Lab 11

Implementing a 4-Function Calculator using Interrupts

Due: Before the start of your lab section on November 30, December 5, or December 6*

This is a team-effort project. You may discuss concepts and syntax with other students, but you may discuss solutions only with your assigned partner(s), the professor, and the TAs. Sharing code with or copying code from a student who is not on your team, or from the internet, is prohibited.

In this assignment, you will write code for your Arduino Nano-based class hardware kit that will use interrupts from external devices and from a timer to drive some of the logic for a four-function calculator.

Figure 1: Interrupts. Image by 20th Century Fox Television

The instructions are written assuming you will edit the code in the Arduino IDE and run it on your Arduino Nano-based class hardware kit, constructed according to the pre-lab instructions. If you wish, you may edit the code in a different environment; however, our ability to provide support for problems with other IDEs is limited.

Learning Objectives

After successful completion of this assignment, students will be able to:

*See Piazza for the due dates of teams with students from different lab sections.

- Work collaboratively on a hardware/software project
- Use tables from a datasheet to determine the bit vectors needed to configure I/O devices
- Configure a hardware timer to generate interrupts
- Register an interrupt service routine for an interrupt vector
- Register an interrupt service routine using a higher abstraction
- Use interrupt-driven I/O to design and implement a simple embedded system

Continuing Forward

This penultimate lab assignment does not contribute to the final lab assignment. By integrating elements of what you learned in this course to design a small embedded system, you will show how much progress you have made this semester.

During Lab Time

During your lab period, the TAs will demonstrate how to use tables from a datasheet to construct bit vectors. Coordinate with your group partner(s) to decide on your working arrangements. Unless you're only going to work on the assignment when you're together, you may want to set up a *private* Git repository that is shared with your partner(s). With your partner(s), think through your system's design and begin implementing it. The TAs will be available for questions.

No Spaghetti Code Allowed

In the interest of keeping your code readable, you may *not* use any **`goto`** statements, nor may you use any **`continue`** statements, nor may you use any **`break`** statements to exit from a loop, nor may you have any functions **`return`** from within a loop.

1 Scenario

"I have various teams working on different projects around here to improve security," Archie reminds you. He glances toward the Zoo's labs, where there's now a guy who looks like the actor who portrayed the fictional actor who portrayed the Norse god Odin, trying to avoid children while wistfully talking about raising rabbits in Montana. You briefly wonder why there are children someplace where there are also carnivorous megafauna, and then you remember that you work at a petting zoo. "What I need your team to do," Archie continues, "is make a four-function calculator so that we can quickly and easily determine whether we have the correct number of specimens, or if any are missing."

2 Lab Overview

Please familiarize yourself with the entire assignment before beginning. Section 3 has the functional specification of the system you will develop. Obviously, detecting inputs is necessary before being able to implement any other requirements, and section 4 describes detecting inputs using interrupts. While you're in an interrupt frame of mind, you might go ahead and implement the display-timeout requirement; section 5 describes configuring a timer to do so. Alternately, you could implement the rest of the calculator's features before tackling the display-timeout requirement.

Note that Figure 2 depicts the "normal" behavior for the system as a state diagram. You might find that designing your software as a state machine may make it easier to implement your system.

2.1 Interrupt-Driven Input/Output – A Different but Familiar Programming Paradigm

In PollingLab you used polling to determine when to respond to a pushbutton or a key on the keypad being pressed. In this lab, you will rely on interrupts to let your code know when it needs to respond to a pushbutton or a key on the keypad being pressed, and you will rely on interrupts to let your code know when a certain amount of time has passed.

Most of the code you wrote in the earlier lab assignments for this courses used imperative programming: you, the programmer, had full control over changes to the program state. In PollingLab, you stretched this model to something resembling shared-memory concurrent programming: you still had full control over changes to the program state of your program's flow of control, but your program periodically read variables (memory-mapped input registers) that could be changed by another flow of control (the physical world).

