Todas os exercícios e projeto final

Repositório Github: https://github.com/GabrielFeliciano/Virtual-Dice-Bluepill

Projeto Final

```c
#include <stdbool.h>
#include "gpio/gpio.h"

// || -----------------------------------------------------------------------------------------
// || ----------------------------------------- Types -----------------------------------------
// ||
// || Todos os tipos necessarios para o programa como *structs* ou *enums*.
// ||
// \/ -----------------------------------------------------------------------------------------

enum ResultStatus {
    Err,
    Ok
};

struct ResultInt {
    enum ResultStatus status;
    int payload;
};

struct ResultArrayOfInt {
    enum ResultStatus status;
    int *payload;
    int size;
};

struct UpcommingSignal {
    int current;
    int last;
};

// || --------------------------------------------------------------------------------------------------
// || ----------------------------------------- Function Types -----------------------------------------
// ||
// || Declara��es dos tipos dos parametros e da saida. Necessario quando se implementa as funcoes depois
// || do *main*.
// ||
// \/ ------------------------------------------------------------------------------------------------

struct ResultArrayOfInt Proteus7Seg_Strategy(int num);

typedef struct ResultArrayOfInt (*DisplayInputMapper)(int);
enum ResultStatus display_simbol(DisplayInputMapper strategy, int display_pins[], int num);

// || ----------------------------------------------------------------------------------------------------
// || ----------------------------------------- Global variables -----------------------------------------
// ||
// || Todas as variaveis globais.
// ||
// \/ ----------------------------------------------------------------------------------------------------

int display_pin[] = {3,4,5,6,7,8,9};

// || ------------------------------------------------------------------------------------------
// || ----------------------------------------- Macros ------------------------------------------
// ||
// || Macros s�o uma maneira de dizer para o compilador onde certa parte precisa ser trocada por
// || um codigo.
// ||
// \/ ------------------------------------------------------------------------------------------


#define button_pin 15
#define write_number_on_proteus_7_seg_display(x) (display_simbol(Proteus7Seg_Strategy, display_pin, x))

// || ------------------------------------------------------------------------------------------- ||
// || ----------------------------------------- Program ----------------------------------------- ||
// || ----------------------------------------- Program ----------------------------------------- ||
// || ----------------------------------------- Program ----------------------------------------- ||
// \/ ------------------------------------------------------------------------------------------- \/

// Configura��o dos pinos GPIO
void configure_pins() {
    // GPIO init
    gpio_init();

    // Button
    gpio_pin_mode(GPIOB, button_pin, gpio_mode_input_pupd); // Input pupd
    gpio_pin_write(GPIOB, button_pin, 0); // Configura Pull down

    // Display pins
    for (int i = 0; i < sizeof(display_pin)/sizeof(int); i++) {
        gpio_pin_mode(GPIOB, display_pin[i], gpio_mode_output_PP_2MHz);
    }
}

// Main
void main() {
    // Configure all pins
    configure_pins();

    // Configure
    srand((unsigned) time(0));

    // Program state
    int number = 1;
    struct UpcommingSignal buttonState = {
        gpio_pin_read(GPIOB, button_pin),
        gpio_pin_read(GPIOB, button_pin)
    };

    while (true) {
        buttonState.last = buttonState.current;
        buttonState.current = gpio_pin_read(GPIOB, button_pin);

        if (buttonState.last != buttonState.current && buttonState.current == 1) {
            number = rand() % 6 + 1;
        }

        enum ResultStatus result = display_simbol(Proteus7Seg_Strategy, display_pin,
number);
    }
}

// || --------------------------------------------------------------------------------
-------------
// || ----------------------------------------- Functions ----------------------------
-------------
// ||
// || Todas as fun□□es necessarias para o codigo.
// ||
// \/ --------------------------------------------------------------------------------
-------------

// Write output to pins
void write_pins(int pins[], int output[], int size) {
    for (int i = 0; i < size; i++) {
        int pin = pins[i];
        int value = output[i];
        gpio_pin_write(GPIOB, pin, value);
    }
}

// Given a strategy and a pin map, prints a number in displayer
// Int -> Pointer List Int -> (Int -> ResultArrayOfInt) -> ResultStatus
enum ResultStatus display_simbol(DisplayInputMapper strategy, int display_pins[], int
num) {
    struct ResultArrayOfInt resultOutput = strategy(num);
    int *output = resultOutput.payload;

    if (sizeof(display_pins) != sizeof(output) || resultOutput.status == Err) return
Err;

    write_pins(display_pins, output, resultOutput.size);

    return Ok;
}

// All Display Strategys

int outputPossibilities_Proteus7Seg_Strategy[6][7] = {
    {1, 0, 0, 1, 1, 1, 1}, // 1
    {0, 0, 1, 0, 0, 1, 0}, // 2
    {0, 0, 0, 0, 1, 1, 0}, // 3
    {1, 0, 0, 1, 1, 0, 0}, // 4
    {0, 1, 0, 0, 1, 0, 0}, // 5
    {0, 1, 0, 0, 0, 0, 0}  // 6
};
struct ResultArrayOfInt Proteus7Seg_Strategy(int number) {
    // Validation
    enum ResultStatus status = Err;
    if (1 <= number && number <= 6) status = Ok;

    // Mapping
    int output[7];
    switch (number) {
        case 1 : *output = outputPossibilities_Proteus7Seg_Strategy[0]; break;
        case 2 : *output = outputPossibilities_Proteus7Seg_Strategy[1]; break;
        case 3 : *output = outputPossibilities_Proteus7Seg_Strategy[2]; break;
        case 4 : *output = outputPossibilities_Proteus7Seg_Strategy[3]; break;
        case 5 : *output = outputPossibilities_Proteus7Seg_Strategy[4]; break;
        case 6 : *output = outputPossibilities_Proteus7Seg_Strategy[5]; break;
    }

    // Return
    struct ResultArrayOfInt result = { status, *output, 7 };
    return result;
}
```

```
// End of All display strategys
```

