Lab 1 Report

Authors: Khoa Tran, Yasin Alissa
ECE/CSE 474, Embedded Systems
University of Washington – Dept. of Electrical andComputer Engineering

Date: 17 January 2021

Table of Contents

1.0 ABSTRACT

This project consists of multiple introductions to C and the Arduino ATMega 2560 microcomputer. The main task was to display "A B C D" for a desired amount of seconds, given by the user, and clearing the display for a desired amount of seconds as well. We tested and experimented on the Arduino microcomputer, which resulted in successful operation of the proposed function.

2.0 INTRODUCTION

There are two main purposes for this project:

1. Introduce the C language through the CodeBlocks C compiler and explore crucial aspects regarding the C language including pointers and multiple file programs, by creating a simple C program and expanding from the original. Additionally, the practice of debugging was implemented in this project
2. Introduce the Arduino ATMega 2560 microcomputer by testing C functions on the microcomputer through Arduino IDE and the C language. Presenting the associated development environment, and the world of embedded systems.

3.0 DESIGN SPECIFICATION

Requested design specifications include six different applications of C, implemented on Codeblock. The six different applications are built upon the previous one as application 1 displays the letters: "A B C D" in the console and flashes them together at a one-second rate. The second application supports user specified delays that allows for the program to take in a user input of seconds and delays the output in the console by the user's given input. The third application builds upon the second application as it creates two different modules that display the letters, clearing them, and delaying them for a certain amount of seconds. This allows for the main function to just call the two separate functions of outputting the data and clearing it. Application 4 then allows for the delay function to be passed into the two functions created in application 3 instead of giving a time delay. Then, application 5 modifies the previous functions by just having one function that takes in the message and the delay function, allowing for simplification of the code. Lastly, application 6 allows for separate files, learning how to implement headers and multiple files.

Besides the project's design specifications in C, troubleshooting and learning the ATMega environment is critical. As a result, the specifications are to troubleshoot and edit the errors

in the two given project files. Lastly, import application 6 into the ATMega 2560, allowing for the user to input delay values on the touchscreen.

4.0 SOFTWARE IMPLEMENTATION

The specifications above were designed in C through Codeblock and Arduino IDE. The implementation of the applications 1 through 6 are based from two different modules that one displays the text output and the other delays the output in order to flash the letters for a certain amount of seconds.

Application 1: In order to perform the task of flashing the message “A B C D” and then clear it for one second, there has to be a separate function from the main method that deals with delaying the outputs of “A B C D” and “\b\b\b\b\b\b\b" for clearing the message. The delay function obtains the start time and sets the current time to become the start time. The function then loops until the difference between the start time and the current time is at least 1 as the current time progresses from the start time. This function can be seen in Figure 1 of the Appendix with the function delay(). As a result, the main method consists of a continuous while loop that prints the “A B C D” message and calls the delay function as well as the same thing for the clearing message.

Application 2: This application has to modify the task of application 1 that parametrizes the delay function described above. As a result, the delay function takes in a parameter for a certain amount of time for the message to be delayed or flashed. The main function then scans for the user input by asking the user how many seconds does the user want to message to be delayed. Afterwards, the message is flashed and then the delay function is called, passing in the scanned input from the user for the required amount of seconds. This is seen in Figure 2 as the delay function is modified.

Application 3: For this application, the purpose was to modify application 2 so there would be two separate delay functions, one for the data and one for the clear message. As seen in Figure 3, f1Data and f2Clear is passed in an unsigned long that is taken from the user scanned input. Then these individual functions print their own message and call the delay function, passing in the unsigned long value, casted as a double. Then the main function just loops until manually stopped the two functions of f1Data and f2Clear.

Application 4: This application modifies the previous application so instead of passing a value through the parameter, the function has a reference pointer as a parameter. To perform this

task, the parameter of f1Data and f2Clear was changed to a pointer, and when f1Data and f2Clear is called in the main function, the address of the scanned user input is passed through. This can be seen in Figure 4.

Application 5: This application condenses the previous application's two functions, f1Data and f2Clear, into a single function with two parameters: the message to be flashed (or cleared) and the amount to delay after the message is printed (passed by reference). This simplifies the code further as evidenced in Figure 5.

Application 6: In this application, the purpose is to separate application 5 into multiple files in order to simplify the code and split into multiple parts. This process allows for ease of debugging and the ability to split the work between multiple people. This process is done by creating a header file for the separate functions of delay and displayAndDelay and having another file that includes the header file that has all the operations of each function. Lastly, the final file is the main file of application 6 as it calls the header files to access the functions needed in the main method. This process can be seen in Figures 6, 7, and 8.

