```
//*******************************************************************
***
//******** CDA3331 Intro to Micro class Nov 21, 2016
//******** Dr. Bassem Alhalabi
//******** Contributors: TA Pablo Pastran
//******** Skeleton Program for Lab 6
//******** Run this program as is to show that you have correct
hardware connections
//******** Explore the program and read the three analog signals
coming form the three sensors
//Final Draft//

#include <msp430.h>

int value=0, i=0 ;
int light = 0, lightroom = 0, dimled=50;
int temp = 0, temproom = 0;
int touch =0, touchroom =0;
int flag =0;
int ADCReading [3];

// Function Prototypes
void fadeLED(int valuePWM);
void ConfigureAdc(void);
void getanalogvalues();


int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                   // Stop WDT
    P1OUT = 0;
    P2OUT = 0;
    P1DIR = 0;
    P1REN = 0;
    P2REN = 0;
    P2DIR = 0;
    P1DIR |= ( BIT4 | BIT5 | BIT6);             // set bits 4, 5, 6 as
outputs
    P2DIR |=   BIT0;                            // set bit  0       as
outputs

    ConfigureAdc();

// reading the initial room values, lightroom, touchroom, temproom

       __delay_cycles(250);
```

```
        getanalogvalues();
        lightroom = light; touchroom = touch; temproom = temp;
        __delay_cycles(250);


for (;;)
{
// reading light, touch, and temp repeatedly at the beginning of the
main loop

        getanalogvalues();

//light controlling LED2 on launch pad (P1.6) via variable dimled
//use the light reading range limits 50-900, and convert them to 0-
100%

        dimled = light;
        dimled = ((dimled- 100)*100)/(800- 100);
        if(dimled <= 5)dimled = 0; else if (dimled >=95) dimled = 100;
        fadeLED(dimled);



// ****************************************
// beginning of area for all students changes



// section 1/3 ****
//light Controlling LED1 of on your breadboard
//Here we have a dead zone of no action to avoid flickering
//(switching on/off over a small fluctuating value on light).
//I chose the range of 1.1 to 1.5 of the baseline value;
//that is, no action if  (1.1 * lightroom < light < 1.5 * lightroom).

        if(light < lightroom * 1.80 && light > lightroom * 1.20)
          {}
        else
          {
            if(light >= lightroom * 1.80) {P1OUT |=  BIT4;
__delay_cycles(200);} // on if dark
            if(light <= lightroom * 1.20) {P1OUT &= ~BIT4;
__delay_cycles(200);} // off if light
          }

//**** requirement:
```

```
//* you need to explain how you understood the concept of dead zone
//* applied above to prevent the LED flickering



// section 2/3 ****
//Temperature Controlling LED2
        if (temp < temproom * 1.02 && temp > temproom)
        {/*deadzone -> dont do anything*/ }
        else
        {
            if(temp >= temproom * 1.02) { P1OUT |=  BIT5;
  __delay_cycles(200); }//On
            if (temp <= temproom) { P1OUT &= ~BIT5;
  __delay_cycles(200); }//Off
        }

// this code will cause the LED2 to fluctuate as the temp crosses
value of
// (temproom * 1.04) going up or down.
// This is due to the temp value being analog in nature and so noisy.

//**** requirement:
//* Change the code above to add the dead zone concept of no action
//* between 1.01-1.03 of the temproom baseline, just similar to the
//* way we did it for the light.



// section 3/3 ****
//Touch Controlling LED3
//The following code uses a dead zone of no action between 0.7-0.9 of
the value touch
//Had to use smaller deadzone because my sensor was insensitive

        if(touch > touchroom * 0.96 && touch < touchroom)
          {/*Deadzone*/}
        else
          {
            if(touch >= touchroom) {flag=1;}  //No touch
            if(touch <= touchroom * 0.96 && flag == 1) {P2OUT ^=
BIT0;   __delay_cycles(200); flag=0;}  //LED toggle
          }

// the two lines above make a simple turn-on while still touching,
// and a simple turn-off when not touching.
```

```
//**** requirement:
//* Change the code (the two lines above) so that with every touch,
LED3
//* toggles and stays, and when you un-touch, the LED does not toggle
back.
//* Hint: use the variable flag as to set when you touch and toggle,
so that
//* as long as you keep touching, and software goes in loops, it does
not
//* keep toggling. Only when you remove your touch, the flag is
cleared,
//* so that when touch again the software will toggle again only once.


// end of area for all students changes
// ****************************************

}
}

void ConfigureAdc(void)
{
   ADC10CTL1 = INCH_2 | CONSEQ_1;            // A2 + A1 + A0, single
sequence
   ADC10CTL0 = ADC10SHT_2 | MSC | ADC10ON;
   while (ADC10CTL1 & BUSY);
   ADC10DTC1 = 0x03;                         // 3 conversions
   ADC10AE0 |= (BIT0 | BIT1 | BIT2);         // ADC10 option select
}

void fadeLED(int valuePWM)
{
    P1SEL |= (BIT6);                         // P1.0 and P1.6 TA1/2
options
    CCR0 = 100 - 0;                          // PWM Period
    CCTL1 = OUTMOD_3;                        // CCR1 reset/set
    CCR1 = valuePWM;                         // CCR1 PWM duty cycle
    TACTL = TASSEL_2 + MC_1;                 // SMCLK, up mode
}

void getanalogvalues()
{
 i = 0; temp = 0; light = 0; touch =0;       // set all analog values
to zero
    for(i=1; i<=5 ; i++)                     // read all three analog
values 5 times each and average
```

```
  {
    ADC10CTL0 &= ~ENC;
    while (ADC10CTL1 & BUSY);                 //Wait while ADC is busy
    ADC10SA = (unsigned)&ADCReading[0];       //RAM Address of ADC
Data, must be reset every conversion
    ADC10CTL0 |= (ENC | ADC10SC);             //Start ADC Conversion
    while (ADC10CTL1 & BUSY);                 //Wait while ADC is busy
    light += ADCReading[1];                   // sum  all 5 reading
for the three variables
    touch += ADCReading[0];
    temp += ADCReading[2];
  }
 light = light/5; touch = touch/5; temp = temp/5;      // Average the 5
reading for the three variables
}


#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
    __bic_SR_register_on_exit(CPUOFF);
}
```