Etapas

Etapa A

```
#include <stdbool.h>
#include "gpio/gpio.h"

// Macros

#define button_pin 15
#define led_pin 6

// Program

void configure_pins() {
    // GPIO init
    gpio_init();

    // Button
    gpio_pin_mode(GPIOB, button_pin, gpio_mode_input_pupd);
    gpio_pin_write(GPIOB, button_pin, gpio_pupd_pu);

    // Led
    gpio_pin_mode(GPIOB, led_pin, gpio_mode_output_PP_10MHz);
    gpio_pin_write(GPIOB, led_pin, 0);
}

void main() {
    // Configure all pins
    configure_pins();

    bool buttonStatus;
    while (true) {
        buttonStatus = !gpio_pin_read(GPIOB, button_pin);
        gpio_pin_write(GPIOB, led_pin, !buttonStatus);
        gpio_pin_write(GPIOC, 13, buttonStatus);
    }
}
```

Etapa B

```
#include <stdbool.h>
#include "gpio/gpio.h"

// Macros
```

```
#define button_pin 15
#define led_pin 6

// Program

void configure_pins() {
    // GPIO init
    gpio_init();

    // Button
    gpio_pin_mode(GPIOB, button_pin, gpio_mode_input_pupd);
    gpio_pin_write(GPIOB, button_pin, gpio_pupd_pd);

    // Led
    gpio_pin_mode(GPIOB, led_pin, gpio_mode_output_PP_10MHz);
    gpio_pin_write(GPIOB, led_pin, 0);
}

void main() {
    // Configure all pins
    configure_pins();

    bool buttonStatus;
    while (true) {
        buttonStatus = gpio_pin_read(GPIOB, button_pin);
        gpio_pin_write(GPIOB, led_pin, !buttonStatus);
        gpio_pin_write(GPIOC, 13, buttonStatus);
    }
}
```

Etapa C

```
#include <stdbool.h>
#include "gpio/gpio.h"

// Macros

#define button_pin 15
#define led_pin 6

// Program

void configure_pins() {
    // GPIO init
    gpio_init();

    // Sensor
    gpio_pin_mode(GPIOB, button_pin, gpio_mode_input_floating);

    // Led
    gpio_pin_mode(GPIOB, led_pin, gpio_mode_output_PP_10MHz);
    gpio_pin_write(GPIOB, led_pin, 0);
}

void main() {
    // Configure all pins
    configure_pins();

    bool sensorStatus;
    while (true) {
        sensorStatus = gpio_pin_read(GPIOB, button_pin);
        gpio_pin_write(GPIOB, led_pin, !sensorStatus);
        gpio_pin_write(GPIOC, 13, sensorStatus);
    }
}
```

