```c
/*
 * CE2812 - 021
 * Winter 2016
 * Lab 3 - LCD API
 * Name: Yahui Liang
 * Created: 12/17/2016
 */

/* All header files */
#include "lcd_api.h"
#include "delay_api.h"
#include <inttypes.h>
#include <stdio.h>
#include <string.h>
#include "reg_struct.h"

/* private method prototypes */
static void init_GPIOB();
static void init_GPIOC();
static void init_GPIOx_clock(char);
static void init_PB0_to_PB2_mode();
static void init_PC8_to_PC11_mode();
static void lcd_set_up();
static void lcd_cmd(uint8_t);
static void lcd_data(uint8_t);
static void lcd_exec(uint8_t);
static void lcd_set_upper_nibble(uint8_t);
static void lcd_set_lower_nibble(uint8_t);
static void lcd_latch();

/* Private variables for file scope */
static volatile GPIOx* gpiob = (GPIOx*) GPIOB_BASE;
static volatile GPIOx* gpioc = (GPIOx*) GPIOC_BASE;

/**
 * Initializes the lcd.
 */
void lcd_init() {
	init_GPIOB();
	init_GPIOC();
	lcd_set_up();
}

/**
 * Clears all contents on the lcd.
 */
void lcd_clear() {
	uint8_t cmd = 1;
	lcd_cmd(cmd);
	delay_ms(2);
}

/**
 * Returns the cursor to the home position.
 */
void lcd_home() {
	uint8_t cmd = 0b10;
```

```
	lcd_cmd(cmd);
	delay_ms(2);
}

/**
 * Sets the cursor position.
 * Args:
 * row: 0 is the first row, and 1 is the second row.
 * col: 0 is the first col, and 15 is the last col.
 */
void lcd_set_position(uint8_t row, uint8_t col) {
	uint8_t ddrm_adr = col;
	ddrm_adr += row * 0x40;
	uint8_t cmd = (1 << 7) + ddrm_adr;
	lcd_cmd(cmd);
}

/**
 * Prints a string on lcd.
 * Args:
 * string: the string which needs to be displayed.
 */
int lcd_print_string(char string[]) {
	int str_len = strlen(string);
	for (int i = 0; i < str_len; i++) {
		lcd_data(string[i]);
	}
	return str_len;
}

/**
 * Prints a number on lcd.
 * Args:
 * num: the number which needs to be displayed.
 */
void lcd_print_num(int num) {
	char num_str[32];
	sprintf(num_str, "%d", num);
	lcd_print_string(num_str);
}

/**
 * Initializes GPIOB port which includes enabling clock and
 * setting mode.
 */
static void init_GPIOB() {
	init_GPIOx_clock('B');
	init_PB0_to_PB2_mode();
}

/**
 * Initializes GPIOC port which includes enabling clock and
 * setting mode.
 */
static void init_GPIOC() {
	init_GPIOx_clock('C');
	init_PC8_to_PC11_mode();
```

```
}

/**
 * Initializes a specified GPIO clock.
 * Args:
 * port: a letter which indicates which GPIO port
 * needs to be enabled.
 */
static void init_GPIOx_clock(char port) {
	/* enable the clock */
	volatile uint32_t* rcc_ahb1enr;
	rcc_ahb1enr = (uint32_t*) RCC_AHB1ENR;
	if (port == 'B') {
		*rcc_ahb1enr |= GPIOBEN;
	} else if (port == 'C') {
		*rcc_ahb1enr |= GPIOCEN;
	}
}

/**
 * Sets mode for PB0 - PB2.
 */
static void init_PB0_to_PB2_mode() {
	for (int bit = 0; bit < 3; bit++) {
		gpiob -> MODER &= ~(0b11 << (bit * 2)); // make sure all bits are
set to zero.
		gpiob -> MODER |= OUTPUT << (bit * 2); // set mode from PB0 to
PB2 to output mode.
	}
}

/**
 * Sets mode for PC8 - PC11.
 */
static void init_PC8_to_PC11_mode() {
	for (int bit = 8; bit < 12; bit++) {
		gpioc -> MODER &= ~(0b11 << (bit * 2)); // make sure all bits are
set to zero.
		gpioc -> MODER |= OUTPUT << (bit * 2); // set mode from PC7 to
PC11 to output mode.
	}
}

/**
 * Sets up lcd by writing initialization commands to it.
 */
static void lcd_set_up() {
	delay_ms(100); // delay more than 40ms.
	lcd_cmd(0x28);
	lcd_cmd(0x01);
	delay_us(1500);
	lcd_cmd(0x02);
	delay_us(1500);
	lcd_cmd(0x06);
	lcd_cmd(0x0C);
}
```

```
/**
 * Writes a command to lcd.
 * Args:
 * cmd: the command which needs to be processed.
 */
static void lcd_cmd(uint8_t cmd) {
      uint32_t rw_rs_clr = RS_CLR | RW_CLR;
      /* make sure RS and RW are cleared to be 0 */
      gpiob -> BSRR = rw_rs_clr;
      lcd_exec(cmd);
}

/**
 * Writes a data to lcd.
 * Args:
 * data: the data which needs to be processed.
 */
static void lcd_data(uint8_t data) {
      uint32_t rw_rs_clr = RS_SET | RW_CLR;
      /* set RS and clear RW in order to write data to it */
      gpiob -> BSRR = rw_rs_clr;
      lcd_exec(data);
}

/**
 * Places the instruction at appropriate positions
 * in order to make sure all bits will be written to lcd.
 * Args:
 * instruction: the instruction which needs to be processed.
 */
static void lcd_exec(uint8_t instruction) {
      lcd_set_upper_nibble(instruction);
      lcd_latch();
      lcd_set_lower_nibble(instruction);
      lcd_latch();
      delay_us(40);
}

/**
 * Processes the upper half instruction.
 * Args:
 * instruction: the whole instruction which needs to be
 * processed.
 */
static void lcd_set_upper_nibble(uint8_t instruction) {
      uint16_t upper_half_cmd = (instruction & ~0xF) >> 4;
      gpioc -> BSRR = 0b1111 << (8 + 16); // clear instruction bits
previously hold by the register.
      gpioc -> BSRR = upper_half_cmd << 8;
}

/**
 * Processes the lower half instruction.
 * Args:
 * instruction: the whole instruction which needs to be
 * processed.
 */
```

```
static void lcd_set_lower_nibble(uint8_t instruction) {
	uint16_t lower_half_cmd = instruction & ~(0xF << 4);
	gpioc -> BSRR = 0b1111 << (8 + 16); // clear instruction bits
previously hold by the register.
	gpioc -> BSRR = lower_half_cmd << 8;
}

/**
 * Lets lcd fetch data from 4 logic pins.
 */
static void lcd_latch() {
	gpiob -> BSRR = E_SET;
	delay_us(1);
	gpiob -> BSRR = E_CLR;
	delay_us(1);
}
```