Troubleshooting applications: For project1b-2021.c, the for-loops went from i = 0 to i <= 5, which is 6 iterations. However, our array is only 5 elements long, meaning we iterate past the edge of our array. To fix this, we changed the <= condition in the for-loops to strictly <, assuring we only access data that's a part of our array. For project1c-2021.c, the problem was that it changed the address of our value pointer to be that of the temp pointer. To fix this, we changed the function so that it dereferences the value pointer and assigns its value to the temp value. That way, once we return out of the function's scope, we still have access to the now updated value in the main method. See Figures 9 and 10.

Application 6 on ATMega 2560: Porting application 6 onto the ATMega 2560 was simple. All that was needed was to put all code that runs before the while loop in the setup() method and all code in the loop was put in the loop() method. There were a few differences, however. For one, some variables, like the delay, are declared outside the scope of either of the methods so that both can access them. We also had trouble calling our display function and making it print to the touch screen, as its corresponding source code is in C++ rather than C, making it difficult to port over. Instead, we printed the message onto our touch screen directly and then used our delay function. Evidence of this is in Figure 11.

5.0 TEST PLAN

Test software implementation for the requirements below as it has to be verified in order to have certainty that the software implementation processes successfully.

5.1 Requirements

Application 1-6:

- Operable delay function that uses user input value
- Operable way to process user input
- Handling false type of user input

Troubleshooting:

- Builds and compiles, while performing the tasks described

Application 6 on ATMega 2560:

- Takes user input and performs the tasks given the user's desired delay amount

5.2 Test Coverage

Application 1-6:

- Measures the delay amount by running the delay function after a print statement and seeing how long the delay amount is between another print statement
- See if the scanned user input is received correctly through printing the value of the user input
- Test different input types to make sure the program requires the user to enter a certain type of input value

Troubleshooting:

- The program has to run without any errors and has the correct output

Application 6 on ATMega 2560:

- Run application 6 on ATMega 2560, making sure the user input is handled correctly, as well as displaying the required values and text.

5.3 Test cases

Application 1-6:

- The input to delay has to be a numeric value and the output of delay has to be equal to the amount given. Input of 1 second should delay the message output for 1 second.
- The user has to enter numeric values or else the program has to continuously ask the user until a numeric value is inputted.

Troubleshooting:

- As limits are compilation and correct output, the function has to pass those test coverages, displaying the correct values for a given temporary test case.

Application 6 on ATMega 2560:

- Check for user input functionalities, making sure false type inputs don't crash the program, and the text outputs are displayed correctly on the touchscreen.

6.0 RESULTS

All modules and tasks assignments were finished as it passes all the test cases required to pass. For application 1, the output in the console is "A B C D" and was flashed for 1 second before clearing and repeating the process. This can be seen in Figure 12 as the console outputs the required text values. For application 2, the user is asked for an amount of seconds to delay the console outputs and as seen in Figure 13, the program takes the input and delays the output of the message and clears for the given seconds. For application 3 through 6, the output in the console is the same as it asks the user for a numeric input value and delays the message and clear message by the given seconds. The formatting of the code is the only difference between these applications. As a result, the output remains the same throughout as seen in Figure 14.

6.1 Analysis of Any Resolved Errors

We came across some issues while working on handling invalid user input in Applications 2-6. Our call to scanf() in our first while loop (Figure 2) initially didn't stop the program to wait for input every iteration. This is when we told scanf() to scan for doubles. We decided to scan for a String instead and convert that String to a double (Figure 2), thus solving our issue.

6.2 Analysis of Any Unresolved Errors

While porting our Application 6 code onto the Arduino board, some problems arose. While we were able to call our delay() and displayAndDelay() functions in the program, they were not able to have the behavior we intended. The way we implemented delay() is using the time() method in the time.h library (Figure 7). When using time() in the .ino file, our return value was always 0, thus leaving us in an endless loop. We counteracted this by using the Arduino library's built in delay() function instead, which had the intended effect. Our displayAndDelay() function also had an issue. We wanted to print to the touchscreen, but our function printed to stdout. Even if we tried to edit the function, the touchscreen's library was written in C++ and included classes, something we're unable to use in our C code. Instead, we directly printed onto the touchscreen in our loop() function. While we didn't use our

Application 6 C functions in the final working program, we figured out how to include them into an Arduino project, which was the intended result of the lab.

