Laboratory Worksheet #11
PD Control and Type Casting Exercise

Exercise 1: Type Casting Calculations in the Motor Control Algorithm

In Laboratory 6 of the Course Material, download the worksheet_11.c file. Looking through the file, you can see six (plus a 16-bit version) different expressions that represent the same equation but have different typecasting of the variables in the equation. One of the expressions is a two byte form of temp_motorpw and is included as a further example of potential problems. For reference, the different expressions are shown below.

```
error = desired - actual;

/* 1st control algorithm equation */
        temp_motorpw = pw_neut + kp*(error) + kd*(error - prev_error);
/* 2nd control algorithm */
        temp_motorpw = pw_neut + (signed long)kp*(error) + kd*(error - prev_error);
/* 3rd control algorithm equation */
        temp_motorpw = (signed long)(pw_neut + kp*(error) + kd*(error - prev_error));
/* 4th control algorithm equation */
        temp_motorpw = pw_neut + kp*(signed int)(error) + kd*(signed int)(error - prev_error);
/* 5th control algorithm equation*/
        temp_motorpw = (signed long)pw_neut + (signed long)(kp*(error)) + (signed long)(kd*(error - prev_error));
/* 6th control algorithm equation*/
        temp_motorpw = (signed long)pw_neut + (signed long)kp*(signed long)(error) + (signed long)kd*(signed long)(error –
                prev_error);

prev_error = error;
```

temp_motorpw is the calculated pulsewidth to be implemented
pw_neut is the pulsewidth when the blimp is heading in the desired direction
kp is the proportional gain constant of the control algorithm (use a value of 30 for the calculations)
kd is the derivative gain constant of the control algorithm (use a value of 30 for the calculations)
desired is the reading from the input desired heading
actual is the latest heading measurement
prev_error is the previous calculation of the difference between desired and actual headings

The following four cases represent different physical conditions for the blimp. Based on the numbers provided, calculate the resulting value of temp_motorpw (use a calculator). Run the worksheet10.c code after editing the appropriate variables and compare your calculation with results from the different algorithms. Indicate which algorithms provide the expected answer. Remember, negative results are possible. Use a proportional gain of Kp = 30 and a derivative gain of Kd = 30. Record which typecasting algorithms are consistent with your calculation.

Case 1: center_motorpw=2765, prev_error=-1760, desired=1800, actual=3500 (blimp is turned too far to the right)
temp_motorpw (calculated) = -46435
Algorithms that provide a correct result: 2 & 6

Case 2: center_motorpw=2765, prev_error=1760, desired=3500, actual=1800 (blimp is turned too far to the left)
temp_motorpw (calculated) = 51965
Algorithms that provide a correct result: 1, 3, 4 & 6

Case 3: center_motorpw=2765, prev_error=-250, desired=50, actual=250 (blimp is turned too far to the right)
temp_motorpw (calculated) = -1735
Algorithms that provide a correct result: 2 & 6

Case 4: center_motorpw=2765, prev_error=20, desired=3500, actual=1800 (rudder fan is at full power, but blimp is turning slower than desired)
temp_motorpw (calculated) = 104165
Algorithms that provide a correct result:
1,2,3,4,5,6

Exercise 2: Code execution

Based on your observations, **implement one of the typecasting algorithms in your code** where the pulsewidth is determined for the steering servo. You will need to work with long variable typecasting. Refer to the gondola_info sheet for suggestions on refining your code. Download your code to the microcontroller on a gondola. Set a desired heading of 135° (SE) and a proportional gain constant of 12. Fill in the first two columns in the table below, correcting your error is necessary so that it is bounded -180° < error < 180°.

Run your code and manually position the gondola at the heading directions indicated in the table and fill in the table using output from your code. You will need to print both the 'raw' (before limit correction) calculated pulsewidth and the 'corrected' (after limit correction) pulsewidth. As indicated in the table, record both calculated temp_motorpw before you check for pulsewidth limits and the actual pulsewidth after limits are enforced. Again, remember that the 'raw' pulsewidth can be a negative number. Note, since you are holding the gondola stationary, the differential gain term is zero (previous_error-current_error = 0).

Heading	Heading Error	Manually calculated temp_motorpw	Program calculated temp_motorpw (before limit correction)	Final temp_motorpw (after limit correction)
0.0°	1350	18965	18965	3500
45.0°	900	13565	13565	3500
90.0°	450	8165	8165	3500
135.0°	0	2765	2765	2765
180.0°	-450	-2635	-2635	2000
225.0°	-900	-8035	-8035	2000
270.0°	-1350	-13435	-13435	2000
315.0°	-1800	-18835	-18835	2000

When complete, insert Worksheet 11 in your laboratory notebook. Worksheets are required when the notebooks are graded. Perform any necessary calculations on the left page of the notebook where the worksheet is placed. Keep individual copies of the worksheet for your own records.

