PSEUDOCODE

```
Init Keypad                    // Col[3:0] = P3[7:4];  Row[3:0] = P3[3:0]
Init I2C                       // SDA = P1.2; SCL = P1.3
Init LDC
Init PWM signal                // PWM output = P6.1 (TB3.2)
Set key_val = 0                // Stores input char. 0 if no key input or err
Set speed   = 0                // Stores DC motor speed [%]

repeat
      while (key_val = get_key()) == 0 then
            /* wait for a key to be pressed */
      end

      if speed = 0 then        // reset LCD when speed is reset to 0
            reset_LCD()
      end

      if (key_val >= '0' and key_val <= '9') then     // input is digit
            LCD_Send(key_val)                         // write char to LCD
            speed = (speed * 10) + (key_val – '0')    // update speed value
            if speed > 100 then
                  LCD_SetCursor(0,1)            // Move cursor to next line
                  LCD_write("Invalid Speed!")   // Error message
                  speed = 0                     // reset speed and restart
            end

      else if key_val == '#' then          // # is enter speed key; input end
            LCD_SetCursor(0,1)             // Move cursor to next line
            LCD_write("DONE")              // Completion message
            set_duty_cycle(speed)          // Set motor speed (= PWM duty)
            speed = 0                      // reset speed and restart

      else                                 // Invalid input key
            LCD_SetCursor(0,1)             // Move cursor to next line
            LCD_write("Invalid Input!")    // Error message
            speed = 0                      // reset speed and restart
      end
end
```

C CODE:

1) main.c

```c
#include <msp430.h>
#include <stdio.h>
#include "LiquidCrystal_I2C.h"
#include "keypad_4x4.h"
#include "pwm.h"

void reset_LCD(void);

/**
 * main.c
 */
void main(void){
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

    // INIT & SETUP //

    /* keypad */
    unsigned char key_val = 0;
    keypad_init();

    /* I2C & LCD */
    I2C_init(0x27); // 0x27 signifies the slave address (address of the LCD with the I/O expander).
    LCD_Setup();
    reset_LCD();

    /* PWM signal (for motor) */
    pwm_init();


    /* Other Vars */
    unsigned short speed = 0;


    // MAIN LOOP //
    while(1){
        while((key_val = get_key())==0);            //Wait until a key Pressed

        if(speed == 0){
            reset_LCD();
        }

        if(key_val >= '0' && key_val <= '9'){ /* Enter digits  */
            LCD_Send((int)key_val, Rs | LCD_BACKLIGHT); // send digit to display.

            speed = (speed*10) + (key_val - '0');       // compute speed

            if(speed > 100){ /* Error if speed > 100% */
                LCD_SetCursor(0, 1);    // Set cursor to position (1,0)
                LCD_Write("Invalid Speed!");
                speed = 0;
            }
        }
        else if(key_val == '#'){ /* End input */
            LCD_SetCursor(0, 1);    // Set cursor to position (1,0)
            LCD_Write("DONE");

            /* set new PWM */
            set_duty_cycle(speed);

            speed = 0;
        }
        else{ /* Error if input not digit nor '#' */
            LCD_SetCursor(0, 1);    // Set cursor to position (1,0)
            LCD_Write("Invalid Input!");
            speed = 0;
        }
```

```
    }

    return 0;
}

void reset_LCD(void){
    LCD_ClearDisplay();     // Clear display
    LCD_SetCursor(0, 0);    // Set cursor to position (0,0)
    LCD_Write("Speed: 0  %");
    LCD_SetCursor(7, 0);    // Set cursor to position (7,0)
}
```

2) pwm.h

```
/*
 * pwm.h
 *
 * Set-up/init and generates PWM signal
 */

#ifndef PWM_H_
#define PWM_H_

// Declare #defined MACROS
#define KEYPAD_ROW P3OUT
#define KEYPAD_COL (P3IN  & 0xF0)

// Declare the functions in the header:
void pwm_init(void);
void set_duty_cycle(unsigned short duty_cycle);

#endif /* PWM_H_ */
```