7.0 QUESTIONS

No questions were given

8.0 CONCLUSION

As the purpose of this project was to introduce C through Codeblock and on the Arduino ATMega 2560 microcomputer, every task had a purpose that leads towards the overall goal and was built upon each other. Application 1 through 6 essentially had the same output; however, since it was modified and transformed from a previous application, a vast topic of programming was involved in each application, allowing us to review the C material extremely well. From pointers to transforming into multiple files, this project allows us to gain a strong understanding of the material in C as we were able to successfully display the correct output for each functionality. Besides reviewing major aspects of C, troubleshooting and debugging was also part of this project, which allows for practice of catching errors throughout the code. Since we were able to successfully perform these tasks, our knowledge of C was immensely refined. Lastly, the introduction to the Arduino ATMega 2560 microcomputer was different and interesting. Not having the prior experience and having to port inputs and outputs to and from the microcomputer is a difficult challenge. However, by introducing basic functions on the Arduino IDE, we were able to understand the basics and implemented our functions by observing the examples. Despite the fact we couldn't implement our own functions for the delay functionality, we were able to implement the overall task. We don't have much suggestions or recommendations as this was a well throughout project to introduce C in CodeBlocks and implemented in the Arduino ATMega 2560 microcomputer. Though some of the instructions could be clearer as we were confused at some points throughout building the project.

9.0 CONTRIBUTIONS

Both of us were able to contribute fairly to this project. We worked together in a call, dividing the work up evenly as we both wrote the programs for the applications and then wrote the lab report together. We talked about how to implement each application and then combined at the end to have one final product.

10.0 APPENDICES

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

// Delays the code continuing to run for 1 second
void delay();

// Flashes the message "A B C D" for 1 second then clears it for 1 second. It
// repeats this behavior indefinitely so the program must be manually stopped
int main() {

    // Sets the message we want to flash
    char* message = "A B C D";
    // Sets the string used to clear the message
    char* clear = "\b\b\b\b\b\b\b";
    // Makes sure the console's buffer prints without a new line needed
    setbuf(stdout, NULL);
    // Loop indefinitely
    while (1) {
        // Print the message then pause for a second
        printf("%s", message);
        delay();
        // Clear the message then pause for a second
        printf("%s", clear);
        delay();
    }

    return EXIT_SUCCESS;
}

void delay() {
    time_t startTime, currTime = 0;
    // Get the time we started at and set the current time to the same time
    time(&startTime);
    currTime = startTime;
    // Loop until the difference in the current time and the start time is
    // at least 1 second
    while (difftime(currTime, startTime) < 1.0) {
        // Get the new current time
        time(&currTime);
    }
}
```

Figure 1 – Application 1

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

// Delays the code continuing to run for the given number of seconds.
// Seconds must be greater than or equal to 0
void delay(double seconds);

// Flashes the message "A B C D" for a number of seconds then clears it for
// another number of seconds.
// The number of seconds is taken from the user input.
// It repeats this behavior indefinitely until the
// program is manually stopped
int main() {
    // Makes sure the console's buffer prints without a new line needed
    setbuf(stdout, NULL);
    // Sets the message to A B C D to flash
    char* message = "A B C D";
    double amount = 0;
    // Ask user for desired seconds of flashing
    printf("How many seconds would you like the message to flash?  ");
    // Only continue when we have valid input
    while (amount == 0) {
        // Scan for user input
        char input[20] = "";
        scanf("%s", input);
        // Convert input to a double
        amount = strtod(input, NULL);
        // If input is invalid, inform the user
        if (amount == 0.0) {
            printf("Invalid input. Try again: ");
        }
    }
    // Sets the string used to clear the message
    char* clear = "\b\b\b\b\b\b\b";
    // Loop indefinitely
    while (1) {
        // Print the message then pause for a second
        printf("%s", message);
        // Delay the given user amount
        delay(amount);
        // Clear the message then pause for a second
        printf("%s", clear);
        // Delay the given user amount
        delay(amount);
    }

    return EXIT_SUCCESS;
}

void delay(double seconds) {
    time_t startTime, currTime = 0;
    // Get the time we started at and set the current time to the same time
    time(&startTime);
    currTime = startTime;
    // Loop until the difference in the current time and the start time is
    // at least the given number of seconds
    while (difftime(currTime, startTime) < seconds) {
        // Get the new current time
        time(&currTime);
    }
}
```

