1. [8점] ChucK 프로그램의 쉬레드(shred)에 대한 다음 설명 중에서 맞지 않은 것을 하나 고르시오.
   (1) spork ~ f(); 와 같은 형태로 함수 f를 호출하면 새로운 자식 쉬레드가 생긴다.
   (2) Machine.add(path); 와 같은 형태로 path로 지정한 파일의 코드를 실행하는 새로운 쉬레드를 만들 수 있다.
   (3) 쉬레드는 시간을 중심으로 서로 독립적으로 작동하므로 기본적으로 다른 쉬레드를 알 필요가 없다.
   (4) 동시에 존재할 수 있는 쉬레드의 최대 개수는 256개로 설정되어 있으며, 더 늘리고 싶으면 설정 값을 변경해야 한다.
   (5) 한 쉬레드가 실행을 끝내고 사라지면, 그 쉬레드가 만들어놓은 자식 쉬레드도 모두 같이 사라진다.
   (6) 부모 쉬레드는 자식 쉬레드가 없어져도 그대로 남아있다.

2. [8점] SinOsc 진동기를 다음과 같이 선언하였다.

```
SinOsc s;
```

이 SinOsc 객체 s의 주파수(frequency)와 소리크기(gain)의 기본값은 각각 220.0과 1.0이다. 이 값을 440.0과 0.5로 변경하는 코드를 작성하시오.

```
440.0 => s.freq;
0.5 => s.gain;
```

3. [8점] StkInstrument 악기와 음정, 소리크기를 인수로 받아서 그 악기의 주파수와 소리크기를 설정하는 함수를 다음과 같이 작성하였다. 아래 중에서 맞는 것을 **모두** 고르시오.

```
fun void setNote(StkInstrument instr, float freq, float volume) {
    freq => instr.freq;
    volume => instr.gain;
}

fun void setNote(StkInstrument instr, int note, float volume) {
    Std.mtof(note) => instr.freq;
    volume => instr.gain;
}
```

   (1) Mandolin guitar; setNote(guitar, 330.0, 1.0); 와 같이 호출하면 오류가 발생하지 않고 의도한 대로 잘 작동한다.
   (2) Rhodey piano; setNote(piano, 60, 1); 와 같이 호출하면 오류가 발생하지 않고 의도한 대로 잘 작동한다.
   (3) 위 두 함수는 함수 이름이 같지만 파라미터의 타입이 다르므로 공존이 가능하다.
   (4) 위 두 함수는 함수 이름이 같기 때문에 둘 중 하나는 함수 이름을 다르게 수정해야 한다.

4. [8점] public class와 private class를 틀리게 설명한 것을 하나 고르시오.
   (1) 앞에 public 또는 private을 언급하지 않으면 private class 로 취급한다.
   (2) private class는 동일 파일 내에서만 사용 가능하다.
   (3) public class는 다른 파일에서만 사용 가능하고, 동일 파일에서는 사용가능하지 않다.
   (4) public class는 한 번 선언하면 버철 머신에 계속 남아 있으므로, 개발 중에는 지양하는 것이 편리하다.

5. [8점] static 필드 변수를 사용하는 가장 중요한 목적은 무엇인지 하나 고르시오.
   (1) 변수의 값을 고정하기 위해서
   (2) 변수의 값을 선언한 파일의 외부에서도 공유할 수 있도록 하기 위해서
   (3) 변수의 값을 수시로 바꿀 수 있게 하기 위해서
   (4) 변수의 이름을 독점하기 위해서

6. [8점] 다음 프로그램을 실행한 결과에 대해서 맞게 설명한 것을 아래에서 고르시오.

```
public class MyClarinet extends Clarinet {

    fun void noteOn(int note, float volume) {
        Std.mtof(note) => this.freq;
        volume => this.noteOn;
    }
}

MyClarinet clarinet => dac;

clarinet.noteOn(72, 1);
second => now;
clarinet.noteOff(1.0);
second => now;
clarinet.noteOn(1.0);
second => now;
```

(1) 첫 번째 noteOn 메소드 호출에서 둘째 인수의 타입이 맞이 않아 컴파일 오류가 발생한다.
(2) noteOff 가 선언되어 있지 않으므로 noteOff 메소드를 호출하면서 오류가 발생한다.
(3) 두 번째 noteOn 메소드 호출에서 인수가 하나만 주어졌으므로 오류가 발생한다.
(4) noteOn 메소드는 Clarinet 클래스에 있는 같은 이름의 메소드를 중복하여 선언한 것으로, 인수의 개수가 하나인 두 번째 호출은 MyClarinet 클래스에서 선언한 noteOn 메소드 대신 Clarinet 클래스에서 선언한 noteOn 메소드를 사용하게 되므로 문제없이 작동한다.