3) pwm.c

```
/*
 * pwm.c
 *
 * Set-up/init and generates PWM signal for DC Motor
 */

#include <msp430.h>
#include "pwm.h"

// Using Timer3_B, CC2, and P6.1 for PWM

// Init PWM & port
void pwm_init(void){
    //-- Configure GPIO ------------------------------------------------
    P6DIR |= BIT1;                      // P6.1 output
    P6SEL0 |= BIT1;                     // P6.1  options select (TB3.2)

    //-- Disable the GPIO power-on default high-impedance mode ----------
    PM5CTL0 &= ~LOCKLPM5;

    //-- Setup Timer3_B CCR --------------------------------------------
    TB3CCR0 = 600;                      // PWM Period ~18.3ms or FSreq 54.6Hz
    TB3CCTL2 = OUTMOD_7;                // CCR2 reset/set
    TB3CCR2 = 0;                        // CCR2 PWM duty cycle

    //-- Setup Timer3_B ------------------------------------------------
    TB3CTL = TBSSEL_1 | MC_1 | TBCLR;   // ACLK, up mode, clear TBR
}

void set_duty_cycle(unsigned short duty_cycle){
    TB3CCR2 = 6*duty_cycle;
}
```

4) keypad_4x4.h

```c
/*
 * keypad_4x4.h
 *
 * Init and read 4x4 keypad inputs
 */

#ifndef KEYPAD_4X4_H_
#define KEYPAD_4X4_H_

// Declare #defined MACROS
#define KEYPAD_ROW P3OUT
#define KEYPAD_COL (P3IN  & 0xF0)

// Declare the functions in the header:
void keypad_init(void);
unsigned char get_key(void);

#endif /* KEYPAD_4X4_H_ */
```

5) keypad_4x4.c

```c
/*
 * keypad_4x4.c
 *
 * Init and read 4x4 keypad inputs
 */

#include <msp430.h>
#include "keypad_4x4.h"

// USING PORT P3

unsigned char key_arr[] = {'1','2','3','A',
                           '4','5','6','B',
                           '7','8','9','C',
                           '*','0','#','D'};
// Init keypad ports
void keypad_init(void){
    //-- Configure GPIO ----------------------------------------------
    P3DIR = 0x0F; // Set P3.0 to 3.3 Output, Set P3.4 to 3.7 Input
    P3REN = 0xFF; // Set P3.0 to 3.7 enable pull up/down
    P3OUT = 0xF0; // Set P3.0 to 3.3 output to 0; Select P3.4 to 3.7 as pull-up inputs

    PM5CTL0 &= ~LOCKLPM5;       // Turn on GPIO
}

// Read keys from keypad
// If key is pressed, return respective char. Otherwise, return 0.
unsigned char get_key(void){
    unsigned int r,key = 0;

    KEYPAD_ROW = 0xF0;
    if(KEYPAD_COL == 0xF0){
        return 0; /* no key pressed */
    }

    /* Else: key is presses */
    for(r=0; r<4 ; r++){
        KEYPAD_ROW = ~(0x01<<r);    // Scan for a Key by sending '0' on KEYPAD_ROW
        if(KEYPAD_COL == 0xF0){         // Skip row if all KEYPAD_COL bits are '1'
            key += 4;
            continue;
        }
        else if(KEYPAD_COL == 0xE0){    // Col 0
            key += 0;
        }
        else if(KEYPAD_COL == 0xD0){    // Col 1
            key += 1;
        }
```

```
        else if(KEYPAD_COL == 0xB0){    // Col 2
            key += 2;
        }
        else if(KEYPAD_COL == 0x70){    // Col 3
            key += 3;
        }
        else{                           // Error or key not presses
            return 0;
        }
        while(KEYPAD_COL != 0xF0);
        __delay_cycles(100000);             // wait ~0.1 sec for debouncing
        return key_arr[key];
    }
    return 0;  // Error or key not presses
}
```