Figure 2 – Application 2

```c
// Seconds must be greater than or equal to 0
void delay(double seconds);

// Prints the message "A B C D" in the console then pauses for the given
// delay in seconds
void flData(double delay1);

// Clears the console then pauses for the given delay in seconds
void f2Clear(double delay2);

// Flashes the message "A B C D" for a number of seconds then clears it for
// another number of seconds. It repeats this behavior indefinitely so the
// program must be manually stopped
int main() {
    // Makes sure the console's buffer prints without a new line needed
    setbuf(stdout, NULL);
    double amount = 0;
    // Ask user for desired seconds of flashing
    printf("How many seconds would you like the message to flash?  ");
    // Only continue when we have valid input
    while (amount == 0) {
        // Scan for user input
        char input[20] = "";
        scanf("%s", input);
        // Convert input to a double
        amount = strtod(input, NULL);
        // If input is invalid, inform the user
        if (amount == 0.0) {
            printf("Invalid input. Try again: ");
        }
    }
    while (1) {
        // Print the message then pause for a second
        flData(amount);

        // Clear the message then pause for a second
        f2Clear(amount);
    }

    return EXIT_SUCCESS;
}

void delay(double seconds) {
    time_t startTime, currTime = 0;
    // Get the time we started at and set the current time to the same time
    time(&startTime);
    currTime = startTime;
    // Loop until the difference in the current time and the start time is
    // at least the given number of seconds
    while (difftime(currTime, startTime) < seconds) {
        // Get the new current time
        time(&currTime);
    }
}

void flData(double delay1) {
    // Sets the message we want to flash
    char* message = "A B C D";
    // Print the message then delays for the given number of seconds
    printf("%s", message);
    delay((double) delay1);
}

void f2Clear(double delay2) {
    char* clear = "\b\b\b\b\b\b\b";
    // Print the message then delays for the given number of seconds
    printf("%s", clear);
    delay((double) delay2);
}
```

Figure 3 – Application 3

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

// Delays the code continuing to run for the given number of seconds.
// Seconds must be greater than or equal to 0
void delay(double seconds);

// Prints the message "A B C D" in the console then pauses for the given
// delay in seconds
void f1Data(double* delay1);

// Clears the console then pauses for the given delay in seconds
void f2Clear(double* delay2);

// Flashes the message "A B C D" for a number of seconds then clears it for
// another number of seconds. It repeats this behavior indefinitely so the
// program must be manually stopped
int main() {
    // Makes sure the console's buffer prints without a new line needed
    setbuf(stdout, NULL);
    double amount = 0;
    // Ask user for desired seconds of flashing
    printf("How many seconds would you like the message to flash?  ");
    // Only continue when we have valid input
    while (amount == 0) {
        // Scan for user input
        char input[20] = "";
        scanf("%s", input);
        // Convert input to a double
        amount = strtod(input, NULL);
        // If input is invalid, inform the user
        if (amount == 0.0) {
            printf("Invalid input. Try again: ");
        }
    }
    while (1) {
        // Print the message then pause for the message delay
        f1Data(&amount);

        // Clear the message then pause for the clear delay
        f2Clear(&amount);
    }

    return EXIT_SUCCESS;
}

void delay(double seconds) {
    time_t startTime, currTime = 0;
    // Get the time we started at and set the current time to the same time
    time(&startTime);
    currTime = startTime;
    // Loop until the difference in the current time and the start time is
    // at least the given number of seconds
    while (difftime(currTime, startTime) < seconds) {
        // Get the new current time
        time(&currTime);
    }
}

void f1Data(double* delay1) {
    // Sets the message we want to flash
    char* message = "A B C D";
    // Print the message then delays for the given number of seconds
    printf("%s", message);
    delay((double) *delay1);
}

