive_2, tritone_3);
    return;
}
```

```
// COLTRANE MATRIX OFF I-CHORD
// ["C", "Db", "D", "Eb", "E", "F", "Gb", "G", "Ab", "A", "Bb", "B"]
fun void trane( float length, float vol, int matrix_4 )
{
   <<< "All aboard the sky trane!." >>>;
   // C(M7)
   <<< "CM7" >>>;
   if( matrix_4 == 1)
   {
      for( length/6 => float c; c < (length/3); c+(length/6) => c )
      {
         0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos;
         vol => pad_root.gain => pad_third.gain => pad_fifth.gain => pad_seventh.gain;
         0 => pad_ninth.gain => pad_octave.gain => pad_extension.gain;
         lower_scale[0] => pad_root.read;
         lower_scale[4] => pad_third.read;
         lower_scale[7] => pad_fifth.read;
         lower_scale[11] => pad_seventh.read;

         2 :: second => now;
      }
      // Eb7
      <<< "Eb7" >>>;
      for( length/6 => float c; c < (length/3); c+(length/6) => c )
      {
         lower_scale[3] => pad_root.read;
         lower_scale[7] => pad_third.read;
         lower_scale[10] => pad_fifth.read;
         upper_scale[1] => pad_seventh.read;

         2 :: second => now;
      }
      // Ab(M7)
      <<< "Ab(M7)" >>>;
      for( length/6 => float c; c < (length/3); c+(length/6) => c )
      {
         lower_scale[8] => pad_root.read;
         upper_scale[0] => pad_third.read;
         upper_scale[3] => pad_fifth.read;
         upper_scale[7] => pad_seventh.read;

         2 :: second => now;
      }
```

```
        // B7
        <<< "B7" >>>;
        for( length/6 => float c; c < (length/3); c+(length/6) => c )
        {
           lower_scale[11] => pad_root.read;
           upper_scale[3] => pad_third.read;
           lower_scale[6] => pad_fifth.read;
           lower_scale[9] => pad_seventh.read;

           2 :: second => now;
        }
        // E(M7)
        <<< "E(M7)" >>>;
        for( length/6 => float c; c < (length/3); c+(length/6) => c )
        {
           lower_scale[4] => pad_root.read;
           lower_scale[8] => pad_third.read;
           lower_scale[11] => pad_fifth.read;
           upper_scale[3] => pad_seventh.read;

           2 :: second => now;
        }
        // G7
        <<< "G7" >>>;
        for( length/6 => float c; c < (length/3); c+(length/6) => c )
        {
           lower_scale[7] => pad_root.read;
           lower_scale[11] => pad_third.read;
           upper_scale[2] => pad_fifth.read;
           lower_scale[5] => pad_seventh.read;

           2 :: second => now;
        }
    }
    return;
}


// IV-vi-II7-V7
// ["C", "Db", "D", "Eb", "E", "F", "Gb", "G", "Ab", "A", "Bb", "B"]

fun void four_six( float length, float vol, int substitution, int tritone, int deceptive )
{
    <<< "Four-six reached." >>>;
```

```
// F(M7)
for( length/4 => float f; f <= (length/2); f+(length/4) => f)
{
    0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos;
    0 => pad_octave.gain => pad_ninth.gain => pad_extension.gain;
    lower_scale[5] => pad_root.read;
    lower_scale[9] => pad_third.read;
    lower_scale[0] => pad_fifth.read;
    upper_scale[4] => pad_seventh.read;

    2 :: second => now;
}
// a-
for( length/4 => float s; s < (length/2); s+(length/4) => s)
{
    0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_octave.pos;
    0 => pad_seventh.gain => pad_ninth.gain;
    lower_scale[9] => pad_root.read;
    upper_scale[0] => pad_third.read;
    lower_scale[4] => pad_fifth.read;
    upper_scale[9] => pad_octave.read;

    2 :: second => now;
}
// D7
for( length/4 => float t; t < (length/2); t+(length/4) => t)
{
    0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos;
    0 => pad_octave.gain => pad_ninth.gain => pad_extension.gain;
    upper_scale[2] => pad_root.read;
    lower_scale[6] => pad_third.read;
    lower_scale[9] => pad_fifth.read;
    upper_scale[0] => pad_seventh.read;

    2 :: second => now;
}
```

```
    // G7
    for( length/4 => float f; f < (length/2); f+(length/4) => f)
    {
       0 => pad_root.pos => pad_third.pos => pad_fifth.pos => pad_seventh.pos;
       0 => pad_octave.gain => pad_ninth.gain => pad_extension.gain;
       lower_scale[7] => pad_root.read;
       lower_scale[11] => pad_third.read;
       upper_scale[2] => pad_fifth.read;
       upper_scale[5] => pad_seventh.read;

       2 :: second => now;
    }
    return;
 }


 // USE SNDBUF2 DAC FOR PANNING SAMPLES. LET EACH DATA STREAM DRIVE
 STEREO PANNING OF ONE SONORITY IN THE CHORD
// POSSIBLY ROOT-3RD-5TH
```