Lab 9: AVR Programming with C and Atmel Studio 7.

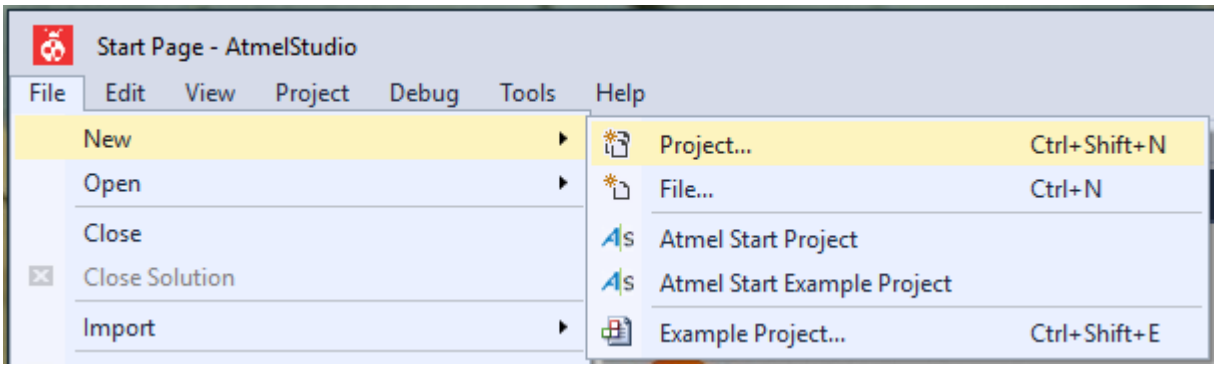
At the end of the lab, submit main.c, which contains the solution to the exercise at the end.

Either at the beginning or end of this lab, your instructor may leave the room so that you can complete the lab evaluations. The evaluations can be completed at https://evals.csc.uvic.ca.

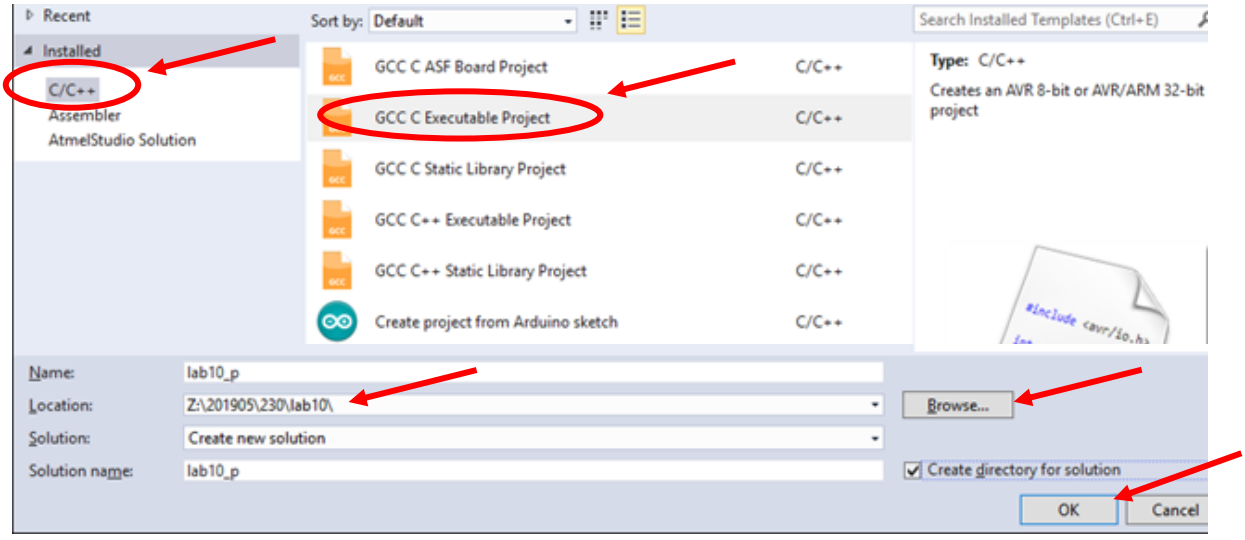
Although we have spent most of the semester writing assembly code for the ATmega2560, the majority of AVR programming in practice is done in C. To make this possible, a fully-featured library of C bindings is available for the ATmega2560, which allows all of the features of the board (Ports, Timers, the LCD screen, etc.) to be used with C code.

I. Creating a C project in Atmel Studio 7.

1. Start the Atmel Studio 7.0 and click on "File --> Project…" at the top left corner.



2. Select the "GCC C Executable Project" and click the "Browse…" button to choose a location (H drive); choose a name of your project and click the "OK" button, as shown below.



3. The rest of the steps are the same as in the previous labs. When the project is created, it will contain an empty .c file for your C code.

After creating the C project, download **CSC230.h** file and save it in the same directory as the main.c file. Then, add it to your project and include it in your main.c program by using the C include directive. We will do this for all C programs that we write in CSC 230. The resulting main.c should look like the following:

```
1  /*
2   * lab9.c
3   *
4   * Created: 2019-11-11 1:01:03 PM
5   * Author : Your Name
6   */
7
8  #include <avr/io.h>
9  #include "CSC230.h"
10
11
12 int main(void)
13 {
14     /* Replace with your application code */
15     while (1)
16     {
17     }
18
19 }
20
```

Note that the basic program skeleton is created for you, which includes the main() function – this is the program entry point, which is why we no longer specify the entry file, as we did with Assembly programs. This skeleton also includes the infinite loop in the form of while(1), which is equivalent to the Assembly "done: JMP done" at the end of the program. One more point of interest is the "#include <avr/io.h>" line, which is similar to the m2560def.inc file – it includes all of the definitions for our special purpose registers and memory addresses, like DDRL and PORTL.