void f2Clear(double* delay2) {
    char* clear = "\b\b\b\b\b\b\b";
    // Print the message then delays for the given number of seconds
    printf("%s", clear);
    delay((double) *delay2);
}
```

Figure 4 – Application 4

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
// Delays the code continuing to run for the given number of seconds.
// Seconds must be greater than or equal to 0
void delay(double seconds);

// Prints the given message in the console then pauses for the given
// delay in seconds
void displayAndDelay(char* message, double* toDelay);


// Flashes the message "A B C D" for a number of seconds then clears it for
// another number of seconds. It repeats this behavior indefinitely so the
// program must be manually stopped
int main() {
    // Makes sure the console's buffer prints without a new line needed
    setbuf(stdout, NULL);
    // Sets the message to A B C D to flash
    char* message = "A B C D";
    double amount = 0;
    // Ask user for desired seconds of flashing
    printf("How many seconds would you like the message to flash?  ");
    // Only continue when we have valid input
    while (amount == 0) {
        // Scan for user input
        char input[20] = "";
        scanf("%s", input);
        // Convert input to a double
        amount = strtod(input, NULL);
        // If input is invalid, inform the user
        if (amount == 0.0) {
            printf("Invalid input. Try again: ");
        }
    }
    // Sets the string used to clear the message
    char* clear = "\b\b\b\b\b\b\b";
    // Loop indefinitely
    while (1) {
        // Print the message then pause for the message delay
        displayAndDelay(message, &amount);
        // Clear the message then pause for the clear delay
        displayAndDelay(clear, &amount);
    }

    return EXIT_SUCCESS;
}

void delay(double seconds) {
    time_t startTime, currTime = 0;
    // Get the time we started at and set the current time to the same time
    time(&startTime);
    currTime = startTime;
    // Loop until the difference in the current time and the start time is
    // at least the given number of seconds
    while (difftime(currTime, startTime) < seconds) {
        // Get the new current time
        time(&currTime);
    }
}

void displayAndDelay(char* message, double* toDelay) {
    // Print the message then delays for the given number of seconds
    printf("%s", message);
    delay((double) *toDelay);
}

```

Figure 5 – Application 5

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include "app6_func.h"

// Flashes the message "A B C D" for a number of seconds then clears it for
// another number of seconds. It repeats this behavior indefinitely so the
// program must be manually stopped
int main() {
    // Makes sure the console's buffer prints without a new line needed
    setbuf(stdout, NULL);
    // Sets the message to A B C D to flash
    char* message = "A B C D";
    double amount = 0;
    // Ask user for desired seconds of flashing
    printf("How many seconds would you like the message to flash?  ");
    // Only continue when we have valid input
    while (amount == 0) {
        // Scan for user input
        char input[20] = "";
        scanf("%s", input);
        // Convert input to a double
        amount = strtod(input, NULL);
        // If input is invalid, inform the user
        if (amount == 0.0) {
            printf("Invalid input. Try again: ");
        }
    }
    // Sets the string used to clear the message
    char* clear = "\b\b\b\b\b\b\b";
    // Loop indefinitely
    while (1) {
        // Print the message then pause for the message delay
        displayAndDelay(message, &amount);
        // Clear the message then pause for the clear delay
        displayAndDelay(clear, &amount);
    }

    return EXIT_SUCCESS;
}
```

Figure 6 – Application 6 (main method)

```c
#include <stdio.h>
#include <time.h>
#include "app6_func.h"

void delay(double seconds) {
    time_t startTime, currTime = 0;
    // Get the time we started at and set the current time to the same time
    time(&startTime);
    currTime = startTime;
    // Loop until the difference in the current time and the start time is
    // at least 1 second
    while (difftime(currTime, startTime) < seconds) {
        // Get the new current time
        time(&currTime);
    }
}

void displayAndDelay(char* message, unsigned long* toDelay) {
    // Print the message then delays for the given number of seconds
    printf("%s", message);
    delay((double) *toDelay);
}
```

Figure 7 – Application 6 (functions program)

```
// Delays the code continuing to run for the given number of seconds.
// Seconds must be greater than or equal to 0
void delay(double seconds);

// Prints the given message in the console then pauses for the given
// delay in seconds
void displayAndDelay(char* message, unsigned long* toDelay);
```

Figure 8 – Application 6 (function header file)

```
#include <stdio.h>

//  this is a simple routine that demonstrates how to fill and display an array of characters

int main()
{
    int i = 0;                                  //  declare a working variable

    char myArray[5];                            //  declare a character array

    for (i = 0; i < 5; i++)                 //  fill array with characters
    {
        //  fill with the ascii characters A..F
        //  65 is the ascii value for A

        myArray[i]= 65+i;
    }

    for (i = 0; i < 5; i++)                 //  display the array
    {
        printf("%c \n", myArray[i]);
    }

    printf("\n");

    return 0;
}
```

Figure 9 – Troubleshooting project1b-2021.b

```
#include <stdio.h>

//  function prototypes

//  get data from the user
void getData(int* aValuePtr);

int main ()
{
    //  declare a shared variable and a pointer to it
    int myValue;
    int* myPtr = &myValue;      //  let myPtr point to myValue