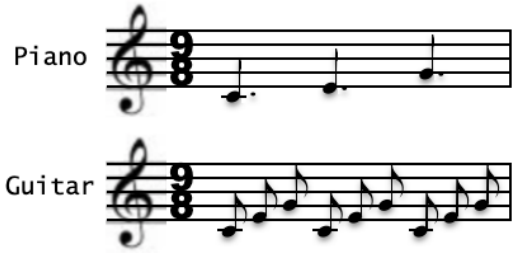
7. [8점] 다음 중에서 Event를 발생시키는 것을 모두 고르시오.
(1) signal()
(2) broadcast()
(3) 마우스 클릭
(4) 키보드 누르기
(5) 춤추기

8. [8점] 다음 문장에서 사실과 다른 부분을 긋고 수정하시오.

MIDI 메시지는 3개의 정수 바이트 메시지로 구성되며, 1로 시작하는 첫 바이트는 메시지의 종류를 나타내고, 0으로 시작하는 나머지 두 바이트는 음정과 소리크기를 각각 나타낸다. 메시지는 하나의 객체로 취급하며, 각 바이트는 data1, data2, data3으로 접근할 수 있다. 1 바이트는 8비트이므로 십진수 정수로 표현하면, 각 바이트 별로 가질 수 있는 값의 범위는 모두 0~256 사이의 값을 가진다.

MIDI 메시지는 3개의 정수 바이트 메시지로 구성되며, 1로 시작하는 첫 바이트는 메시지의 종류를 나타내고, 0으로 시작하는 나머지 두 바이트는 음정과 소리크기를 각각 나타낸다. 메시지는 하나의 객체로 취급하며, 각 바이트는 data1, data2, data3으로 접근할 수 있다. 1 바이트는 8비트이므로 십진수 정수로 표현하면, 각 바이트 별로 가질 수 있는 값의 범위는 ~~**모두 0~255 사이의 값**~~을 가진다. **(수정: 첫 바이트는 128~255 사이의 값, 나머지 두 바이트는 0~127 값을 갖는다.)**

다음 9번과 10번은 피아노와 기타가 듀엣으로 연주하는 이 악보를 연주하는 프로그램을 작성하는 문제이다.



9. [18점] 이벤트 구동 방식으로 위 악보를 무한 반복 연주하는 다음 프로그램의 빈 부분을 채워서 완성하자.

```
Event e1, e2, e3, e4;
[60,64,67] @=> int majorC3[];
Rhodey piano;
Mandolin guitar;
spork ~ play(piano, e1, majorC3, 0.9::second, 1.0);
spork ~ play(guitar, e2, majorC3, 0.3::second, 0.5);
spork ~ play(guitar, e3, majorC3, 0.3::second, 0.5);
spork ~ play(guitar, e4, majorC3, 0.3::second, 0.5);


while (true) { // 이 부분을 채워서 프로그램을 완성하자.









    0.9::second => now;
}

fun void play(StkInstrument stk, Event e, int notes[], dur length, float volume) {
    stk => dac;
    while (true) {
        e => now;
        for (0 => int i; i < notes.size(); i++) {
            Std.mtof(notes[i]) => stk.freq;
            volume => stk.noteOn;
            length => now;
            1 => stk.noteOff;
        }
    }
}
```

완성코드

```
Event e1, e2, e3, e4;
[60,64,67] @=> int majorC3[];
Rhodey piano;
Mandolin guitar;
spork ~ play(piano, e1, majorC3, 0.9::second, 1.0);
spork ~ play(guitar, e2, majorC3, 0.3::second, 0.5);
spork ~ play(guitar, e3, majorC3, 0.3::second, 0.5);
spork ~ play(guitar, e4, majorC3, 0.3::second, 0.5);

2 => int turn;
while (true) { // 이 곳을 채워서 프로그램을 완성하자.
    me.yield();
    if (turn == 2) {
        e1.signal();
        e2.signal();
        3 => turn;
    }
    else if (turn == 3) {
        e3.signal();
        4 => turn;
    }
    else {
        e4.signal();
        2 => turn;
    }
    0.9::second => now;
}

fun void play(StkInstrument stk, Event e, int notes[], dur length, float volume) {
    stk => dac;
    while (true) {
        e => now;
        for (0 => int i; i < notes.size(); i++) {
            Std.mtof(notes[i]) => stk.freq;
            volume => stk.noteOn;
            length => now;
            1 => stk.noteOff;
        }
    }
}
```

10. [18점] 다음 왼쪽 컬럼의 세 프로그램은 MIDI 메시지를 보내는 방식으로 위 악보를 무한 반복 연주하는 프로그램으로,메시지를 받는 파일 둘과 메시지를 보내는 파일 하나로 구성된 완성된 프로그램이다. 오른쪽 컬럼의 세 프로그램은 OSC 메시지를 보내는 방식으로 똑같은 연주를 하려고 작성 중이다. directorosc.ck 파일의 빈 칸을 채워서 이 프로그램을 완성하자.

