1 C

This shows the Reflex Game example for the C target of Lingua Franca using the "default" style.

```
/**
 * This example illustrates the use of logical and physical actions,
 * asynchronous external inputs, the use of startup and shutdown reactions, and
 * the use of actions with values.
 *
 * The example is fashioned after an Esterel implementation given by Berry and
 * Gonthier in "The ESTEREL synchronous programming language: design, semantics,
 * implementation," Science of Computer Programming, 19(2) pp. 87-152, Nov.
 * 1992, DOI: 10.1016/0167-6423(92)90005-V.
 *
 * @author Edward A. Lee
 * @author Marten Lohstroh
 */
target C {
    keepalive: true
}

/**
 * Produce a counting sequence at random times with a minimum and maximum time
 * between outputs specified as parameters.
 *
 * @param min_time The minimum time between outputs.
 * @param max_time The maximum time between outputs.
 */
reactor RandomSource(min_time: time = 2 sec, max_time: time = 8 sec) {
    preamble {=
        // Generate a random additional delay over the minimum.
        // Assume millisecond precision is enough.
        interval_t additional_time(interval_t min_time, interval_t max_time) {
            int interval_in_msec = (max_time - min_time) / MSEC(1);
            return (rand() % interval_in_msec) * MSEC(1);
        }
    =}
    input another: int
    output out: int
    logical action prompt(min_time)
    state count: int = 0

    reaction(startup) -> prompt {=
        printf("**********************************************\n");
        printf("Watch for the prompt, then hit Return or Enter.\n");
        printf("Type Control-D (EOF) to quit.\n\n");

        // Random number functions are part of stdlib.h, which is included by reactor.h.
        // Set a seed for random number generation based on the current time.
        srand(time(0));

        // Schedule the first event.
        lf_schedule(prompt, additional_time(0, self->max_time - self->min_time));
    =}

    reaction(prompt) -> out {=
```

```
53          self->count++;
54          printf("%d. Hit Return or Enter!", self->count);
55          fflush(stdout);
56          lf_set(out, self->count);
57      =}
58
59      reaction(another) -> prompt {=
60          // Schedule the next event.
61          lf_schedule(prompt, additional_time(0, self->max_time - self->min_time));
62      =}
63  }
64
65  /**
66   * Upon receiving a prompt, record the time of the prompt, then listen for user
67   * input. When the user hits return, then schedule a physical action that
68   * records the time of this event and then report the response time.
69   */
70  reactor GetUserInput {
71      preamble {=
72          // Thread to read input characters until an EOF is received.
73          // Each time a newline is received, schedule a user_response action.
74          void* read_input(void* user_response) {
75              int c;
76              while(1) {
77                  while((c = getchar()) != '\n') {
78                      if (c == EOF) break;
79                  }
80                  lf_schedule_copy(user_response, 0, &c, 1);
81                  if (c == EOF) break;
82              }
83              return NULL;
84          }
85      =}
86
87      physical action user_response: char
88      state prompt_time: time = 0
89      state total_time_in_ms: int = 0
90      state count: int = 0
91
92      input prompt: int
93      output another: int
94
95      reaction(startup) -> user_response {=
96          // Start the thread that listens for Enter or Return.
97          lf_thread_t thread_id;
98          lf_thread_create(&thread_id, &read_input, user_response);
99      =}
100
101     reaction(prompt) {= self->prompt_time = lf_time_logical(); =}
102
103     reaction(user_response) -> another {=
104         if (user_response->value == EOF) {
105             lf_request_stop();
106             return;
107         }
108         // If the prompt_time is 0, then the user is cheating and
109         // hitting return before being prompted.
```

```
110             if (self->prompt_time == 0LL) {
111                 printf("YOU CHEATED!\n");
112                 lf_request_stop();
113             } else {
114                 int time_in_ms = (lf_time_logical() - self->prompt_time) / 1000000LL;
115                 printf("Response time in milliseconds: %d\n", time_in_ms);
116                 self->count++;
117                 self->total_time_in_ms += time_in_ms;
118                 // Reset the prompt_time to indicate that there is no new prompt.
119                 self->prompt_time = 0LL;
120                 // Trigger another prompt.
121                 lf_set(another, 42);
122             }
123         =}
124
125         reaction(shutdown) {=
126             if (self->count > 0) {
127                 printf("\n**** Average response time: %d.\n", self->total_time_in_ms/self->count);
128             } else {
129                 printf("\n**** No attempts.\n");
130             }
131         =}
132     }
133
134     main reactor ReflexGame {
135         p = new RandomSource()
136         g = new GetUserInput()
137         p.out -> g.prompt
138         g.another -> p.another
139     }
```

