Goals:
The goal of the lab is to modify the given code to produce a PWM waveform using Phase Correct Mode in C.

Equipment Usage
For this lab the following equipment will be used:

- LED
- Atmega328P
- AVR Studio 5
- AVRISP mkII

Background:
C vs. ASM
The transition from ASM to C is a transition from low-level programming to high-level programming. Most functions that were written in multiple steps can now be written in single line. For example the following ASM code adds 5+6:

LDI r16, 0x05
LDI r17, 0x06
ADD r17, r16

In C this can be achieved in one simple line of code:

X = 5+6

High-level code like C takes statements like the one above and converts it into machine language during compilation.
Phase Correct PWM

In previous labs, counting went only in one direction. When one of the timers reached its TOP value the counter would trigger the overflow flag (TOV) and reset the counter to 0. In Phase Correct mode, the counter will count up and hold it TOP value for 1 clock cycle and the count back down. The overflow flag will not trigger until the counter reaches the BOTTOM value. The value of TOP and BOTTOM are dependent on the type of WGM values allowing TOP to be 0xFF or OCRxA. If OCRxA is used then the values in the COM bits will dictate if the waveform is inverted or non-inverted. During Output Compare Mode non-inverted mode will clear the OCRxA during the up-counting and set the OCRxA bit during the down-counting.

Formulas:

\[ f_{\text{freq of Phase Correct PWM}} = \frac{f_{\text{clk I/O}}}{N \cdot 510} \]

\[ F_{\text{OC0}} = \frac{f_{\text{clk}}}{N(510)} \]

\[ \text{Duty Cycle} = \frac{\text{OCR}x}{255} \times 100 \]

\[ \text{Duty Cycle} = \frac{255 \cdot \text{OCR}0}{255} \times 100 \]

N = prescaler factor (1, 8, 64, 256, or 1024)
**Prelab:**
Design 1: Modify the given code to produce a PWM waveform on the Atmega328P that will change from 50% duty cycle to 100% duty cycle. Adjust the frequency so that the LED appears to dim not blink.

```c
#define  F_CPU 10000000UL    //XTAL = 1 MHz
#include <avr/io.h>
#include <util/delay.h>

int main ()
{
    unsigned char i;
    DDRB = 0x08;       //PB3 as output
    i = 127;
    OCR0 = 127;        //duty cycle = 50%
    TCCR0 = 0x73;      //Phase Correct PWM, inverted, N = 64

    while (i != 0)
    {
        OCR0 = i;
        _delay_ms(25);    //use AVR studio library delay
        i++;
    }
    while (1);        //wait here forever after dimming is done
    return 0;
}
```

**Lab Experiments:**

**Experiment 1:** Modify the above code to send a PWM waveform ranging from 10% to 100% duty cycle for the ATmega328P.

**Experiment 2:** Modify your code to use Timer1 instead of Timer0
Questions:

1.) Take the following assembly code from Lab 5 and convert it into C code. Make sure your C code compiles and works in Atmel Studio

```
SBI DDRB, 1
BEGIN:
LDI R20, 0x7A
STS OCR1AH, R20
LDI R20, 0xAF
STS OCR1AL, R20
LDI R20, 0x0B
STS TCCR1B, R20 ;CTC mode, prescaler = 64, WGM = 0100 CS = 101
LDI R20, 0x40
STS TCCR1A, R20 ;COM1A = 01 toggle on compare match

;load second pulse of duty cycle
LDI R20, 0x3C
STS OCR1AH, R20
LDI R20, 0x6D
STS OCR1AL, R20
LDI R20, 0x0B
STS TCCR1B, R20 ;CTC mode, prescaler = 64 CS = 101, WGM = 0100
LDI R20, 0x40
STS TCCR1A, R20 ;TOGGLE on match COM1A = 01
RJMP BEGIN
```

2.) Which type of code compiles faster, C or assembly? Explain why

3.) Give an example of when it would be better to use assembly code over C code, and give an example of when it would be better to use C instead of assembly.

Post-Lab Deliverables:
1) Submit your completed project report including your working code.