In this lab assignment, you will write code that reacts to things happening in the physical world; your code program state will not change except in reaction to interrupts. Conceptually, this isn't very different from event-driven programming that you learned in CSCE 156, RAIK 184H, or SOFT 161. While configuring the hardware timer is a little more complex than configuring a GUI framework's timer, handling a timer interrupt is very much like writing an **`onTimeout()`** event handler. Similarly, handling a change in a pushbutton's position is very much like writing an **`onMouseClick`** event handler. Your code is not focused on *when* the button is pressed or released, nor even on *detecting* that a button has been pressed or released; it is focused solely on *what should happen* when the button is pressed or released.

2.2 Constraints

You may re-use your code that uses memory-mapped I/O registers, or you can use functions provided by the CowPi library to read from and write to pins.

You may *not* poll the matrix keypad nor the pushbuttons to determine if they have been pressed. You must use interrupts to determine if a key or button has been pressed. Once a press has been detected, you may scan the matrix keypad or read the pushbuttons to determine which key or button has been pressed.

You may poll the left switch to determine if its position has changed; however, the specification has been written such that your code should only need to occasionally check the switch's position rather than polling it for changes.

While it is possible to configure the I^{2}C hardware to generate an interrupt after the content of the TWI Data Register is transmitted, you may use your **`send_halfbyte()`** function that polls the `TWINT` bit, or you may use the Cow Pi library's default implementation.

2.2.1 Constraints on the Arduino core

You may use **`attachInterrupt()`** and **`digitalPinToInterrupt()`** to register interrupt handlers. You may use the **`delayMicroseconds()`** function to introduce a 1–2μs delays if necessary. You may (but are not required to) use **`Serial.print()`** and **`Serial.println()`** instead of **`printf()`** if you wish.

While you may use **`millis()`** for debouncing, you may *not* use **`millis()`** nor **`micros()`** to implement the display timeout. You must use an interrupt from the ATmega328's Timer1 or Timer2 as part of implementing the display timeout.

You may not use any other libraries, functions, macros, types, or constants from the Arduino core.[1]

2.2.2 Constraints on AVR-libc

You may use any AVR-specific functions, macros, types, or constants of avr-libc.[2]

2.2.3 Constraints on the CowPi library

You may use any functions that are described in Section 2 of the Cow Pi datasheet, and you may use any data structures provided by the CowPi library.

2.2.4 Constraints on other libraries

You may not use any libraries beyond those explicitly identified here.

[1] https://www.arduino.cc/reference/en/

[2] https://www.nongnu.org/avr-libc/user-manual/index.html

2.2.5 Constraints on code from earlier assignments

You may use any code written by you or your partner(s), including code that you or your partner(s) wrote for this course.

3 Four-Function Calculator Specification

Integer Calculator

1. The calculator shall be an infix[3] decimal calculator capable of performing addition, subtraction, multiplication, and division.
   - Division shall be integer division; *i.e.*, the fractional portion of quotients values shall be truncated.
2. A decimal point shall not be displayed. Digit-separating commas are optional, neither required nor prohibited.
3. Each arithmetic operation shall use two operands, referred in this specification as *operand1* and *operand2*.
   - Because no further history of operands is maintained, the algebraic order of operations is not preserved.
   - The value of *operand1* can only be changed as the result of a calculation or by clearing its value.
   - The value of *operand2* is "built" through keypresses on the matrix keypad.
   - *operand1* and *operand2* are defined in the problem domain. Your solution space might not have direct corollaries (for example, in the problem domain, *operand2* can be undefined, which is not a characteristic of any number types in C).
4. The **display module** shall display the decimal representation of *operand1*'s value on the top row, right-justified, except when an error condition is present.
5. If *operand2* has a defined value (is being built) then the **display module** shall display the decimal representation of *operand2*'s value on the bottom row, right-justified, except when an error condition is present. If *operand2* does not have a defined value then the visual space for *operand2*'s value shall be blank.
6. If an operation is specified, then the character for that operation ($+, -, \times, \div$) shall be displayed in the lower-left corner of the **display module**. If no operation is specified, then the visual space for the operation's character shall be blank.