2 C++

This shows the Reflex Game example for the C++ target of Lingua Franca using the "algol_nu" style.

```
/**
 * This example illustrates the use of logical and physical actions,
 * asynchronous external inputs, the use of startup and shutdown reactions, and
 * the use of actions with values.
 *
 * @author Felix Wittwer
 * @author Edward A. Lee
 * @author Marten Lohstroh
 */
target Cpp {
    keepalive: true,
    cmake-include: "ReflexGame.cmake"
}

/**
 * Produce a counting sequence at random times with a minimum and maximum time
 * between outputs specified as parameters.
 *
 * @param min_time The minimum time between outputs.
 * @param max_time The maximum time between outputs.
 */
reactor RandomSource(min_time: time(2 sec), max_time: time(8 sec)) {
    private preamble {=
        // Generate a random additional delay over the minimum.
        // Assume millisecond precision is enough.
        reactor::Duration additional_time(reactor::Duration min_time, reactor::Duration max_time) {
            int interval_in_msec = (max_time - min_time) / std::chrono::milliseconds(1);
            return (std::rand() % interval_in_msec) * std::chrono::milliseconds(1);
        }
    =}
    input another: void
    output out: void
    logical action prompt(min_time)
    state count: int(0)

    reaction(startup) -> prompt {=
        std::cout << "**********************************************" << std::endl;
        std::cout << "Watch for the prompt, then hit Return or Enter." << std::endl;
        std::cout << "Type Control-D (EOF) to quit." << std::endl << std::endl;

        // TODO: Manual inclusion of header necessary?
        // Set a seed for random number generation based on the current time.
        std::srand(std::time(nullptr));

        // Schedule the first event.
        prompt.schedule(additional_time(0ms, max_time - min_time));
    =}

    reaction(prompt) -> out {=
        count++;
        std::cout << count << ". Hit Return or Enter!" << std::endl << std::flush;
        out.set();
```

```
    =}

    reaction(another) -> prompt {=
        // Schedule the next event.
        prompt.schedule(additional_time(0ms, max_time - min_time));
    =}
}

/**
 * Upon receiving a prompt, record the time of the prompt, then listen for user
 * input. When the user hits return, then schedule a physical action that
 * records the time of this event and then report the response time.
 */
reactor GetUserInput {
    public preamble {=
        #include <thread>
    =}

    physical action user_response: char
    state prompt_time: {= reactor::TimePoint =}({= reactor::TimePoint::min() =})
    state total_time: time(0)
    state count: int(0)
    state thread: {= std::thread =}

    input prompt: void
    output another: void

    reaction(startup) -> user_response {=
        // Start the thread that listens for Enter or Return.
        thread = std::thread([&] () {
            int c;
            while(1) {
                while((c = getchar()) != '\n') {
                    if (c == EOF) break;
                }
                user_response.schedule(c, 0ms);
                if (c == EOF) break;
            }
        });
    =}

    reaction(prompt) {= prompt_time = get_physical_time(); =}

    reaction(user_response) -> another {=
        auto c = user_response.get();
         if (*c == EOF) {
             environment()->sync_shutdown();
             return;
         }
         // If the prompt_time is 0, then the user is cheating and
         // hitting return before being prompted.
         if (prompt_time == reactor::TimePoint::min()) {
             std::cout << "YOU CHEATED!" << std::endl;
             environment()->sync_shutdown();
         } else {
             reactor::TimePoint logical = get_logical_time();
             std::chrono::duration elapsed = (logical - prompt_time);
```

```
110                 auto time_in_ms = std::chrono::duration_cast<std::chrono::milliseconds>(elapsed);
111                 std::cout << "Response time in milliseconds: " << time_in_ms << std::endl;
112                 count++;
113                 total_time += time_in_ms;
114                 // Reset the prompt_time to indicate that there is no new prompt.
115                 prompt_time = reactor::TimePoint::min();
116                 // Trigger another prompt.
117                 another.set();
118             }
119         =}
120
121         reaction(shutdown) {=
122             thread.join();
123             if (count > 0) {
124                 std::cout << std::endl << "**** Average response time: " << std::chrono::duration_cast<std::chrono::m
125             } else {
126                 std::cout << std::endl << "**** No attempts." << std::endl;
127             }
128         =}
129     }
130
131     main reactor ReflexGame {
132         p = new RandomSource()
133         g = new GetUserInput()
134         p.out -> g.prompt
135         g.another -> p.another
136     }
```

