```c
/*
 * FileName: main.c
 * Version: 1
 *
 * Created: 11/9/2022 1:54 PM
 * Author: Ethan Zeronik
 *
 * Operations: lcd test
 *
 * Hardware:
 *   Atmega2560          micro controller
 *   PORTL               LCD data
 *   PORTD.0             LCD RS
 *   PORTD.1             LCD R/W
 *   PORTD.2             LCD E
 */

/* NOTE: Includes */
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>

#include "LiquidCrystalDisplay.h"

/* NOTE: Custom Macros */
// TODO: None

/* NOTE: Global Variables */
char message[] = {"Test"};

/* NOTE: Function prototypes */
// inits IO ports
void IO_init(void);

/* NOTE: Application implementation */
// the main loop of the function, provided to us
int main(void)
{
    IO_init();
    LCD_init(&DDRD, &PORTD, &DDRL, &PORTL);

    LCD_sendInstruction(0x01);
    _delay_ms(2); // can remove if use Busy Flag check

    LCD_sendInstruction(0x02);
    _delay_ms(2); // can remove if use Busy Flag check

    LCD_sendInstruction(0x86);
    _delay_ms(50);

    LCD_sendString(message);
    _delay_ms(50);

    LCD_sendInstruction(0xC6);
    _delay_us(50); // can remove if use Busy Flag check

    LCD_sendString(message);

    while(1)
    {
    }
}

/* NOTE: Function implementations */
void IO_init(void)
{
    // do nothing
}

```

```c
/*
 * FileName: main.c
 * Version: 1
 *
 * Created: 11/2/2022 1:39:55 PM
 * Author: Ethan Zeronik
 *
 * Operations: basic serial
 *
 * Hardware:
 *   Atmega2560          micro controller
 *   PORTA               LED bar
 *   PORTC               Button/Switches
 */

/* NOTE: Includes */
#include <avr/io.h>
#include <avr/interrupt.h>

#include "Serial.h"

/* NOTE: Custom Macros */
// TODO: None

/* NOTE: Global Variables */
// TODO: None

/* NOTE: Function prototypes */
// inits IO ports
void IO_init(void);
// handler
void asyncGetHandler(char c);

/* NOTE: Application implementation */
// the main loop of the function, provided to us
int main(void)
{
    IO_init();

    // init async uart and bind an interrupt handler
    SERIAL_uartInitAsync(USART0, 9600);
    SERIAL_uartAsyncGetHandler(USART0, &asyncGetHandler);

    sei();

    while(1)
    {
        SERIAL_uartSendFixed(USART0, (char const * const)&PINC, 1);
    }
}

/* NOTE: Function implementations */
void IO_init(void)
{
    // set portA as an output
    DDRA  = 0xFF;
    PORTA = 0x00;

    // set portC as an input
    DDRC  = 0x00;
    PORTC = 0xFF;
}

void asyncGetHandler(char c)
{
    PORTA = c;
}

```

```c
/*
 * FileName: main.c
 * Version: 1
 *
 * Created: 11/9/2022 2:26:50 PM
 * Author: Ethan Zeronik
 *
 * Operations: uart to lcd
 *
 * Hardware:
 *   Atmega2560          micro controller
 *   PORTL               LCD data
 *   PORTD.0             LCD RS
 *   PORTD.1             LCD R/W
 *   PORTD.2             LCD E
 */

/* NOTE: Includes */
#include <avr/io.h>
#include <avr/interrupt.h>

#include "LiquidCrystalDisplay.h"
#include "Serial.h"

/* NOTE: Custom Macros */
// TODO: None

/* NOTE: Global Variables */
// buffer for uart
char    message[24]  = {0};
uint8_t messageIndex = 0;
uint8_t readFlag     = 0;

/* NOTE: Function prototypes */
// inits IO ports
void IO_init(void);
// handler
void asyncGetHandler(char c);

/* NOTE: Application implementation */
// the main loop of the function, provided to us
int main(void)
{
    IO_init();
    LCD_init(&DDRD, &PORTD, &DDRL, &PORTL);

    // init async uart and bind an interrupt handler
    SERIAL_uartInitAsync(USART0, 9600);
    SERIAL_uartAsyncGetHandler(USART0, &asyncGetHandler);

    sei();

    LCD_sendInstruction(0x01);
    LCD_sendInstruction(0x02);

    while(1)
    {
        if(readFlag)
        {
            LCD_sendInstruction(0x01);
            LCD_sendInstruction(0x02);
            LCD_sendString(message);

            readFlag = 0;
        }
    }
}

/* NOTE: Function implementations */
void IO_init(void)
{
    // do nothing
}

void asyncGetHandler(char c)
{
    if(c != 0x0d && c != 0x0a && c != '\0')
    {
        if(messageIndex < 23)
        {
            // add to array
            message[messageIndex]     = c;
            message[messageIndex + 1] = '\0';

            messageIndex++;
        }
    }
    else
    {
        for(uint8_t i = 0; i < 24; i++)