[3] https://en.wikipedia.org/wiki/Calculator_input_methods#Infix_notation

7. Initially, *operand1* shall hold the value 0, *operand2* shall have no defined value, and no arithmetic operation shall be specified.

8. When building *operand2*:

   (a) Except as otherwise specified in requirement 18, whenever the user presses a numeral button on the **matrix keypad**, the corresponding digit shall be displayed in the least-significant position of the value on the **display module**, and any digits already displayed shall increase in significance by one order of magnitude. For example, if `234` is displayed and the user presses `5` then `2345` shall be displayed.

   (b) Except as otherwise specified in requirement 18, whenever the user presses the **left pushbutton**, the value being built shall be negated, and the visual representation of the number shall reflect this.

   - If *operand2* has a defined value (is being built) then *operand2* shall be negated.
   - If *operand2* has no defined value (*operand2* is not displayed, but *operand1* is) then *operand1* shall be negated.

   (c) In no case shall the calculator allow the user to input a value requiring more than nine (9) digits, (the negative sign, if present, is not considered to be one of these digits). If the user attempts to enter a value that requires more digits, then the value shall remain unchanged.

9. If a negative value is displayed, the negative sign shall be displayed immediately to the left of the most-significant digit being displayed. For example, `-456` is correctly displayed, but `-    456` is not correctly displayed.

10. A positive sign shall not be displayed as part of a positive value.

11. Leading 0s shall not be displayed. For example, `782` is correctly displayed, but `00000782` is not correctly displayed. However, when the value to be displayed is 0, then `0` shall be displayed.

12. The `A` button indicates addition. The `B` button indicates subtraction. The `C` button indicates multiplication. The `D` button indicates division. The `#` button directs that the calculation shall be executed without specifying another arithmetic operation.

13. Except as otherwise specified in requirement 18, when the user presses the `#` button, directing that the calculation is to be executed, the previously-specified operation shall be performed, the resulting value shall become *operand1* (and shall be displayed on the top row), and there shall no longer be a valid *operand2* (and thus cannot be displayed) nor a specified operation (and thus cannot be displayed). If there was no previously-specified operation, then *operand2* shall become *operand1*, and there shall no longer be a valid *operand2*.

14. Except as otherwise specified in requirement 18, when the user presses an operation button (`A-D`), the previously-specified operation shall be performed, the resulting value shall become *operand1* (and shall be displayed on the top row), there shall no longer be a valid *operand2* (and thus cannot be displayed), and the next operation to be performed shall be that which corresponds to the button pressed. If there was no previously-specified operation, then *operand2* shall become *operand1*, *operand2* shall no longer have a defined value, and the next operation to be performed shall be that which corresponds to the button pressed. In accordance with requirement 6, the next operation shall be displayed.

    - If the resulting value (*operand1*) is too great to be displayed, the **display module** shall display `Error` .
    - If *operand2* is 0 and the operation is division, the **display module** shall display `Error` .
    - If *operand2* is undefined, then *operand1* shall remain unchanged, the previously-specified operation (if any) shall be replaced (it cannot be performed due to an absence of *operand2*), and the next operation to be performed shall be that which corresponds to the button pressed

15. The calculator shall not support scientific notation, neither for operand entry nor for displaying a result.

16. Except as otherwise specified in requirement 18, whenever the user presses the **right pushbutton**:

    (a) If *operand2* is being built, then *operand2* shall be reset to no defined value, the next operation (if any) shall be cleared, and *operand1* shall be displayed.

    (b) If *operand2* is not being built, then *operand1* shall be reset to 0 (and 0 shall be displayed), and the next operation (if any) shall be cleared.

17. Display timeout: If no button or key has been pressed for a designated amount of time, the **display module**'s backlight shall deluminate; however, *operand1*, *operand2* being built (if any), and the next operation (if any) shall be retained. When the **left switch** is in the left position, the designated time is exactly 20 seconds; when the **left switch** is in the right position, the designated time is exactly 7.5 seconds.

    - Do not place the microcontroller in Idle mode, nor any other Sleep mode.