Etapa D

```
#include <stdbool.h>
#include "gpio/gpio.h"

// Types

enum ResultStatus {
    Err,
    Ok
};

struct ResultInt {
    enum ResultStatus status;
    int payload;
};

struct ResultArrayOfInt {
    enum ResultStatus status;
    int *payload;
    int size;
};

struct UpcommingSignal {
    int current;
    int last;
};

// Global variables

int display_pin[] = {3,4,5,6,7,8,9};

// Function signatures

typedef struct ResultArrayOfInt (*DisplayInputMapper)(int);
enum ResultStatus display_simbol(DisplayInputMapper strategy, int display_pins[], int
num);

// Macros

#define display_turn_off_all(x) (display_simbol(TurnOffAll_Strategy, display_pin, x))
#define display_turn_on_all(x) (display_simbol(TurnOnAll_Strategy, display_pin, x))

struct ResultArrayOfInt TurnOffAll_Strategy(int number);
struct ResultArrayOfInt TurnOnAll_Strategy(int number);

// Program

void configure_pins() {
    // GPIO init
    gpio_init();

    // Display pins
    for (int i = 0; i < sizeof(display_pin)/sizeof(int); i++) {
        gpio_pin_mode(GPIOB, display_pin[i], gpio_mode_output_PP_2MHz);
    }
}

void bad_delay (int n) {
    for (int i = 0; i < n; i++);
}

void main() {
    // Configure all pins
    configure_pins();

    while (true) {
        enum ResultStatus result_on = display_turn_on_all(42);
        bad_delay(500000);
        enum ResultStatus result_off = display_turn_off_all(42);
        bad_delay(500000);
    }
}


// -----------------
// --- Functions ---
// -----------------

// Write output to pins
void write_pins(int pins[], int output[], int size) {
    for (int i = 0; i < size; i++) {
        int pin = pins[i];
        int value = output[i];
        gpio_pin_write(GPIOB, pin, value);
    }
}
```

```
// Given a strategy and a pin map, prints a number in displayer
// Int -> Pointer List Int -> (Int -> ResultArrayOfInt) -> ResultStatus
enum ResultStatus display_simbol(DisplayInputMapper strategy, int display_pins[], int
num) {
    struct ResultArrayOfInt resultOutput = strategy(num);
    int *output = resultOutput.payload;

    if (sizeof(display_pins) != sizeof(output) || resultOutput.status == Err) return
Err;

    write_pins(display_pins, output, resultOutput.size);

    return Ok;
}

// All Display Strategys

// Turn on All leds
int outputPossibilities_TurnOnAll_Strategy[1][7] = {
    {0, 0, 0, 0, 0, 0, 0}, // All
};
struct ResultArrayOfInt TurnOnAll_Strategy(int number) {
    // Validation
    enum ResultStatus status = Ok;

    // Mapping
    int output[7];
    *output = outputPossibilities_TurnOnAll_Strategy[0];

    // Return
    struct ResultArrayOfInt result = { status, *output, 7 };
    return result;
}

// Turn off All leds
int outputPossibilities_TurnOffAll_Strategy[1][7] = {
    {1, 1, 1, 1, 1, 1, 1}, // All
};
struct ResultArrayOfInt TurnOffAll_Strategy(int number) {
    // Validation
    enum ResultStatus status = Ok;

    // Mapping
    int output[7];
    *output = outputPossibilities_TurnOffAll_Strategy[0];

    // Return
    struct ResultArrayOfInt result = { status, *output, 7 };
    return result;
}

// End of All display strategys
```

Etapa E

```c
#include <stdbool.h>
#include "gpio/gpio.h"

// Macros

// Program

void configure_pins() {

}

void bad_delay (int n) {
    for (int i = 0; i < n; i++);
}

void main() {
    // Configure all pins
    configure_pins();
    srand(time(0));

    while (true) {
        int random_number = rand();
        bad_delay(1000000);
    }
}
```