3 Python

This shows the Piano example for the Python target of Lingua Franca using the "arduino" style.

```
target Python {
    files: [gui.py, keys.png, soundfont.sf2],
    threading: true,
    keepalive: true
};


/*
 * Receives key presses from the pygame piano process
 */
reactor GetUserInput {
    preamble {=
        import threading
        def listen_for_input(self, user_response):
            while 1:
                try:
                    c = self.user_input.recv()
                except EOFError:
                    request_stop()
                    return
                # Each time a key press is received, schedule a user_response event
                user_response.schedule(0, c)
    =}
    physical action user_response;
    input user_input_pipe_init;
    output user_input;
    state user_input({=None=}) # multiprocessing.connection.PipeConnection

    reaction(user_input_pipe_init) -> user_response {=
        # starts a thread to receive key presses from the pygame process
        self.user_input = user_input_pipe_init.value
        t = self.threading.Thread(target=self.listen_for_input, args=(user_response, ))
        t.start()
    =}

    reaction(user_response) -> user_input {=
        user_input.set(user_response.value)
    =}
}


/*
 * Sends graphics updates to the pygame piano process
 */
reactor UpdateGraphics {
    input note;
    input update_graphics_pipe_init;
    state update_graphics({=None=}); # multiprocessing.connection.PipeConnection
    state pressed_keys({=set()=})

    reaction(update_graphics_pipe_init) {=
        self.update_graphics = update_graphics_pipe_init.value
```

```
    =}

    reaction(note) {=
        key_down, note_t = note.value
        if key_down and note_t not in self.pressed_keys:
            self.pressed_keys.add(note_t)
            self.update_graphics.send(self.pressed_keys)
        if not key_down and note_t in self.pressed_keys:
            self.pressed_keys.remove(note_t)
            self.update_graphics.send(self.pressed_keys)
    =}
}


/*
 * Plays sound using fluidsynth upon receiving signal from TranslateKeyToNote
 */
reactor PlaySound {
    state lowest(4);   # the octave of the lowest "C" on the piano.
    state channel(8);
    state Note;
    state fluidsynth;
    input note;
    input play_sound_init;

    reaction(play_sound_init) {=
        self.fluidsynth, self.Note = play_sound_init.value
    =}

    reaction(note) {=
        # upon receiving a note, play or stop the note depending on if its a key down or key up.
        key_down, note_t = note.value
        if key_down:
            self.fluidsynth.play_Note(self.Note(note_t[0], self.lowest + note_t[1]), self.channel, 100)
        else:
            self.fluidsynth.stop_Note(self.Note(note_t[0], self.lowest + note_t[1]), self.channel)
    =}
}

/*
 * Translates key presses to piano keys and triggers the initialization of StartGui
 */
reactor TranslateKeyToNote {
    preamble {=
    piano_keys = {
        "z": ("C", 0),
        "s": ("C#", 0),
        "x": ("D", 0),
        "d": ("D#", 0),
        "c": ("E", 0),
        "v": ("F", 0),
        "g": ("F#", 0),
        "b": ("G", 0),
        "h": ("G#", 0),
        "n": ("A", 0),
        "j": ("A#", 0),
        "m": ("B", 0),
```

```
        "w": ("C", 1),
        "3": ("C#", 1),
        "e": ("D", 1),
        "4": ("D#", 1),
        "r": ("E", 1),
        "t": ("F", 1),
        "6": ("F#", 1),
        "y": ("G", 1),
        "7": ("G#", 1),
        "u": ("A", 1),
        "8": ("A#", 1),
        "i": ("B", 1)
        }
    =}

    input user_input;
    input translate_init;
