```
// PIC18F4520 Configuration Bit Settings

// 'C' source line config statements

// CONFIG1H
#pragma config OSC = INTIO67    // Oscillator Selection bits (Internal oscillator block, port function on RA6 and RA7)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)

// CONFIG2L
#pragma config PWRT = ON        // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 3         // Brown Out Reset Voltage bits (Minimum setting)

// CONFIG2H
#pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC   // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = OFF     // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital I/O on Reset)
#pragma config LPT1OSC = OFF    // Low-Power Timer1 Oscillator Enable bit (Timer1 configured for higher power operation)
#pragma config MCLRE = ON       // MCLR Pin Enable bit (RE3 input pin enabled; MCLR disabled)

// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection bit (Block 0 (000800-001FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection bit (Block 1 (002000-003FFFh) not code-protected)
#pragma config CP2 = OFF        // Code Protection bit (Block 2 (004000-005FFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection bit (Block 3 (006000-007FFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
```

```
#pragma config CPD = OFF       // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF      // Write Protection bit (Block 0 (000800-001FFFh) not write-protected)
#pragma config WRT1 = OFF      // Write Protection bit (Block 1 (002000-003FFFh) not write-protected)
#pragma config WRT2 = OFF      // Write Protection bit (Block 2 (004000-005FFFh) not write-protected)
#pragma config WRT3 = OFF      // Write Protection bit (Block 3 (006000-007FFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF      // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF      // Boot Block Write Protection bit (Boot block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF      // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF     // Table Read Protection bit (Block 0 (000800-001FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF     // Table Read Protection bit (Block 1 (002000-003FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR2 = OFF     // Table Read Protection bit (Block 2 (004000-005FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF     // Table Read Protection bit (Block 3 (006000-007FFFh) not protected from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF     // Boot Block Table Read Protection bit (Boot block (000000-0007FFh) not protected from table reads executed in other blocks)

/*
 * File:   main.c
 * Author: KAKA
 *
 * PIC18F4520 Stepper Motor Control with Keypad Interface
 * - Crystal frequency: 4MHz
 * - Keypad: 4x4 matrix connected to PORTB with external pull-up resistors on columns
 * - LCD: 8-bit mode connected to PORTD, RS=RC1, EN=RC2
 * - Stepper Motor: Half stepping via ULN2003 connected to RA0-RA3
 */

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define _XTAL_FREQ 4000000     // 4MHz Crystal Oscillator
```

```
// LCD pins
#define LCD_RS      PORTCbits.RC1  // Register Select
#define LCD_EN      PORTCbits.RC2  // Enable
#define LCD_DATA    PORTD          // LCD data port

// Keypad configuration - Definition of rows and columns for clarity
#define ROW1 0x01  // RB0
#define ROW2 0x02  // RB1
#define ROW3 0x04  // RB2
#define ROW4 0x08  // RB3
#define COL1 0x10  // RB4
#define COL2 0x20  // RB5
#define COL3 0x40  // RB6
#define COL4 0x80  // RB7

// Stepper motor pins
#define STEPPER_PORT PORTA
#define STEPPER_TRIS TRISA

// LCD Commands
#define LCD_CLEAR          0x01
#define LCD_RETURN_HOME    0x02
#define LCD_ENTRY_MODE     0x06
#define LCD_DISPLAY_ON     0x0C
#define LCD_FUNCTION_8BIT  0x38
#define LCD_LINE1          0x80
#define LCD_LINE2          0xC0

// Function prototypes
void initialize(void);
void LCD_Command(unsigned char cmd);
void LCD_Data(unsigned char data);
void LCD_Init(void);
void LCD_String(const char *str);
void LCD_Clear(void);
void LCD_SetCursor(unsigned char row, unsigned char col);
unsigned char Keypad_Scan(void);
