Task 2 (5 Marks)

Problem Description

Using the callstack-microtask-macrotask table

1. illustrate the execution of the following JavaScript program,
2. explain what will be printed on the console output
3. if the program leads to a non-termination, just show one cycle of execution.

```
import EventEmitter from 'events';
const ev1 = new EventEmitter();
const ev2 = new EventEmitter();
let count = 0;
let promise1 = new Promise( (resolve, reject) => {
    resolve(count);
})
let promise2 = new Promise( (resolve, reject) => {
    resolve(count);
})
function foo(x) {
    return new Promise((resolve, reject) => {
        if (x > 10) {
            resolve();
        } else if (x % 2 == 0) {
            ev1.emit('run', ++x);
        } else {
            ev2.emit('run', ++x);
        }
    })
}
ev1.on('run', (data) => { setImmediate(() => {
    console.log(`data ${data} received by ev1`);
    promise2.then((x) => foo(data)); });
});
ev2.on('run', (data) => { setImmediate(() => {
    console.log(`data ${data} received by ev2`);
    promise1.then((x) => foo(data)); });
});
ev2.emit('run', count);
```

The first few steps of the execution is given as follows to help you get started. Hint: `setImmediate()` enqueue task to *macro queue*.

program counter (line num)	call stack	micro queue	promises	macro queue	event reg
5	[main()]	[]	{promise@5}	[]	{}
8	[main()]	[]	{promise@5, promise@8}	[]	{}
22	[main()]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22 }
26	[main()]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }
30	[main()]	[]	{promise@5, promise@8}	[function@26(0)]	{ ev1.run:function@22, ev2.run:function@26 }
eof	[]	[]	{promise@5, promise@8}	[function@26(0)]	{ ev1.run:function@22, ev2.run:function@26 }

Solution

The callstack-microtask-macrotask table of the abovementioned program can be found in `task-2-trace.xlsx`. The table was generated programmatically (refer to `index.js` for more information).

The following will be printed on the console:

```
data 0 received by ev2
data 1 received by ev1
data 2 received by ev2
data 3 received by ev1
data 4 received by ev2
data 5 received by ev1
data 6 received by ev2
data 7 received by ev1
data 8 received by ev2
data 9 received by ev1
data 10 received by ev2
data 11 received by ev1
```

This is because the event emitters `ev1` and `ev2` take turns to handle and receive data, starting from `ev2`. The variables `data` and `x` in the function are somewhat interchangeable, and their value is incremented each time an event emitter receives the data. This continues until the termination condition `x > 10` is reached (when `x = 11`), in which case, neither event emitter is able to receive any data, all promises are resolved, and the entire program terminates.