        {
            message[i] = '\0';
        }

        messageIndex = 0;
    }

    // set update flag
    readFlag = 1;
}

```

```c
/*
 * FileName: LiquidCrystalDisplay.h
 * Version: 1
 *
 * Created: 11/9/22 1:59 PM
 * Author: Ethan Zeronik
 *
 * Operations: lcd definition
 */

#ifndef LiquidCrystalDisplay_h_INCLUDED
#define LiquidCrystalDisplay_h_INCLUDED

#if defined(__cplusplus)
extern "C" {
#endif

#pragma message("WARNING: this module uses lower 3 bits of control and entire data buffer assigned")

#include <stdint.h>

/* NOTE: Custom Types */
// custom type for charcater definition
typedef uint8_t LcdCustomCharacter_t[8];
// typing for the allowed character addresses
typedef enum LcdCharacterAddress_t
{
    // the first charcater slot
    lcdFirstSlot   = 0x40,
    // the second charcater slot
    lcdSecondSlot  = 0x48,
    // the third charcater slot
    lcdThirdSlot   = 0x50,
    // the fourth charcater slot
    lcdFourthSlot  = 0x58,
    // the fifth charcater slot
    lcdFifthSlot   = 0x60,
    // the sixth charcater slot
    lcdSixthSlot   = 0x68,
    // the seventh charcater slot
    lcdSeventhSlot = 0x70,
    // the eighth charcater slot
    lcdEighthSlot  = 0x78,
} LcdCharacterAddress_t;

/* NOTE: Function prototypes */
// init for the lcd
// the data register is entirely used
// the control register is only used for the lower 3 bits
void LCD_init(uint8_t volatile * const pControlRegister, uint8_t volatile * const pControlPort, uint8_t volatile * const pDataRegister, uint8_t volatile * const pDataPort);
// sends the given instruction
void LCD_sendInstruction(uint8_t input);
// send a single char
void LCD_sendChar(char c);
// send a string
void LCD_sendString(char const * const pData);
// creates the custom character at the given address
void LCD_createCharacter(LcdCharacterAddress_t address, LcdCustomCharacter_t custom);
// gets the created character
// to be used as a char in a astring
char LCD_getCharacter(LcdCharacterAddress_t address);

#if defined(__cplusplus)
} /* extern "C" */
#endif

#endif // LiquidCrystalDisplay_h_INCLUDED
```

```c
/*
 * FileName: LiquidCrystalDisplay.c
 * Version: 1
 *
 * Created: 11/9/22 1:59 PM
 * Author: Ethan Zeronik
 *
 * Operations: lcd implementation
 */

/* NOTE: Includes */
#include "LiquidCrystalDisplay.h"

#if !defined(F_CPU)
    #define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <util/delay.h>

/* NOTE: Global Variables */
// instance pointer to the control logic
static uint8_t * sContolPort;
// instance pointer to the data port
static uint8_t * sDataPort;

/* NOTE: Local function implementations */
void LCD_init(uint8_t volatile * const pControlRegister, uint8_t volatile * const pControlPort, uint8_t volatile * const pDataRegister, uint8_t volatile * const pDataPort)
{
    // configure port register and turn off port
    *pDataRegister |= 0xff;
    *pDataPort = 0x00;

    // configure port register and turn off port
    *pControlRegister |= 0x07;
    *pControlPort = (*pControlPort & 0xf8) | 0x00;

    sContolPort = (uint8_t *)pControlPort;
    sDataPort   = (uint8_t *)pDataPort;

    // wait for lcd to power up
    _delay_ms(35);

    // set lcd to 8 bits, 2 lines, display off
    LCD_sendInstruction(0x38);
    _delay_us(50);

    // set lcd to display on, cursor off, blink off
    LCD_sendInstruction(0x0C);
    _delay_us(50);

    // clear the display
    LCD_sendInstruction(0x01);
    _delay_ms(2);

    // incrmement mode
    LCD_sendInstruction(0x06);
}

void LCD_sendInstruction(uint8_t input)
{
    // set controls to RS = 0 E = 0, R/!W=0 then take E high
    *sContolPort = (*sContolPort & 0xf8) | 0x00;
    *sContolPort |= 0x04;

    // send data then delay for at least 50us
    *sDataPort = input;
    _delay_us(50);

    // take E low
    *sContolPort = *sContolPort & 0xf9;

    _delay_ms(5);
}

void LCD_sendChar(char c)
{
    // set controls to RS = 1 E = 0, R/!W=0 then take E high
    *sContolPort = (*sContolPort & 0xf8) | 0x01;
    *sContolPort = *sContolPort | 0x04;

    // send data then delay for at least 50us
    *sDataPort = c;
    _delay_us(50);

    // take E low
    *sContolPort = (*sContolPort & 0xf8) | 0x01;