18. If the display has timed out, then pressing any key or button will cause the backlight to illuminate. The key/button press that takes the display out of its timed-out mode shall not otherwise affect the system's state; specifically, it shall not contribute to *operand2*'s value, it shall not cause an operation to be executed, it shall not specify an operation to be performed, nor shall it clear any operands or specified operations.

Figure 2 depicts a problem-space state diagram of the 4-function integer calculator, not including too-great values, and not including the display's time-out and resume-display behavior.

4 Detecting Inputs

Read Section 5 of the Cow Pi datasheet for an overview of handling interrupts for your Arduino Nano.

To detect keypresses and buttonpresses, you will take advantage of having the NAND of the pushbuttons and the NAND of the keypad columns being input to Arduino pins D2 and D3.

Add code to **`setup()`** to register **`handle_buttonpress()`** as your buttonpress handler and **`handle_keypress()`** as your keypress handler. Use the **`attachInterrupt()`** function described in Section 5.3 of the datasheet.

Initially, you don't have much for these functions to do – you will add code to take appropriate action as you implement your system. For now, place code in your **`handle_buttonpress()`** to illuminate the left LED when the left button is pressed and to illuminate the right LED when the right button is pressed. You will later delete this code; this is simply to satisfy yourself that you are detecting the buttons being pressed and can determine which button has been pressed. Similarly, add code to **`handle_keypress()`** to display the presed key on the display module. You will alter delete this code; this is simply to satsify yourself that you are detecting keys being pressed and can determine which key has been pressed.

You will *not* use interrupts to detect the switches changing position – the specification has been written such that no action needs to be taken at the moment that a switch is toggled; you only need to check the switch positions after something else has triggered some code.

NOTE Any global variables used by your interrupt handlers should be declared as **`volatile`**.

Don't forget to reset the 7.5-second / 20-second countdown so that your code doesn't blank the display too soon.

Because we did not provide a hardware solution to switch bounce, you can expect the pin input to both rise and fall a few times when a button or key is pressed and again when it is released – but there is no guarantee that bouncing will occur. For this reason we recommend:

- The software debouncing will look similar to what you used in PollingLab except that it only needs to be a few milliseconds instead of 500. Since we are not polling the buttons and keypad to detect presses, we do not need to worry about the button or key being held for several dozen milliseconds being interpretted as multiple presses.
  - I very strongly advise against using **`delay()`** for software debouncing:
  - As described in the PollingLab assignment sheet, **`delay()`** will leave your system unresponsive to anything except interrupts.