void Stepper_Rotate(unsigned int steps);
void Display_Rotation_Info(char key1, char key2, unsigned int angle_times);

// Half step sequence for stepper motor
unsigned char step_sequence[8] = {
   0b0001, // Step 1: IN1 = 1, IN2 = 0, IN3 = 0, IN4 = 0
   0b0011, // Step 2: IN1 = 1, IN2 = 1, IN3 = 0, IN4 = 0
   0b0010, // Step 3: IN1 = 0, IN2 = 1, IN3 = 0, IN4 = 0
   0b0110, // Step 4: IN1 = 0, IN2 = 1, IN3 = 1, IN4 = 0
   0b0100, // Step 5: IN1 = 0, IN2 = 0, IN3 = 1, IN4 = 0
   0b1100, // Step 6: IN1 = 0, IN2 = 0, IN3 = 1, IN4 = 1
   0b1000, // Step 7: IN1 = 0, IN2 = 0, IN3 = 0, IN4 = 1
   0b1001  // Step 8: IN1 = 1, IN2 = 0, IN3 = 0, IN4 = 1
};
```

```
// Main function
void main(void) {
    OSCCON = 0xEF;
    initialize();

    char key1 = 0, key2 = 0;
    unsigned char keyCount = 0;
    unsigned int rotationTimes = 0;

    LCD_Clear();
    LCD_String("Enter Key Seq:");
    LCD_SetCursor(1, 0);

    while (1) {
        unsigned char key = Keypad_Scan();

        if (key != 0) {
            // Display the key on LCD for debugging
            LCD_Data(key);

            if (keyCount == 0) {
                key1 = key;
                keyCount = 1;
            } else if (keyCount == 1) {
                key2 = key;
                keyCount = 2;

                // Determine rotation angle based on key combination
                if ((key1 == '1' && key2 == '8') || (key1 == '8' && key2 == '1')) {
                    rotationTimes = 1;
                } else if ((key1 == '2' && key2 == '3') || (key1 == '3' && key2 == '2')) {
                    rotationTimes = 2;
                } else if ((key1 == '4' && key2 == '6') || (key1 == '6' && key2 == '4')) {
                    rotationTimes = 3;
                } else if ((key1 == '7' && key2 == '9') || (key1 == '9' && key2 == '7')) {
                    rotationTimes = 4;
                } else if ((key1 == 'A' && key2 == 'C') || (key1 == 'C' && key2 == 'A')) {
                    rotationTimes = 5;
                } else if ((key1 == 'D' && key2 == 'B') || (key1 == 'B' && key2 == 'D')) {
                    rotationTimes = 6;
                } else if ((key1 == '5' && key2 == '#') || (key1 == '#' && key2 == '5')) {
                    rotationTimes = 7;
                } else if ((key1 == '0' && key2 == '*') || (key1 == '*' && key2 == '0')) {
                    rotationTimes = 8;
                } else {
                    // Invalid key combination
                    LCD_Clear();
                    LCD_String("Invalid Keys!");
                    __delay_ms(3000);
                    LCD_Clear();
                    LCD_String("Enter Key Seq:");
```

```
            LCD_SetCursor(1, 0);
            keyCount = 0;
            continue;
        }

        // Display rotation info and rotate motor
        Display_Rotation_Info(key1, key2, rotationTimes);

        // Rotate stepper motor - 45 degrees = 50 steps for half stepping (approx)
        Stepper_Rotate(rotationTimes * 1);

        // Reset for next input
        __delay_ms(2000);
        LCD_Clear();
        LCD_String("Enter Key Seq:");
        LCD_SetCursor(1, 0);
        keyCount = 0;
      }

      // Debounce delay
      __delay_ms(300);

      // Wait until key is released
      while(Keypad_Scan() != 0) {
        __delay_ms(10);
      }
    }
  }
}

// Initialize MCU
void initialize(void) {
  // Configure port direction registers
  TRISA = 0x00;     // PORTA all outputs (stepper motor)
  TRISB = 0xF0;     // RB0-RB3 outputs (rows), RB4-RB7 inputs (columns)
  TRISC = 0x00;     // PORTC all outputs (LCD control)
  TRISD = 0x00;     // PORTD all outputs (LCD data)

  // Initialize ports
  PORTA = 0x00;     // Clear stepper motor outputs
  PORTB = 0x0F;     // Set row outputs high initially
  PORTC = 0x00;     // Clear LCD control signals
  PORTD = 0x00;     // Clear LCD data bus

  // Disable analog functionality on PORTB if needed
  ADCON1 = 0xFF;    // All pins digital

  // Initialize LCD
  LCD_Init();

  // Initial display
  LCD_Clear();
```

```
    __delay_ms(800);
    LCD_String("System Ready!!!");
    __delay_ms(2000);
}

// Send command to LCD