II. Interrupts.

In the past labs, we achieved the effect of a blinking LED light in two different ways. One way was to use a busy-loop delay, to explore how this can be done with C download and review the **lab9_LED_blink_delay.c** file. Note that the delay is achieved using the _delay_ms() function, which is provided in the shared utility library <util/delay.h>. For your convenience, this library (along with a few others) is included from the **CSC230.h** file.

As evident by this example, the busy-loop delay allows us to run things periodically and to implement some timing into our programs. However, a much more reliable way to achieve timing on AVR is by using the built-in timers/counters together with interrupts. For an example on how to do that with C, download and examine the **lab9_LED_blink_isr.c** file. A complete list of the interrupt handler names and the corresponding interrupt vector locations is available online (https://www.microchip.com/webdoc/AVRLibcReferenceManual/group__avr__interrupts.html) and provided here for convenience:

Vector Number	Handler Name	Vector Number	Handler Name
2	INT0_vect	30	ADC_vect
3	INT1_vect	31	EE_READY_vect
4	INT2_vect	32	TIMER3_CAPT_vect
5	INT3_vect	33	TIMER3_COMPA_vect
6	INT4_vect	34	TIMER3_COMPB_vect
7	INT5_vect	35	TIMER3_COMPC_vect
8	INT6_vect	36	TIMER3_OVF_vect
9	INT7_vect	37	USART1_RX_vect
10	PCINT0_vect	38	USART1_UDRE_vect
11	PCINT1_vect	39	USART1_TX_vect
12	PCINT2_vect	40	TWI_vect
13	WDT_vect	41	SPM_READY_vect
14	TIMER2_COMPA_vect	42	TIMER4_CAPT_vect
15	TIMER2_COMPB_vect	43	TIMER4_COMPA_vect
16	TIMER2_OVF_vect	44	TIMER4_COMPB_vect
17	TIMER1_CAPT_vect	45	TIMER4_COMPC_vect
18	TIMER1_COMPA_vect	46	TIMER4_OVF_vect
19	TIMER1_COMPB_vect	47	TIMER5_CAPT_vect
20	TIMER1_COMPC_vect	48	TIMER5_COMPA_vect
21	TIMER1_OVF_vect	49	TIMER5_COMPB_vect
22	TIMER0_COMPA_vect	50	TIMER5_COMPC_vect
23	TIMER0_COMPB_vect	51	TIMER5_OVF_vect
24	TIMER0_OVF_vect	52	USART2_RX_vect
25	SPI_STC_vect	53	USART2_UDRE_vect
26	USART0_RX_vect	54	USART2_TX_vect
27	USART0_UDRE_vect	55	USART3_RX_vect
28	USART0_TX_vect	56	USART3_UDRE_vect
29	ANALOG_COMP_vect	57	USART3_TX_vect

III. The LCD screen.

To work with the LCD screen, in addition to including the CSC230.h file in your program, we need to include the LCD subroutines (functions) in our project. The necessary code is provided in the CSC230_LCD.c file, which needs to be located in the same directory as the main.c file and added to the project via the Solution Explorer in Atmel Studio 7. You can do so by right-clicking the project name in the Solution Explorer and choose “Add -> Existing Item…”, then select the CSC230_LCD.c file and click on the “Add” button.

For an example on how to work with the LCD screen using C, download and examine the **lab9_LCD_pattern_isr.c** file. This program uses the timer to periodically change the string that is being displayed on the LCD screen. The string is chosen from an array of strings using a global index variable, which is changed from a timer-driven ISR. Notice that because it is updated from an ISR and used elsewhere in the program, we use the keyword **volatile** at the front of the index variable to ensure that the corresponding registers are always updated from memory before being used.

When displaying things on LCD, it is sometimes useful to format the output or to convert variables into string format. This can be done using the C sprintf() function, its complete documentation is available online, here, http://www.cplusplus.com/reference/cstdio/sprintf/, and here, https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm. For an example on how to use it with the LCD screen on AVR, download and examine the **lab9_LCD_sprintf.c** file.

IV. ADC and buttons.

To get input from the buttons located on the LCD Shield, as before, we use the Analog to Digital Converter (ADC). The procedure is the same as what we have done in Assembly. First, we configure the ADC, and when the time comes to check which button is being pressed, we instruct the ADC to initiate conversion and wait for it to complete before retrieving the result. The example provided in **lab9_ADC_show_result.c** periodically polls the ADC and displays the result on the LCD screen represented in hexadecimal format. Pressing different buttons will produce different values on the LCD. Note that the conversion from the short variable type to hexadecimal format could also be achieved using the sprintf() function; however, this example achieves it using the short_to_hex() function, which shows how to use a look-up table.

V. Exercise.

Download all of the C examples provided with this lab, build them and upload the resulting .hex files to the AVR board. Familiarize yourself with the code.

Create a new project and write a program that periodically polls the ADC using a timer-driven interrupt twenty times per second and displays which button is being pressed using the words “UP”, “DOWN”, “LEFT”, and “RIGHT” on the top line of the LCD screen. At the same time, using another timer-driven interrupt, count how many seconds have passed since the program started and display the count on the second line of the LCD screen. Remember to **update the LCD only from the main loop, and not from the ISRs**. The ISRs should simply update their corresponding global variables which the main loop would display. Don’t forget to use the volatile keyword properly.

Either at the beginning or end of this lab, your instructor may leave the room so that you can complete the lab evaluations. The evaluations can be completed at https://evals.csc.uvic.ca.

At the end of the lab, submit main.c, which contains the solution to the exercise at the end.