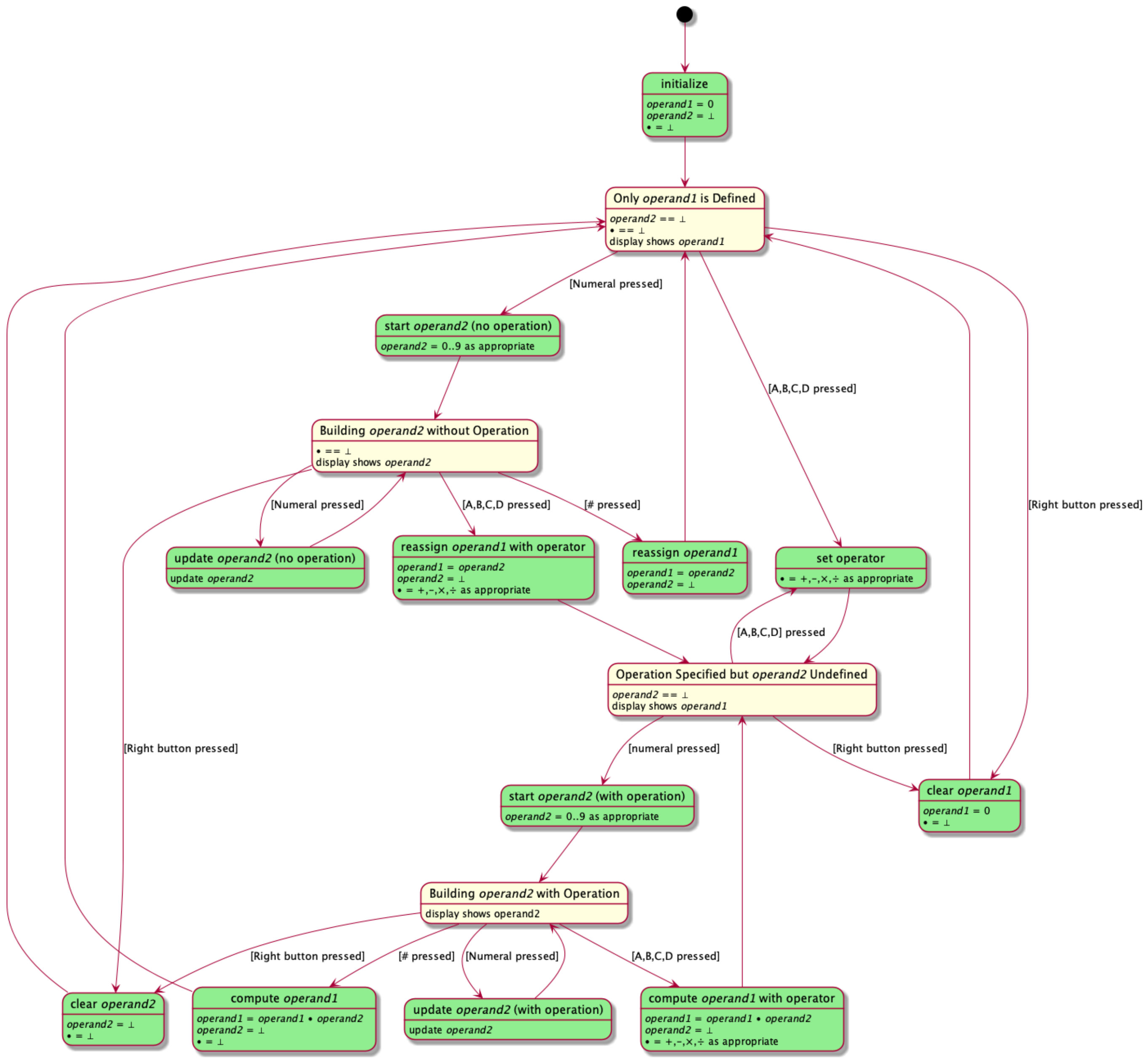

Figure 2: State diagram depicting "normal" behavior for integer calculator. Yellow states depict invariant conditions; green states depict updates to the problem-domain variables *operand1*, *operand2*, and • (operator). Diagram by Bohn

  - Including **`delay()`** calls in an interrupt handler is particularly ill-advised since interrupts are disabled by default in the interrupt handler. The **`delay()`** function will never exit, blocking forever, if interrupts are disabled.

If you arrive at a different solution that works, you may use your solution.

5 Implementing Timeout

Read Section 6 of the Cow Pi datasheet to familiarize yourself with timer interrupts on the Arduino Nano. Note that Sections 6.3–6.5 are generally duplicates of each other except for their tables and a warning at the start of Section 6.3; for now you can read Sections 6.1, 6.2, 6.3, and 6.6. Later, after you determine which timer you will use, you can re-visit Section 6.4 or 6.5 to consult its tables.

You must use timer interrupts from either Timer1 or Timer2 as part of your implementation of the display timeout. To implement the timeout, you will need to determine an appropriate timer period, from which you will determine the necessary comparison value and prescaler. NOTE: there is no combination of interrupt period and comparison value to produce an interrupt period of 20 seconds, or even 7.5 seconds; you will need to use a shorter interrupt period with some other logic to determine when it is time to dim the displpay module.