void LCD_Command(unsigned char cmd) {
    LCD_RS = 0;         // Command mode
    LCD_DATA = cmd;     // Send command
    LCD_EN = 1;         // Enable pulse
    __delay_ms(1);      // Delay
    LCD_EN = 0;         // Disable pulse
    __delay_ms(3);      // Wait for command execution
}

// Send data to LCD
void LCD_Data(unsigned char data) {
    LCD_RS = 1;         // Data mode
    LCD_DATA = data;    // Send data
    LCD_EN = 1;         // Enable pulse
    __delay_ms(1);      // Delay
    LCD_EN = 0;         // Disable pulse
    __delay_ms(1);      // Wait for data processing
}

// Initialize LCD
void LCD_Init(void) {
    __delay_ms(20);     // Wait for LCD to stabilize

    LCD_Command(LCD_FUNCTION_8BIT);  // 8-bit mode, 2 lines, 5x8 dots
    LCD_Command(LCD_DISPLAY_ON);     // Display on, cursor off
    LCD_Command(LCD_ENTRY_MODE);     // Auto increment cursor position
    LCD_Command(LCD_CLEAR);          // Clear display
    __delay_ms(50);                  // Wait for display to clear
}

// Display string on LCD
void LCD_String(const char *str) {
    while (*str) {
        LCD_Data(*str++);
    }
}

// Clear LCD display
void LCD_Clear(void) {
    LCD_Command(LCD_CLEAR);
    __delay_ms(2);
}

// Set LCD cursor position
void LCD_SetCursor(unsigned char row, unsigned char col) {
    if (row == 0) {
```

```
        LCD_Command(LCD_LINE1 + col);
    } else {
        LCD_Command(LCD_LINE2 + col);
    }
}

// Scan keypad for pressed key - Completely rewritten for reliability
unsigned char Keypad_Scan(void) {
    // Keymap definition - standard 4x4 keypad layout
    const unsigned char keyMap[4][4] = {
        {'1', '2', '3', 'A'},
        {'4', '5', '6', 'B'},
        {'7', '8', '9', 'C'},
        {'*', '0', '#', 'D'}
    };

    unsigned char row, col;

    // Check each row one by one
    for(row = 0; row < 4; row++) {
        // Set all rows high
        PORTB = 0x0F;

        // Set current row low
        PORTB &= ~(1 << row);

        // Small delay for signal to stabilize
        __delay_us(50);

        // Read columns
        // When a key is pressed, the corresponding column reads low
        // due to the connection through the keypad matrix

        // Check column 1 (RB4)
        if(!(PORTB & COL1)) {
            __delay_ms(10); // Debounce
            if(!(PORTB & COL1)) { // Double check
                while(!(PORTB & COL1)); // Wait for release
                return keyMap[row][0];
            }
        }

        // Check column 2 (RB5)
        if(!(PORTB & COL2)) {
            __delay_ms(10); // Debounce
            if(!(PORTB & COL2)) { // Double check
                while(!(PORTB & COL2)); // Wait for release
                return keyMap[row][1];
            }
        }

        // Check column 3 (RB6)
```

```
        if(!(PORTB & COL3)) {
            __delay_ms(10); // Debounce
            if(!(PORTB & COL3)) { // Double check
                while(!(PORTB & COL3)); // Wait for release
                return keyMap[row][2];
            }
        }

        // Check column 4 (RB7)
        if(!(PORTB & COL4)) {
            __delay_ms(10); // Debounce
            if(!(PORTB & COL4)) { // Double check
                while(!(PORTB & COL4)); // Wait for release
                return keyMap[row][3];
            }
        }
    }

    return 0; // No key pressed
}

// Rotate stepper motor specified number of steps
void Stepper_Rotate(unsigned int steps) {
    unsigned int i;

    for (i = 0; i < steps; i++) {
        STEPPER_PORT = (STEPPER_PORT & 0xF0) | step_sequence[i % 8];
        __delay_ms(500);  // Adjust delay for desired speed
    }

    // Turn off all coils to save power
    STEPPER_PORT &= 0xF0;
}

// Display rotation info on LCD
void Display_Rotation_Info(char key1, char key2, unsigned int angle_times) {
    LCD_Clear();
    __delay_ms(600);
    LCD_String("Keys: ");
    __delay_ms(400);
    LCD_Data(key1);
    LCD_Data(key2);

    LCD_SetCursor(1, 0);
    char buffer[16];
    __delay_ms(200);
    sprintf(buffer, "Rotate: %d x 45deg", angle_times);
    __delay_ms(300);
    LCD_String(buffer);
}
```