    output note;
    output gui_init;

    reaction(translate_init) -> gui_init {=
        gui_init.set(self.piano_keys)
    =}

    reaction(user_input) -> note {=
        key_down, c = user_input.value
        if c in self.piano_keys:
            note.set((key_down, self.piano_keys[c]))
    =}
}

reactor StartFluidSynth {
    preamble {=
        import sys
        import os

        try:
            from mingus.containers.note import Note
        except:
            print("Import Error: Failed to import 'mingus'. Try 'pip3 install mingus'")
            request_stop()
            sys.exit(1)

        try:
            from mingus.midi import fluidsynth
        except:
            if sys.platform == "darwin":
                print("Import Error: fluidsynth is missing. Try 'brew install fluidsynth'")
            elif sys.platform == "linux" or sys.platform == "linux2":
                print("Import Error: fluidsynth is missing. Try 'sudo apt-get install -y fluidsynth'")
            else:
                print("Import Error: fluidsynth is missing. ")
            request_stop()
            sys.exit(1)
    =}
    state soundfont({=self.os.path.join(self.os.path.dirname(__file__), "soundfont.sf2")=})
    output translate_init;
```

```
    output play_sound_init;

    reaction(startup) -> play_sound_init, translate_init {=
        if not self.os.path.exists(self.soundfont):
            print("Error: Soundfont file does not exist.")
            print("Try downloading a soundfont file from here (this is the soundfont used for testing the demo):
            print("http://www.schristiancollins.com/generaluser.php")
            print("Alternatively, pick and download a soundfont from here:")
            print("https://github.com/FluidSynth/fluidsynth/wiki/SoundFont")
            print("Rename the soundfont to \"soundfont.sf2\" and put it under the same directory as Piano.lf.")
            request_stop()
            return

        # initialize fluidsynth
        driver = None
        if self.sys.platform == "linux" or self.sys.platform == "linux2":
            driver = "alsa"
        if not self.fluidsynth.init(self.soundfont, driver):
            print("Error: Failed to initialize fluidsynth")
            request_stop()
            return

        play_sound_init.set((self.fluidsynth, self.Note))
        translate_init.set(0)
    =}
}

/*
 * Starts the GUI and triggers initialization of UpdateGraphics and GetUserInput reactors.
 */
reactor StartGui {
    preamble {=
        import gui
    =}
    input gui_init;
    output user_input_pipe;
    output update_graphics_pipe;

    reaction(gui_init) -> user_input_pipe, update_graphics_pipe {=
        piano_keys = gui_init.value
        user_input_pout, update_graphics_pin = self.gui.start_gui(piano_keys)
        user_input_pipe.set(user_input_pout)
        update_graphics_pipe.set(update_graphics_pin)
    =}
}

main reactor {
    gui = new StartGui()
    fs = new StartFluidSynth()
    translate = new TranslateKeyToNote()
    update_graphics = new UpdateGraphics()
    get_user_input = new GetUserInput()
    play_sound = new PlaySound()

    fs.translate_init -> translate.translate_init;
    fs.play_sound_init -> play_sound.play_sound_init;
    gui.user_input_pipe -> get_user_input.user_input_pipe_init
```

```
224       gui.update_graphics_pipe -> update_graphics.update_graphics_pipe_init
225       get_user_input.user_input -> translate.user_input
226       translate.note -> update_graphics.note
227       translate.note -> play_sound.note
228       translate.gui_init -> gui.gui_init
229   }
```