6) LiquidCystal_I2C.h

```
/*
 * LiquidCrystal_I2C.h
 *
 * Init and controls I2C & LCD
 */

#ifndef LIQUIDCRYSTAL_I2C_H_
#define LIQUIDCRYSTAL_I2C_H_

extern unsigned int TXBUF;

#define LCD_4_BIT 0x20
#define LCD_init  0x30

// commands
#define LCD_CLEARDISPLAY 0x01       // These are shortcuts for the commands in hex.
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10       // These are standard for 2x16
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

```

```
// flags for backlight control
#define LCD_BACKLIGHT 0x08
#define LCD_NOBACKLIGHT 0x00

#define En 0x04  // Enable bit
#define Rw 0x02  // Read/Write bit
#define Rs 0x01  // Register select bit

// Declare the functions in the header:
void I2C_init(int addr);        // Address of the device in the argument.
void I2C_Send (int value);
void pulseEnable (int value);
void write4bits (int value);
void LCD_Send(int value, int mode) ;
void LCD_Write (char *text);
void LCD_WriteNum(unsigned int num);
void LCD_SetCursor(int col, int row);
void LCD_ClearDisplay(void);
void LCD_leftToRight(void) ;
void LCD_rightToLeft(void) ;
void LCD_Setup(void);

#endif /* LIQUIDCRYSTAL_I2C_H_ */
```

7) LiquidCystal_I2C.c

```
/*
 * LiquidCrystal_I2C.c
 *
 * Init and controls I2C & LCD
 */

#include <msp430.h>
#include <string.h>
#include "LiquidCrystal_I2C.h"

// USING eUSCI_B0

unsigned int TXBUF;             // The value to put in the buffer to transmit information.

// Init I2C & Ports
void I2C_init(int addr){
    UCB0CTLW0 |= UCSWRST;       // Put eUSCI_B0 into software reset

    //-- Configure uUSCI_B0 ------------------------------------------
    UCB0CTLW0 |= UCSSEL_3;      // Choose BRCLK = SMCLK = 1MHz
    UCB0BRW = 10;               // Divide BRCLK by 10 for SCL = 100kHz

    UCB0CTLW0 |= UCMODE_3;      // Put into I2C Mode
    UCB0CTLW0 |= UCMST;         // Put into Master Mode
    UCB0CTLW0 |= UCTR;          // Put into Tx Mode
    UCB0I2CSA = addr;           // Slave Address = 0x27

    UCB0CTLW1 |= UCASTP_2;      // Auto stop when UCB0TBCNT reached
    UCB0TBCNT =0x01;            // Send 1 byte of data

    //-- Configure GPIO ----------------------------------------------
    P1SEL1 &= ~BIT3;            // P1.3 = SCL
    P1SEL0 |=  BIT3;

    P1SEL1 &= ~BIT2;            // P1.2 = SDA
    P1SEL0 |=  BIT2;

    PM5CTL0 &= ~LOCKLPM5;       // Turn on GPIO

    //-- Take eUSCI_B0 out of SW reset -------------------------------
    UCB0CTLW0 &= ~UCSWRST;      // Put eUSCI_B0 out of software reset

    UCB0IE |= UCTXIE0;          // Enable I2C_B0 Tx IRQ
    __enable_interrupt();
}