Use the equation in the datasheet's Section 6.2 to iterate through possible comparison values and prescalers until you arrive at a prescaler that is available for one of the timers and a comparison value that fits in the available counter bits for that timer. When you have done so:

- Uncomment the timer declaration on line 22 or 23, as appropriate, in the starter code
- Use the address offset listed in Section 6.4 or 6.5 to assign the approriate address to the `timer` pointer in **`setup()`** on line 31
- Use the tables for your chosen timer to select the WGM bits for "Clear on Timer Compare" mode and the CS bits for your prescaler; use these bits to generate the bit vector that you will assign to `timer->control` in **`setup()`** on line 32
- Subtract 1 from the comparison value and assign that to `timer->compareA` in **`setup()`** on line 33
- Use the address offset listed in Section 6.6 to assign the appropriate address to the `timer_interrupt_masks` pointer in **`setup()`** on line 34
- Create an ISR for the `TIMER`*`n`*`_COMPA_vect` interrupt, where `n` is the timer number, using the **`ISR()`** macro as described in Section 5.2 of the datasheet; for now, write very simple code in the ISR, such as turning an LED on or off, or displaying/clearing "timeout!" – you will add useful code later

- Index the `timer_interrupt_masks` array with the timer number and enable the `TIMERn_COMPA_vect` interrupt, where `n` is the timer number, in **`setup()`** on line 35

Now you can add code to your ISR to implement your display-timeout logic.

NOTE Any global variables used by your ISR should be declared as **`volatile`**.

NOTE If you ever need to “reset” a timer’s count back to 0, you can simply write 0 to the `timer->counter` field.

6 Turn-in and Grading

When you have completed this assignment, upload *CalculatorLab.ino* to Canvas.

This assignment is worth 50 points.

Rubric:

_____ **+2** The source code is well-organized and is readable.

Pushbutton Interrupts

_____ **+4** Pushbutton presses are detected with an external interrupt.

_____ **+2** The pushbuttons’ interrupt handler determines which button was pressed.

_____ **+1** Pushbutton interrupt handler is not excessively long.

Matrix Keypad Interrupts

_____ **+4** Matrix keypad presses are detected with an external interrupt.

_____ **+2** The keypad’s interrupt handler determines which key was pressed.

_____ **+1** Keypad interrupt handler is not excessively long.

Timer Interrupts

_____ **+2** Timer interrupts for Timer1 or Timer2 are enabled and handled.

_____ **+4** The timer interrupts configured such that, when combined with other logic, the software is able to determine when exactly 7.5 seconds or exactly 20 seconds have passed.

_____ **+1** Timer ISR is not excessively long.

Timeout Logic

_____ **+2** The system is able to determine when exactly 7.5 seconds have passed since a button or key was pressed.

_____ **+2** The system is able to determine when exactly 20 seconds have passed since a button or key was pressed.

Normal Calculator Functionality

_____ **+1** Operand is built in accordance with requirements 8, 9, 10, and 11.

_____ **+1** After nine digits, subsequent numeral presses are ignored (except for resetting the display-blanking countdown).

_____ **+2** Addition performs correctly.

_____ **+2** Subtraction performs correctly.

_____ **+2** Multiplication performs correctly.

_____ **+2** Division performs correctly.

_____ **+1** The result of the previous arithmetic operation can be used as the first operand for the next arithmentic operation.

_____ **+1** The "clear" button clears the operand being built and any specified operation.

_____ **+1** If no operand is currently being built (the previous arithmetic's result is displayed) then the "clear" button clears the previous arithemtic's result and any specified operation.

_____ **+1** The result of the previous operation is displayed on the top row (if there was no previous operation, then `0`).

_____ **+1** The specified operation, if any, is displayed in the lower-left corner.

_____ **+1** The operand being built, if any, is displayed on the bottom row.

Other Calculator Functionality

_____ **+1** Too-great results cause `Error` to be displayed.

_____ **+2** The display's backlight deluminates after exactly 7.5 seconds or exactly 20 seconds of inactivity, depending on the position of the left switch.

_____ **+2** When the display is dim, pressing any button or key causes the backlight to illuminate. . .

_____ **+2** . . . and has no other effect, allowing operation to resume where the user had left off before letting the display timeout.