    _delay_ms(5);
}

void LCD_sendString(char const * const pData)
{
    char * localPointer = (char * const)pData;

    // set controls to RS = 1 E = 0, R/!W=0
    *sContolPort = (*sContolPort & 0xf8) | 0x01;

    while(*localPointer != '\0')
    {
        // take E high
        *sContolPort = *sContolPort | 0x04;

        // send data then delay for at least 50us
        *sDataPort = *localPointer++;
        _delay_us(50);

        // take E low
        *sContolPort = (*sContolPort & 0xf8) | 0x01;

        _delay_us(50); /* Delay REQUIRED */
    }

```

```c
    _delay_ms(5);
}

void LCD_createCharacter(LcdCharacterAddress_t address, LcdCustomCharacter_t custom)
{
    // make sure that the character is correct and valid
    LcdCustomCharacter_t safety = {
        0x40 | (custom[0] & 0x1f),
        0x40 | (custom[1] & 0x1f),
        0x40 | (custom[2] & 0x1f),
        0x40 | (custom[3] & 0x1f),
        0x40 | (custom[4] & 0x1f),
        0x40 | (custom[5] & 0x1f),
        0x40 | (custom[6] & 0x1f),
        0x40 | (custom[7] & 0x1f),
    };

    // set the address of the cgram (must be between 64 - 127)
    LCD_sendInstruction(address);

    // send the sterile character
    LCD_sendString((char const * const)safety);
}

char LCD_getCharacter(LcdCharacterAddress_t address)
{
    return (address - 64) / 8;
}
```

```c
/*
 * FileName: Serial.h
 * Version: 1
 *
 * Created: 11/2/2022 1:41 PM
 * Author: Ethan Zeronik
 *
 * Operations: serial definition
 */

#ifndef Serial_h_INCLUDED
#define Serial_h_INCLUDED

#if defined(__cplusplus)
extern "C" {
#endif

#pragma message("WARNING: this module defaults to TX0/RX0 for interrupts and serial
communication")

#include <stdint.h>

/* NOTE: Custom Types */
// typing for the stepper motor enum
typedef enum SerialPortSelector_t
{
    // usart 0 selector
    serialUsart0 = 0,
    // usart 1 selector
    serialUsart1 = 1,
    // usart 2 selector
    serialUsart2 = 2,
    // usart 3 selector
    serialUsart3 = 3,
} SerialPortSelector_t;

// typing for the handler function
typedef void (*SerialAsyncGetHandler_t)(char);

/* NOTE: Function prototypes */
// init for the serial sync mode
void SERIAL_uartInit(SerialPortSelector_t port, uint32_t baud);
// init for the serial async mode
void SERIAL_uartInitAsync(SerialPortSelector_t port, uint32_t baud);
// sends the buffer to the desired port
// the string must be null terminated
void SERIAL_uartSend(SerialPortSelector_t port, char const * const pTransmitString);
// sends the buffer to the desired port
void SERIAL_uartSendFixed(SerialPortSelector_t port, char const * const pTransmitString,
uint16_t length);
// get a char from the serial buffer
char SERIAL_uartGetSync(SerialPortSelector_t port);
// set the async handler
// will run on every character to the uart buffer
void SERIAL_uartAsyncGetHandler(SerialPortSelector_t port, SerialAsyncGetHandler_t cb);

#if defined(__cplusplus)
} /* extern "C" */
#endif

#endif // Serial_h_INCLUDED
```

```c
/*
 * FileName: Delay.c
 * Version: 1
 *
 * Created: 11/2/2022 1:41 PM
 * Author: Ethan Zeronik
 *
 * Operations: serial implementation
 */

/* NOTE: Includes */
#include "Serial.h"

#if !defined(F_CPU)
    #define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <avr/interrupt.h>

/* NOTE: Local declarations */
// a helper to poll for the given registers then set the given value
// register and mask are related
// set will be set to the value at read once polling finishes
void pollThenSetHelper(uint8_t volatile * const pRegister, uint8_t mask, uint8_t volatile * const pSet, uint8_t volatile * const pRead);
// a helper to remove redundant logic for sending data
void sendCharHelper(SerialPortSelector_t port, char const * const value);

/* NOTE: Global Variables */
// the handlers for each main uart channel
static SerialAsyncGetHandler_t interruptCallback[4];

/* NOTE: Local function implementations */
void SERIAL_uartInit(SerialPortSelector_t port, uint32_t baud)
{
    uint16_t baudCalc = ((F_CPU / baud) / 16) - 1;

    switch(port)
    {
        default:
        case serialUsart0:
        {
            UCSR0A = 0x00;
            // enable UART TX and RX with interrupt flag
            UCSR0B = 0x18;