EVB Pin	
1	2
3	4
5	6
7	8
9	10
11	12
13	14
15	16
17	18
19	20
21	22
23	24
25	26
27	28
29	30
31	32
33	34
35	36
37	38
39	40
41 ⟷	60

	Port Bit	Bit Addresses & Labels
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		
10.		
11.	P1.3	0x93 POT
12.		
13.		
14.	P0.6	0x86 CEX2
15.	P0.7	0x87 CEX3
16.	P0.4	0x84 CEX0
17.	P0.5	0x85 CEX1
18.	P0.2	0x82 SDA
19.	P0.3	0x83 SCL
20.		
21.		
22.		
23.		
24.		
25.		
26.		
27.		
28.		
29.		
30.		
31.		
32.	P3.7	0xB7 SS
33.		
34.		
35.		
36.		
37.		
38.		
39.		
40.		

Software Initializations

A) Port I/0

```
P1MDIN &= 0xF7;
P1MDOUT &= 0xF7
P1 |= ~0xF7;

P3MDOUT &= 0x7F;
P3 |= ~0x7F;

P0MDOUT &= 0xF3;
P0MDOUT |= 0xF0;
P0 |= ~0xF3;
```

B) Timers

C) Interrupts

```
EA = 1;
EIE1 |= 0x08;
```

D) A/D

```
REF0CN = 0x03;
ADC1CN = 0x80;
ADC1CF |= 0x01;
```

E) PCA

```
PCA0MD =0x81;
PCA0CPM2 = 0xC2;
PCA0CN = 0x40;
```

F) XBAR

```
XBR0 = 0x25;
```

G) I2C

```
SMB0CR = 0x93;
ENSMB = 1;
```

```
compile derivatives
   #include <studio.h>
   #include <<stdlib.h>
   #include <c8051_SDCC.h>
   #include <i2c.h>

declare global variables
   const unsigned char RANGER_ADDRESS, COMPASS_ADDRESS, POT_ADDRESS, STAGE0, STAGE1,
STAGE2.
   unsigned char RANGER_DATA[2], COMPASS_DATA[2], RANGER_READ_INDICATOR,
COMPASS_READ_INDICATOR, AD_VALUE, OUTPUT_MODE.
   unsigned int DISTANCE, HEADING, RUDDER_FAN_PW, THRUST_ANGLE_PW,
LEFT_THRUST_POWER_FAN_PW, RIGHT_THRUST_POWER_FAN_FAN, PCA_COUNTER,
TARGET_HEADING, PCA_START, DRIVE_MOTOR_NEUT AND STEERING_SERVO_NEUT.
   int HEADING_ERROR.
   long TEMP_PW
   sbit POT, SS, CF.

function prototypes
   void Port_Init(void);
   void PCA_Init(void);
   void XBR0_Init(void);
   void Interrupt_Init(void);
   void SMB_Init(void);
   void ADC_Init(void);
   void PCA_ISR(void) __interrupt 9;
   void Drive_Fan(void);
   void Speed_Controller(void);
   void Ping_Ranger(void);
   void Read_Ranger(void);
   void Fan_Init(void);
   void Steering_Servo(void);
   void Direction_Controller(void);
   void Read_Compass(void);
   void Steering_Servo_Init(void);
   void Read_AD_Input(void);
   void POT_Reader(void);
   void Read_Keypad(void);
   void Flight_Recorder(void);
   unsigned int Is_SS_On(void);
   void Turn_BILED_Green(void);
   void Turn_BILED_Red(void);
   void Turn_BILED_Off(void);
   void Reset_PCA_Counter(void);
   void Wait_For_1s(void);

main function
   declare local variables
      NONE
   initialize system, ports and PCA_COUNTER
      Sys_Init();
      putchar(' ');
      Port_Init();
```

```
        PCA_Init();
        XBR0_Init();
        Interrupt_Init();
        SMB_Init();
        ADC_Init();
    Turn off the BILED.
    Read the data from POT using POT_Reader.
    initialize the drive motor using Drive_Motor_Init.
    initialize the steering servo using Steering_Servo_Init.
    Begin infinite loop
        If the slide switch is on
            Turn BILED to green
            Set the drive motor PW to forward
            Drive the motor
            Wait for 3s
            Begin infinite loop while the SS is on
                wait for 3 second
                call Read_Ranger
                call Speed_Controller
                call Drive_Motor
                call Direction_Controller
                call Steering_Servo
        End if
        Else if the slide switch is off
            read the input from the keypad
            set DRIVE_MOTOR_PW to DRIVE_MOTOT_MAX
            call Drive_Motor
            set STEERING_SERVO_PW to STEERING_SERVO_NEUT
            call Steering_Servo
        End if
    End infinite loop

functions
    void Drive_Fan(void)
        Set the high and low byte of alll CEXs.
    End function

    void Speed_Controller(void)
    if STAGE is 0