4 Rust

This shows the Snake example for the Rust target of Lingua Franca using the "colorful" style.

```
//! A snake terminal game. Does not support windows.
//!
//! Highlights of this example:
//! - external packages are linked in using Cargo (see `cargo-dependencies` target property)
//! - a support library is linked into the generated crate (see `rust-include` target property)
//! - physical actions are used to handle keyboard input asynchronously (see `KeyboardEvents.lf`)
//! - logical actions are used to implement a timed loop with variable period
//! - the game may be configured with the CLI
//!
//! This example was presented at the ESWEEK Tutorial
//! "Deterministic Reactive Programming for Cyber-Physical
//! Systems Using Lingua Franca" on October 8th, 2021.
//!
//! Author: Clément Fournier
//!
//! Note: Git history of this file may be found in https://github.com/lf-lang/reactor-rust
//! under the path examples/src/Snake.lf

target Rust {
    // LF-Rust programs integrate well with Cargo
    cargo-dependencies: {
        termcolor: "1",
        termion: "1", // (this doesn't support windows)
        rand: "0.8",
    },
    // This will be linked into the root of the crate as a Rust module: `pub mod snakes;`
    rust-include: "snakes.rs",
    // This is a conditional compilation flag that enables the CLI.
    // Without it, command-line arguments are ignored and produce a warning.
    cargo-features: ["cli"],
};

// Import a shared reactor
import KeyboardEvents from "KeyboardEvents.lf";

// main reactor parameters can be set on the CLI, eg:
//  ./snake --main-grid-side 48
main reactor Snake(grid_side: usize(32),
                   tempo_step: time(40 msec),
                   food_limit: u32(2)) {
    preamble {=
        use crate::snakes::*;
        use crate::snakes;
        use termion::event::Key;
        use rand::prelude::*;
    =}

    /// this thing helps capturing key presses
    keyboard = new KeyboardEvents();

    // model classes for the game.
    state snake: CircularSnake ({= CircularSnake::new(grid_side) =});
```

```
53      state grid: SnakeGrid ({= SnakeGrid::new(grid_side, &snake) =}); // note that this one borrows snake temporar
54
55      /// Triggers a screen refresh, not a timer because we can
56      /// shrink the period over time to speed up the game.
57      logical action screen_refresh;
58      /// The game speed level
59      state tempo: u32(1);
60      state tempo_step(tempo_step);
61
62      /// Changes with arrow key presses, might be invalid.
63      /// Only committed to snake_direction on grid update.
64      state pending_direction: Direction ({= Direction::RIGHT =});
65      /// Whither the snake has slithered last
66      state snake_direction: Direction ({= Direction::RIGHT =});
67
68      /// manually triggered
69      logical action manually_add_more_food;
70      /// periodic
71      timer add_more_food(0, 5 sec);
72      // state vars for food
73      state food_on_grid: u32(0);
74      state max_food_on_grid(food_limit);
75
76      // @label startup
77      reaction(startup) -> screen_refresh {=
78          // KeyboardEvents makes stdout raw on startup so this is safe
79          snakes::output::paint_on_raw_console(&self.grid);
80
81          // schedule the first one, then it reschedules itself.
82          ctx.schedule(screen_refresh, after!(1 sec));
83      =}
84
85      // @label schedule_next_tick
86      reaction(screen_refresh) -> screen_refresh {=
87          // select a delay depending on the tempo
88          let delay = delay!(400 ms) - (self.tempo_step * self.tempo).min(delay!(300 ms));
89
90          ctx.schedule(screen_refresh, After(delay));
91      =}
92
93      // @label refresh_screen
94      reaction(screen_refresh) -> manually_add_more_food {=
95          // check that the user's command is valid
96          if self.pending_direction != self.snake_direction.opposite() {
97              self.snake_direction = self.pending_direction;
98          }
99
100         match self.snake.slither_forward(self.snake_direction, &mut self.grid) {
101             UpdateResult::GameOver => { ctx.request_stop(Asap); return; },
102             UpdateResult::FoodEaten => {
103                 self.food_on_grid -= 1;
104                 if self.food_on_grid == 0 {
105                     ctx.schedule(manually_add_more_food, Asap);
106                 }
107                 self.tempo += 1;
108             },
109             UpdateResult::NothingInParticular => {/* do nothing in particular. */}
```

```
110             }
111
112             snakes::output::paint_on_raw_console(&self.grid);
113         =}
114
115         // @label handle_key_press
116         reaction(keyboard.arrow_key_pressed) {=
117             // this might be overwritten several times, only committed on screen refreshes
118             self.pending_direction = match ctx.get(keyboard__arrow_key_pressed).unwrap() {
119                 Key::Left => Direction::LEFT,
120                 Key::Right => Direction::RIGHT,
121                 Key::Up => Direction::UP,
122                 Key::Down => Direction::DOWN,
123                 _ => unreachable!(),
124             };
125         =}
126
127         // @label add_food
128         reaction(manually_add_more_food, add_more_food) {=
129             if self.food_on_grid >= self.max_food_on_grid {
130                 return; // there's enough food there
131             }
132
133             if let Some(cell) = self.grid.find_random_free_cell() {
134                 self.grid[cell] = CellState::Food; // next screen update will catch this.
135                 self.food_on_grid += 1;
136             }
137         =}
138
139         // @label shutdown
140         reaction(shutdown) {=
141             println!("New high score: {}", self.snake.len());
142         =}
143     }
```

5 TypesScript

This shows the Chat Application example for the TypeScript target of Lingua Franca using the "vs" style.

```
/**
 * This program is a simple chat application for two users.
 *
 * @author Byeonggil Jun (junbg@hanyang.ac.kr)
 * @author Hokeun Kim (hokeunkim@berkeley.edu)
 */

target TypeScript {
    coordination-options: {advance-message-interval: 100 msec}
}

reactor InputHandler {
    output out:string;
    physical action response;

    preamble {=
        import * as readline from "readline";
    =}

    reaction(startup, response) -> out, response {=
        const rl = readline.createInterface({
            input: process.stdin,
            output: process.stdout
        });

        if (response !== undefined) {
            out = response as string;
        }

        rl.question("Enter message to send: ", (buf) => {
            actions.response.schedule(0, buf as string);
            rl.close();
        });
    =}
 }

reactor Printer {
    input inp:string;

    reaction(inp) {=
        console.log("Received: " + inp);
    =}
}

reactor ChatHandler {
    input receive:string;
    output send:string;
    u = new InputHandler();
    p = new Printer();

    reaction(u.out) -> send {=
        send = u.out;
```

```
53      =}
54      reaction(receive) -> p.inp {=
55          p.inp = receive;
56      =}
57  }
58
59  federated reactor SimpleChat {
60      a = new ChatHandler();
61      b = new ChatHandler();
62      b.send -> a.receive;
63      a.send -> b.receive;
64  }
65
```