            // set the UART for N, 8, 1
            UCSR0C = 0x06;
            // set BAUD Rate for 16MHz clock
            UBRR0L = baudCalc;
            UBRR0H = (baudCalc >> 8) & 0x0f;
        }
        break;
        case serialUsart1:
        {
            UCSR1A = 0x00;
            UCSR1B = 0x18;
            UCSR1C = 0x06;
            UBRR1L = baudCalc;
            UBRR1H = (baudCalc >> 8) & 0x0f;
        }
        break;
        case serialUsart2:
        {
            UCSR2A = 0x00;
            UCSR2B = 0x18;
            UCSR2C = 0x06;
            UBRR2L = baudCalc;
            UBRR2H = (baudCalc >> 8) & 0x0f;
        }
        break;
        case serialUsart3:
        {
            UCSR3A = 0x00;
            UCSR3B = 0x18;
            UCSR3C = 0x06;
            UBRR3L = baudCalc;
            UBRR3H = (baudCalc >> 8) & 0x0f;
        }
        break;
    }
}

void SERIAL_uartInitAsync(SerialPortSelector_t port, uint32_t baud)
{
    SERIAL_uartInit(port, baud);

    // turn on the rx interrupt
    switch(port)
    {
        default:
        case serialUsart0:
            UCSR0B = 0x98;
            break;
        case serialUsart1:
            UCSR1B = 0x98;
            break;
        case serialUsart2:
            UCSR2B = 0x98;
            break;
        case serialUsart3:
            UCSR3B = 0x98;
            break;
    }
}

void SERIAL_uartSend(SerialPortSelector_t port, char const * const pTransmitString)
{
    char const * pWorker = (char const *)pTransmitString;

    // while we are not at the end of the string
    while(*pWorker != '\n')
    {
        // wait for uart tx to be ready then send out uart
```

```c
        sendCharHelper(port, pWorker);

        pWorker++;
    }
}

void SERIAL_uartSendFixed(SerialPortSelector_t port, char const * const pTransmitString,
uint16_t length)
{
    for(uint16_t i = 0; i < length; i++)
    {
        // wait for uart tx to be ready then send out uart
        sendCharHelper(port, pTransmitString + i);
    }
}

char SERIAL_uartGetSync(SerialPortSelector_t port)
{
    char ch;

    // wait for uart rx to be ready and save to the char
    switch(port)
    {
        default:
        case serialUsart0:
            pollThenSetHelper(&UCSR0A, (1 << RXC0), (uint8_t volatile * const)&ch, &UDR0);
            break;
        case serialUsart1:
            pollThenSetHelper(&UCSR1A, (1 << RXC1), (uint8_t volatile * const)&ch, &UDR1);
            break;
        case serialUsart2:
            pollThenSetHelper(&UCSR2A, (1 << RXC2), (uint8_t volatile * const)&ch, &UDR2);
            break;
        case serialUsart3:
            pollThenSetHelper(&UCSR3A, (1 << RXC3), (uint8_t volatile * const)&ch, &UDR3);
            break;
    }

    return ch;
}

void SERIAL_uartAsyncGetHandler(SerialPortSelector_t const port, SerialAsyncGetHandler_t cb)
{
    // set the interal callback pointer to the one we were given
    interruptCallback[port] = cb;
}

/* NOTE: Local function implementations */
void pollThenSetHelper(uint8_t volatile * const pRegister, uint8_t mask, uint8_t volatile *
const pSet, uint8_t volatile * const pRead)
{
    // wait for the register
    while((*pRegister & mask) == 0)
    {
    }

    // save to the pSet
    *pSet = *pRead;
```

```c
}

void sendCharHelper(SerialPortSelector_t port, char const * const value)
{
    switch(port)
    {
        default:
        case serialUsart0:
            pollThenSetHelper(&UCSR0A, (1 << UDRE0), &UDR0, (uint8_t volatile * const)value);
            break;
        case serialUsart1:
            pollThenSetHelper(&UCSR1A, (1 << UDRE1), &UDR1, (uint8_t volatile * const)value);
            break;
        case serialUsart2:
            pollThenSetHelper(&UCSR2A, (1 << UDRE2), &UDR2, (uint8_t volatile * const)value);
            break;
        case serialUsart3:
            pollThenSetHelper(&UCSR3A, (1 << UDRE3), &UDR3, (uint8_t volatile * const)value);
            break;
    }
}

ISR(USART0_RX_vect)
{
    interruptCallback[serialUsart0](UDR0);
}

ISR(USART1_RX_vect)
{
    interruptCallback[serialUsart1](UDR1);
}

ISR(USART2_RX_vect)
{
    interruptCallback[serialUsart2](UDR2);
}

ISR(USART3_RX_vect)
{
    interruptCallback[serialUsart3](UDR3);
}
```