        Run the code to determine the PW for all three fans.
    if STAGE is 1
        Run the code to determine the PW for the left and right fan.
        Set the PW for CEX0 and CEX1 to NEUT.
    if STAGE is 2
        Run the code to determine the PW for the tail fan.
        Set the PW for the left and right fan to NEUT.
    End function

    void Ping_Ranger(void)
        write the register 0 of the ranger to read the DISTANCE in cm.
    End function

    void Read_Ranger(void)
        Read the data from register 2 and 3 from the ranger
```

```
    Convert the read data to DISTANCE
    Use Ping_Ranger to send the signal again
End function

void Fan_Init(void)
    Set the PW of the drive motor to neutal
    Turn the drive motor on for neutral
    Wait for 1s
End function

void Direction_Controller(void)
    Read the data from compass using Read_Compass
    calculate the HEADING_ERROR
    If the HEADING_ERROR is larger 1800
        HEADING_ERROR minus 3600
    End if
    If the HEADING_ERROR is smaller than -1800
        HEADING_ERROR plus 3600
    End if
    Set the corresponding PW for the tail fan.
    If the STAGE is 0
        Calculate the PW for the tail fan direction.
    If the STAGE is 1
        Do nothing
    If the STAGE is 2
        Calculate the PW for the tail fan direction.
End function

void Read_Compass(void)
    Read the data from the compass register 2 and 3
    Convert the read data to HEADING
End function

void Read_AD_Input(void)
    Set P1.4 as the analog input for the POT
    Clear the "Conversion Completed" flag
    Initiate A/D conversion
    Wait for conversion to complete
    Set the AD_VALUE
End function

void POT_Reader(void)
    Read the AD_VALUE using Read_AD_Input().
    calculate the max of the drive motor
    calcualte the min of the drive motor.
    calculate the right PW for the steering servo.
    calculate the left PW for the steering servo.
End function

void Read_Keypad(void)
    Get the target heading from the keypad
    If the value is out of range
        Ask the user to input another value
    End if
```

```
End function

void Flight_Recorder(void)
   if the MODE is not 1
      Output to user-friendly UI.
   End if
   else
      Print TARGET_HEADING, HEADING, HEADING_ERROR, PW
   End if
End function

unsigned int Is_SS_On(void)
   if SS is on
      return 1
   End if
   else if SS is off
      return 0
   End iff
End function

void PCA_Init (void)
   Enable SYSCLK/12 and enable interrupt.
   Enable CCM0, CCM1, CCM2, CCM3 16bit PWM.
   Enable PCA counter.
   Set PCA_START for 20ms period.
End function

XBR0_Init(void)
   Configure crossbar with UART, SMBus, and CEX channels (0x25)
End function

void Interrupt_Init(void)
   Enable general interrupt.
   Enable PCA overflow interrupts.
End function

void SMB_Init(void)
   Set the clock frequency to be 100kHz.
   Enable SMB.
End function

void ADC_Init(void)
   configure ADC1 to use VREF.
   Set a gain of 1.
   Enable ADC1.
End function

void PCA_ISR(void) __interrupt 9
   increment the PCA_COUNTER
   Set STAGE0 = 1.
   if 20s has passed
      set STAGE1 to 1
      set STAGE0 to 0.
   End if
```

```
    if 40s has passed
        set STAGE2 to 1.
        Set STAGE1 to 0.
    if 1s has passed and switch is on
        call Flight_Recorder
    End if
    if CF
        Clear overflow flag.
        start count for 20ms period
    End if
    handle other PCA0 overflows
End function

void Reset_PCA_Counter(void)
    reset PCA_COUNTER to 0
End function

void Wait_For_1s (void)
    reset PCA_COUNTER
    Begin infinite loop while PCA_COUNTER less than 51
    wait for 1s
    End infinite loop
    reset PCA_COUNTER
End function
```