MIDI 메시지 전송 방식

```
// piano.ck
MidiIn min;
MidiMsg msg;
0 => int port;
if (!min.open(port)) {
    <<< "Error: MIDI port did not open on port: ", port >>>;
    me.exit();
}

Rhodey piano => dac;

int note;
float volume;
while (true) {
    min => now;
    while (min.recv(msg)) { // noteOn = 144, noteOff = 128
        if (msg.data1 == 144) {
            Std.mtof(msg.data2) => piano.freq;
            msg.data3 / 127.0 => piano.gain;
            1 => piano.noteOn;
        }
        else
            1 => piano.noteOff;
    }
}
```

```
// guitar.ck
MidiIn min;
MidiMsg msg;
1 => int port;
if (!min.open(port)) {
    <<< "Error: MIDI port did not open on port: ", port >>>;
    me.exit();
}

Mandolin guitar => dac;

int note;
float volume;
while (true) {
    min => now;
    while (min.recv(msg)) { // noteOn = 144, noteOff = 128
        if (msg.data1 == 144) {
            Std.mtof(msg.data2) => guitar.freq;
            msg.data3 / 127.0 => guitar.gain;
            1 => guitar.noteOn;
        }
        else
            1 => guitar.noteOff;
    }
}
```

OSC 메시지 전송 방식

```
// piano.ck
OscIn oin;
8401 => oin.port;
oin.addAddress("/three/piano");
OscMsg msg;

Rhodey piano => dac;

int note;
float volume;
while (true) {
    oin => now;
    while (oin.recv(msg)) {
        msg.getInt(0) => note;
        msg.getFloat(1) => volume;
        if (volume == 0.0)
            1 => piano.noteOff;
        else {
            Std.mtof(note) => piano.freq;
            volume => piano.gain;
            1 => piano.noteOn;
        }
    }
}
```

```
// guitar.ck
OscIn oin;
8402 => oin.port;
oin.addAddress("/three/guitar");
OscMsg msg;

Mandolin guitar => dac;

int note;
float volume;
while (true) {
    oin => now;
    while (oin.recv(msg)) {
        msg.getInt(0) => note;
        msg.getFloat(1) => volume;
        if (volume == 0.0)
            1 => guitar.noteOff;
        else {
            Std.mtof(note) => guitar.freq;
            volume => guitar.gain;
            1 => guitar.noteOn;
        }
    }
}
```

```
// directormidi.ck
MidiOut piano, guitar;
MidiMsg msg_p, msg_g;
if (! piano.open(0)) me.exit();
if (! guitar.open(1)) me.exit();

[60,64,67] @=> int majorC3[];
spork ~ playOnce();
spork ~ playThrice();
while (true) second => now;

fun void playOnce() {
    while (true)
        play(piano, msg_p, majorC3, 0.9::second, 1.0);
}

fun void playThrice() {
    while (true) {
        play(guitar, msg_g, majorC3, 0.3::second, 0.5);
    }
}

fun void play(MidiOut mout, MidiMsg msg,
              int notes[], dur length, float volume) {
    for (0 => int i; i < notes.size(); i++) {
        144 => msg.data1; // noteOn
        notes[i] => msg.data2;
        Std.ftoi(volume * 127.0) => msg.data3;
        mout.send(msg);
        length => now;
    }
}
```

```
// directorosc.ck
OscOut piano, guitar;
OscMsg msg_p, msg_g;
piano.dest("localhost", 8401);
guitar.dest("localhost", 8402);

[60,64,67] @=> int majorC3[];
spork ~ playOnce();
spork ~ playThrice();
while (true) second => now;

fun void playOnce() {
    while (true)

}

fun void playThrice() {
    while (true) {

    }
}

fun void play(OscOut oout, string address,
              int notes[], dur length, float volume) {
    for (0 => int i; i < notes.size(); i++) {



        length => now;
    }
}
```

완성코드

```
// directorosc.ck
OscOut piano, guitar;
OscMsg msg_p, msg_g;
piano.dest("localhost", 8401);
guitar.dest("localhost", 8402);

[60,64,67] @=> int majorC3[];
spork ~ playOnce();
spork ~ playThrice();
while (true) second => now;

fun void playOnce() {
    while (true)
        play(piano, "/three/piano", majorC3, 0.9::second, 1.0);
}

fun void playThrice() {
    while (true) {
        play(guitar, "/three/guitar", majorC3, 0.3::second, 0.5);
    }
}

fun void play(OscOut oout, string address, int notes[], dur length, float volume) {
    for (0 => int i; i < notes.size(); i++) {
        oout.start(address);
        oout.add(notes[i]);
        oout.add(volume);
        oout.add("");
        oout.send();
        length => now;
    }
```

```
}
```