Etapa F

```c
#include <stdbool.h>
#include "gpio/gpio.h"

// Macros

#define button_pin 15
#define open_pin 14

// Program

void configure_pins() {
    // open pin
    gpio_pin_mode(GPIOB, open_pin, gpio_mode_input_analog);

    // button pin
    gpio_pin_mode(GPIOB, button_pin, gpio_mode_input_pupd);
    gpio_pin_write(GPIOB, button_pin, gpio_pupd_pd);
}

void main() {
    // Configure all pins
    configure_pins();

    // Making everything plus 1 makes sure that we dont have any 0
    int weird_calc = (time(0) || 1) * (gpio_pin_read(GPIOB, open_pin) || 1);
    srand(weird_calc);

    int random_number;
    bool button_read;
    while (true) {
        button_read = gpio_pin_read(GPIOB, button_pin);
        if (button_read) {
            random_number = rand();
        }
    }
}
```

Etapa G

```
#include <stdbool.h>
#include "gpio/gpio.h"

// Types

enum ResultStatus {
    Err,
    Ok
};

struct ResultInt {
    enum ResultStatus status;
    int payload;
};

struct ResultArrayOfInt {
    enum ResultStatus status;
    int *payload;
    int size;
};

struct UpcommingSignal {
    int current;
    int last;
};

// Global variables

int display_pin[] = {3,4,5,6,7,8,9};

// Function signatures
```

```
typedef struct ResultArrayOfInt (*DisplayInputMapper)(int);
enum ResultStatus display_simbol(DisplayInputMapper strategy, int display_pins[], int num);

// Macros

#define write_display(x) (display_simbol(Proteus7Seg_Strategy, display_pin, x))

struct ResultArrayOfInt Proteus7Seg_Strategy(int num);

// Program

void bad_delay (int n) {
    for (int i = 0; i < n; i++);
}

void configure_pins() {
    // GPIO init
    gpio_init();

    // Display pins
    for (int i = 0; i < sizeof(display_pin)/sizeof(int); i++) {
        gpio_pin_mode(GPIOB, display_pin[i], gpio_mode_output_PP_2MHz);
    }
}

void main() {
    // Configure all pins
    configure_pins();

    // Program state
    int number = 0;

    while (true) {
        // Number state update
        number += 1;
        if (number < 1 || number > 6) number = 1;

        enum ResultStatus result = write_display(number);
        bad_delay(500000);
    }
}


// -----------------
// --- Functions ---
// -----------------

// Write output to pins
void write_pins(int pins[], int output[], int size) {
    for (int i = 0; i < size; i++) {
        int pin = pins[i];
        int value = output[i];
        gpio_pin_write(GPIOB, pin, value);
    }
}

// Given a strategy and a pin map, prints a number in displayer
// Int -> Pointer List Int -> (Int -> ResultArrayOfInt) -> ResultStatus
enum ResultStatus display_simbol(DisplayInputMapper strategy, int display_pins[], int
num) {
    struct ResultArrayOfInt resultOutput = strategy(num);
    int *output = resultOutput.payload;

    if (sizeof(display_pins) != sizeof(output) || resultOutput.status == Err) return
Err;

    write_pins(display_pins, output, resultOutput.size);

    return Ok;
}

// All Display Strategys

int outputPossibilities_Proteus7Seg_Strategy[6][7] = {
    {1, 0, 0, 1, 1, 1, 1}, // 1
    {0, 0, 1, 0, 0, 1, 0}, // 2
    {0, 0, 0, 0, 1, 1, 0}, // 3
    {1, 0, 0, 1, 1, 0, 0}, // 4
    {0, 1, 0, 0, 1, 0, 0}, // 5
    {0, 1, 0, 0, 0, 0, 0}  // 6
};
struct ResultArrayOfInt Proteus7Seg_Strategy(int number) {
    // Validation
    enum ResultStatus status = Err;
    if (1 <= number && number <= 6) status = Ok;

    // Mapping
    int output[7];
    switch (number) {
        case 1 : *output = outputPossibilities_Proteus7Seg_Strategy[0]; break;
        case 2 : *output = outputPossibilities_Proteus7Seg_Strategy[1]; break;
        case 3 : *output = outputPossibilities_Proteus7Seg_Strategy[2]; break;
        case 4 : *output = outputPossibilities_Proteus7Seg_Strategy[3]; break;
        case 5 : *output = outputPossibilities_Proteus7Seg_Strategy[4]; break;
        case 6 : *output = outputPossibilities_Proteus7Seg_Strategy[5]; break;
    }

    // Return
    struct ResultArrayOfInt result = { status, *output, 7 };
    return result;
}

// End of All display strategys
```