```
/*
 * customDelay.c
 *
 * Created: 3/1/2021 12:34:40 PM
 *  Author: Alec Derwent
 *
 * A .c file which is designed to control various timers. Each function is detailed below.
 *
 *  void init_timer0(void):
 *      init_timer0 initializes timer 0 in normal mode with no set output or delay. This was done in order
 *      to simplify usage of timer 0 in other functions, not limiting timer 0 to one function or operation type.
 *
 *
 *  void init_timer1(void):
 *      init_timer1 initializes timer 1 as a 9-bit fast PWM mode timer with no output set and a duty cycle of 0.
 *      This timer was designed only for 9-bit fast PWM in mind, hence why it is initialized and not changed
 *      later on.
 *
 *  void init_timer3(void):
 *      init_timer3 initializes timer 3 as a CTC operation PWM timer with a prescaler of 1024. Similarly to
 *      timer 1, it was designed with only that mode of operation in mind.
 *
 *  void delay_ms(uint16_t delay):
 *      This function produces a 1ms delay then loops it for however many desired ms. First, a desired delay
 *      is written in as a parameter. Next, timer 0 is set up with a prescaler of 64 to create a 1ms delay,
 *      stopping its operation after finishing each 1ms duration. This duration is then looped N amount of
 *      timer, where N = delay.
 *
 *  void pwm_timer1(char mode):
 *      pwm_timer1 controls the operation of timer 1, designed to fulfill the requirements of procedure 2.
 *      It uses a parameter "mode" of type char to select between running and stopping. When the run mode
 *      is selected, the output pin is enabled, the prescaler and operation mode is double checked, then
 *      ramp_up_delay_n_steps() is called to ramp up the duty cycle (detailed later in this header).
 *      If Stop mode or the default mode is selected, the output pin is changed back to regular I/O.
 *
 * void pwm_ADC_mod(uint32_t input, char mode):
 *      pwm_ADC_mod() is a function designed to modify the output frequency or duty cycle for timer 3 or
```

```
 *      timer 1 respectively. It does this by adjusting OCRnA based on the input value
    and selected mode.
 *      "F" is to modify the frequency of timer 3, while "D" is to modify the duty
   cycle of timer 1.
 *      Finally, there is a "C" mode, which disables both outputs and sets OCR3A and
   OCR1A to zero.
 *
 * void ramp_up_delay_n_steps(uint8_t start, uint8_t end, uint16_t mS_time, uint8_t
   num_steps):
 *
 *
 * Hardware
 *        Timer 1 OCR1A         PB.5
 *        Timer 3 OCR3A         PE.3
 */

#include "customDelay.h"

void init_timer0(void)
{
    TCNT0 = 0;      // begin with no delay set
    TCCR0A = 0x00;  //timer in normal mode
    TCCR0B = 0x00;  // timer initially off
    OCR0A = 0;      // no output setting
}

void init_timer1(void)
{
    TCNT1 = 0;      // begin with no delay set
    OCR1A = 0;  // set default duty cycle as 10%
    TCCR1A = (1 << WGM11);  //timer in 9-bit fast pwm mode
    TCCR1B = (1 << WGM12) | (1 << CS11);    // timer initially off
    OCR1A = 0;  // no duty cycle
}

void init_timer3(void)
{
    TCNT3 = 0;  // begin with no delay
    TCCR3A = 0; // set output to normal pin operation
    TCCR3B = (1<<WGM32) | (1<<CS32) | (1<<CS30); // CTC operation, 1024 prescaler
    OCR3A = 0;  // no frequency set
}

void delay_ms(uint16_t delay)
{
    // 1 ms delay for each ms desired
    for (uint16_t i = 0; i < delay; i++)
    {
        OCR0A = 250;    //
        TCCR0B = (1 << CS01) | (1 << CS00); // prescaler = 64
        while ((TIFR0 & (1<<OCF0A)) == 0); // begin polling
        TCCR0B = 0; // stop output when finished
```

```
        TIFR0 = TIFR0 |(1<<OCF0A);  // stop operations
    }
}

void pwm_timer1(char mode)
{
    // mode selector for timer1
    switch (mode)
    {
        // run mode
        case 'R':
        {
            TCCR1A |= (1 << COM1A1); // enable output pin
            TCCR1B = (1 << WGM12) | (1 << CS11); // ensure prescaler and 9-bit
              operation
            ramp_up_delay_n_steps(10, 90, 8000, 8); // begin ramping up duty cycle
            break;
        }

        // stop mode
        case 'S':
        {
            TCCR1A &= ~(1 << COM1A1); // change output back to regular I/O
            break;
        }

        // default mode (stops on default)
        default:
        {
            TCCR1A &= ~(1 << COM1A1); // change output back to regular I/O
            break;
        }
    }
}

void pwm_ADC_mod(uint32_t input, char mode)
{
    // mode selector for toggle switch
    switch (mode)
    {
        // frequency change mode
        case 'F':
        {
            TCCR3A |= (1 << COM3A0); // enable output pin
            OCR3A = ((16000000)/(input*1024)) -1; // calculate OCR3A from frequency
            break;
        }

        // duty cycle change mode
        case 'D':
        {
            TCCR1A |= (1 << COM1A1); // enable output pin
```

```
            OCR1A = 512*input/100UL; // calculate OCR1A from duty cycle
            break;
        }

        // clear outputs
        case 'C':
        {
            TCCR1A &= ~(1 << COM1A1); // disable output pin
            OCR1A = 0;  // clear output of timer1
            TCCR3A &= ~(1 << COM3A0); // disable output pin
            OCR3A = 0; // clear output of timer3
        }
    }

}

void ramp_up_delay_n_steps(uint8_t start, uint8_t end, uint16_t mS_time, uint8_t
  num_steps)
{
    OCR1A = 512*start/100UL; // calculate and set start duty cycle
    uint32_t duty_cycle_change = (end-start)/num_steps; // find change in duty cycle
      for each step
    uint32_t duty_cycle_converted = (duty_cycle_change * 512)/100; // calculate
      converted duty cycle change each step
    uint32_t time_per_step = mS_time/num_steps; // calculate timer per step

    // for however many number of steps
    for (uint8_t i = 0; i < num_steps; i++)
    {
        delay_ms(time_per_step); // delay for time calculated for each step
        if(OCR1A <= 512 && OCR1A >= 0) // check if value is within limits
        {
            OCR1A += duty_cycle_converted; // increment duty cycle calculated for each
               step
        }
    }
}
```