Bonus and Penalty

_____ **Bonus +2** Get assignment checked-off by TA or professor during office hours before it is due. (Cannot get both bonuses.)

_____ **Bonus +1** Get assignment checked-off by TA at *start* of your scheduled lab immediately after it is due. (Cannot get both bonuses.)

_____ **-7** The pushbutton interrupt handler's implementation violates one or more constraints identified in Section 2.2.

_____ **-7** The keypad interrupt handler's implementation violates one or more constraints identified in Section 2.2.

_____ **-7** The timer ISR's implementation violates one or more constraints identified in Section 2.2.

_____ **-4** The timeout logic violates one or more constraints identified in Section 2.2.

_____ **-23** The remainder of the calculator's implementation violates one or more constraints identified in Section 2.2.

_____ **-1** for each **`goto`** statement, **`continue`** statement, **`break`** statement used to exit from a loop, or **`return`** statement that occurs within a loop.

Epilogue

After using your calculator to compute how many specimens are still present in the lab, and establish that all specimens are accounted for after Newman's attempted theft. As reports come in of facilities getting secured with Cow Pi-based locks and passages being monitored with Cow Pi-based motion sensors, Archie smiles and tells you that this was a job well done. With all of the excitement neatly wrapped-up and arriving at a satisfactory conclusion, you look forward to a boring career in which there's absolutely no screaming and running for your life.

The End...?

A Lab Checkoff

NOTE: this checklist still needs to be updated to reflect changes for the new display module. An updated rubric will be released soon.

You are not required to have your assignment checked-off by a TA or the professor. If you do not do so, then we will perform a functional check ourselves. In the interest of making grading go faster, we are offering a small bonus if you complete your assignment early and get it checked-off by a TA or the professor during office hours.

Ideally, all team members are present for the check-off; however, only one team member is necessary for the check-off.

( ) Establish that the code you are demonstrating is the code you submitted to to Canvas.

- Download the file into your calculator directory. If necessary, rename it to *calculator.ino.*

Code Examination

1. ( ) Show the TA that your code is well-organized and readable.
   +2 the source code is well-organized and readable
   If you disagree with the TA's assessment of your code quality, you may appeal to the professor.
   In addition to any appeals, the professor will also spot-check a random sample of the submissions and may increase or decrease the code quality scores.
2. ( ) Show in your code that you registered interrupt handlers for the pushbuttons and for the matrix keypad.
3. ( ) Show in your code your interrupt handlers for the pushbuttons and for the matrix keypad.
   +4 pushbutton presses are detected with an external interrupt
   +1 pushbutton interrupt handler is not excessively long
   +4 matrix keypad presses are detected with an external interrupt
   +1 keypad interrupt handler is not excessively long
4. ( ) Show and explain how you determined the comparison value and prescaler for your timer interrupts. Explain why the timer period you selected is appropriate.

$$comparison\_value = 16,000,000 \times interrupt\_period \div prescaler$$

5. ( ) Show in your code that you had configured the timer to use those comparison and prescaler values and that you had configured the correct timer mode.
   +4 timer interrupts are configured such that, when combined with other logic, it is possible to determine when 7.5 seconds or 20 seconds have passed.
6. ( ) Show in your code that you enabled the correct timer.

7. (   ) Show in your code that you created an ISR for the correct timer.
   +2 timer interrupts are enabled and handled
   +1 timer ISR is not excessively long

8. (   ) Upload *CalculatorLab.ino* to your Arduino Nano.

   If you did not implement the calculator, proceed to step 9.
   If you did implement some or all of the calculator, jump to step 12.

Interrupt Handling without Calculator Functionality

9. (   ) Press the left button to show that the system responds to the left button, such as illuminating an LED.

10. (   ) Press the right button to show that the system responds to the right button differently, such as illuminating a different LED.
    +2 pushbutton interrupt handler determines which button was pressed.

11. (   ) Press a few keys on the keypad to show that something unique about each key (such as its face value) is displayed.
    +2 keypad interrupt handler determines which key was pressed.

    If you did not implement the calculator, jump to step 42.