```

```c
// send value by I2C
void I2C_Send (int value){
    UCB0CTLW0 |= UCTXSTT;       // Generate a START condition.
    TXBUF = value;              // Put what you want (store the data) to transmit into the Tx buffer register. See the interrupt vector below.
}

void pulseEnable (int value){
    I2C_Send (value | En);      // En high
    __delay_cycles(150);        // enable pulse must be >450ns
    I2C_Send(value  & ~En);     // En low
    __delay_cycles(1500);       // commands need > 37us to settle
}

// sends 4 bits by I2C to LCD
void write4bits (int value) {
    I2C_Send (value);
    __delay_cycles(50);
    pulseEnable (value);
}

// Sends 1 char to LCD
void LCD_Send(int value, int mode) {
    int high_b = value & 0xF0;
    int low_b = (value << 4) & 0xF0;            // Shift the bits to the left and then set all bits except the ones in 0xF0 to 0.
    write4bits ( high_b | mode);                // write4bits is a function call with one arg and the arg is the result of a bitwise or | (one pipe symbol).
    write4bits ( low_b  | mode);                // The arg of write4bits uses 4 bits for the value and 4 bits for the mode.
                                                // It is being called first with high bits, then with th elow bits to write 8 bits.
}

// Send string to LCD
void LCD_Write (char *text){
    unsigned int i;
    for (i=0; i < strlen(text); i++){
        LCD_Send((int)text[i], Rs | LCD_BACKLIGHT);
    }
}

// Send decimal number to LCD
void LCD_WriteNum(unsigned int num) {
    unsigned int reverseNum = 0;
    unsigned int digits = 0;                    // To use as a digit counter. For now, no digits are counted yet until we enter the first while loop.
    int i;                                      // This is for the for loop to run digits iterations.

    if (num == 0) {                             // If the user input 0 on the keypad...
        LCD_Send(0 | 0x30, Rs | LCD_BACKLIGHT); // ...then display 0 for 0% duty cycle.
    }
    else {

        while (num > 0) {
            reverseNum = reverseNum * 10 + (num % 10);
            num /= 10;
            digits++;                           // Increment digits; this means it is counting how many digits the user input from the keypad.
        }

        for(i = 0; i < digits; ++i) {           /* It will run digits iterations while it does the modulo and division operation. Now it knows how many digits
                                                   it will print. This fixes the zeroes issue; now it can display #0 and 100 on the LCD successfully. */
            LCD_Send((reverseNum % 10) | 0x30, Rs | LCD_BACKLIGHT);
            reverseNum /= 10;
        }
    }
}

// Set LCD cursor position
void LCD_SetCursor(int col, int row) {      // This function converts a column and row to a single number the LCD is expecting to set the cursor position.
    int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 }; // The LCD must have an interface where the starting position of each row is 0x00, 0x40, 0x14, and 0x54.
    LCD_Send(LCD_SETDDRAMADDR | (col + row_offsets[row]),  LCD_BACKLIGHT);
}

// Clear LCD
void LCD_ClearDisplay(void){
    LCD_Send(LCD_CLEARDISPLAY,  LCD_BACKLIGHT);
    __delay_cycles(50);
}

void LCD_leftToRight(void) {
    LCD_Send(LCD_ENTRYMODESET | LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT,  LCD_BACKLIGHT);
}

void LCD_rightToLeft(void) {
    LCD_Send(LCD_ENTRYMODESET | LCD_ENTRYRIGHT | LCD_ENTRYSHIFTDECREMENT,  LCD_BACKLIGHT);
}

// Setup LCD
void LCD_Setup(void){
    int _init[] = {LCD_init, LCD_init, LCD_init, LCD_4_BIT};
    int _setup[5];
    int mode = LCD_BACKLIGHT;
    _setup[0] = LCD_FUNCTIONSET | LCD_4BITMODE | LCD_2LINE | LCD_5x8DOTS;
    _setup[1] = LCD_CLEARDISPLAY;
    _setup[2] = LCD_RETURNHOME;
    _setup[3] = LCD_ENTRYMODESET | LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
    _setup[4] = LCD_DISPLAYCONTROL | LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;

    write4bits(_init[0]);   // Waiting for the enable function to be written.
    __delay_cycles(108000);// __delay_cycles(4500*us);  <--- equivalent to this.      It is the value we need to establish between the enable.
    write4bits(_init[1]);
    __delay_cycles(108000);
    write4bits(_init[2]);
    __delay_cycles(3600);
    write4bits(_init[3]);

    LCD_Send(_setup[0], mode);
    LCD_Send(_setup[1], mode);
    __delay_cycles(50);
    LCD_Send(_setup[2], mode);
    LCD_Send(_setup[3], mode);
    LCD_Send(_setup[4], mode);
}

#pragma vector = EUSCI_B0_VECTOR
__interrupt void EUSCI_B0_I2C_ISR(void){
    UCB0TXBUF = TXBUF;                      // The value that we want to transmit is in the buffer.
}
```