Calculator Functionality

12. (   ) The display shows:
```
              0 
                
```

13. (   ) Place the left switch in the right position.

14. (   ) Enter 123,456. The display shows:
```
              0 
         123456 
```
    +.5 positive numbers
    +1 the operand being built shown on bottom row
    +2 keypad interrupt handler determines which key was pressed.

15. (   ) **If you implemented display timeout:** After a few seconds, he display goes blank.

16. ( ) **If you implemented display timeout:** Press 7. The display resumes, showing what it previously showed:
```
               0
          123456
```
+2 button/key re-illuminates the backlight. . .
+1 . . . without any other changes. . .

17. ( ) Press 7.
```
               0
         1234567
```
+1 . . . allowing user to resume where they left off

18. ( ) Press `A` (addition).
```
         1234567
+
```
+1 the specified operation shown in lower-left corner

19. ( ) Enter 890.
```
         1234567
+            890
```

20. ( ) Press `#` (“=”).
```
         1235457

```
+2 addition works
+1 result shown on top row

21. ( ) Press the right pushbutton (clear).
```
               0

```
+1 clears operand1 when operand2 isn’t being built

22. ( ) Place the left switch in the left position.

23. ( ) Enter -123,456,789, using the left pushbutton for the negative sign.
```
               0
      -123545789
```
+.5 negative numbers
+2 pushbutton interrupt handler determines which key was pressed.

24. ( ) Press 0. The display is unchanged; it still shows:
```
               0
      -123545789
```
+1 maximum of 9 digits entered

25. ( ) Press #.
```
      -123545789

```
26. ( ) Enter 231.
```
      -123545789
             231
```
27. ( ) Press `B` (subtraction) and enter 321.
```
             231
-            321
```
28. ( ) Press #.
```
             -90

```
+2 subtraction works

29. ( ) Press the right pushbutton (clear).
```
               0

```
30. ( ) Press `C` (multiplication) and enter 12.
```
               0
×             12
```
31. ( ) Press #.
```
               0

```
32. ( ) Enter 42.
```
               0
              42
```
33. ( ) Press `D` (division) and enter 6.
```
              42
÷              6
```
34. ( ) Press #.
```
               7

```
+2 division works

35. ( ) Press `C` (multiplication) and enter 73.
```
               7
×             73
```

36. (   ) Press `#`.
```
            511

```
+2 multiplication works
+1 previous result can be used as operand

37. (   ) Press `C` (multiplication) and enter 200.
```
            511
 x          200
```

38. (   ) Press the right pushbutton (clear).

39. (   ) Press `C` (multiplication) and enter 200.
```
            511

```
+1 clears operand2 & operator

40. (   ) Press `C` (multiplication) and enter 2,000,000.
```
            511
        2000000
```

41. (   ) Press `#`.
```
          Error

```
+1 too-great results are an error

Display Timeout

42. (   ) Prepare a stopwatch or another timepiece with a "seconds" display (or a "seconds" hand)

43. (   ) Place the left switch in the right position.

44. (   ) If necessary, re-activate the backlight by pressing a button or key.

45. (   ) Press a button or key on the system at the exact same time as starting the stopwatch (or at the exact moment that the "seconds" display/hand hits *00*).

46. (   ) Exactly 7.5 seconds later, the backlight deluminates.
+2 System can determine when 7.5 seconds have passed
+1 Backlight deluminates after 7.5 seconds of inactivity

47. (   ) Place the left switch in the left position.

48. (   ) Re-activate the backlight by pressing a button or key.

49. (   ) Press a button or key on the system at the exact same time as starting the stopwatch (or at the exact moment that the "seconds" display/hand hits *00*).

50. (   ) Exactly 20 seconds later, the backlight deluminates.
    +2 System can determine when 20 seconds have passed
    +1 Backlight deluminates after 20 seconds of inactivity

This concludes the demonstration of your system's functionality. The TAs will later examine your code for violations of the assignment's constraints and deduct any unearned points accordingly. If your code looks like it is tailored for this checklist, the TAs may re-grade using a different checklist.