    //  get data from the user
    getData(myPtr);

    //  display the data as a character
    printf("The data is: %c \n", *myPtr);

    return 0;

}

//  prompt the user for some data and return it through a shared
//  variable pointed to by valuePtr
//
//  inputs:     pointer to a container in which to place the data
//  outputs:    none
//  function:   the routine accepts a pointer to a container in which to store data from a user,
//              it prompts for the data, accepts the data, displays it, and returns

void getData(int* valuePtr)
{
    //  declare a temp place to store the data
    int tempValue;

    //  prompt for data
    printf("Please enter a single digit between 0-9 \n");

    //  get the data
    tempValue = getchar();

    //  dereference valuePtr so that its value is now the data
    *valuePtr = tempValue;

    //  display its value as a character
    printf("The data is: %c \n", *valuePtr);

    return;

}
```

Figure 10 – Troubleshooting project1b-2021.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Elegoo_GFX.h>    // Core graphics library
#include <Elegoo_TFTLCD.h> // Hardware-specific library

// The control pins for the LCD can be assigned to any digital or
// analog pins...but we'll use the analog pins as this allows us to
// double up the pins with the touch screen (see the TFT paint example).
#define LCD_CS A3 // Chip Select goes to Analog 3
#define LCD_CD A2 // Command/Data goes to Analog 2
#define LCD_WR A1 // LCD Write goes to Analog 1
#define LCD_RD A0 // LCD Read goes to Analog 0

#define LCD_RESET A4 // Can alternately just connect to Arduino's reset pin

#define MAX_LEN 32


// Assign human-readable names to some common 16-bit color values:
#define  BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF

Elegoo_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
// If using the shield, all control and data lines are fixed, and
// a simpler declaration can optionally be used:
// Elegoo_TFTLCD tft;

// The message we're printing
String message = "A B C D";
// Initialize the delay outside of either method's scope
// so both can access the values
int messageDelay = 0;

// Flashes the message "A B C D" for a number of seconds given by the user.
// It repeats this behavior indefinitely.
void setup() {
  Serial.begin(9600);
  Serial.println(F("TFT LCD test"));
```

```
#ifdef USE_Elegoo_SHIELD_PINOUT      // This is defined in Elegoo_TFTLCD.h
  Serial.println(F("Using Elegoo 2.4\" TFT Arduino Shield Pinout"));
#else
  Serial.println(F("Using Elegoo 2.4\" TFT Breakout Board Pinout"));
#endif

  Serial.print("TFT size is "); Serial.print(tft.width()); Serial.print("x");
Serial.println(tft.height());

  tft.reset();

  uint16_t identifier = tft.readID();
  if (identifier == 0x9325) {
    Serial.println(F("Found ILI9325 LCD driver"));
  } else if (identifier == 0x9328) {
    Serial.println(F("Found ILI9328 LCD driver"));
  } else if (identifier == 0x4535) {
    Serial.println(F("Found LGDP4535 LCD driver"));
  } else if (identifier == 0x7575) {
    Serial.println(F("Found HX8347G LCD driver"));
  } else if (identifier == 0x9341) {
    Serial.println(F("Found ILI9341 LCD driver"));
  } else if (identifier == 0x8357) {
    Serial.println(F("Found HX8357D LCD driver"));
  } else if (identifier == 0x0101)
  {
    identifier = 0x9341;
    Serial.println(F("Found 0x9341 LCD driver"));
  }
  else if (identifier == 0x1111)
  {
    identifier = 0x9328;
    Serial.println(F("Found 0x9328 LCD driver"));
  }
  else {
    Serial.print(F("Unknown LCD driver chip: "));
    Serial.println(identifier, HEX);
    Serial.println(F("If using the Elegoo 2.8\" TFT Arduino shield, the line:"));
    Serial.println(F("  #define USE_Elegoo_SHIELD_PINOUT"));
    Serial.println(F("should appear in the library header (Elegoo_TFT.h)."));
    Serial.println(F("If using the breakout board, it should NOT be #defined!"));
    Serial.println(F("Also if using the breakout, double-check that all wiring"));
    Serial.println(F("matches the tutorial."));
    identifier = 0x9328;
```

```
  }
  tft.begin(identifier);
  // Asks the user for input
  Serial.print("How many seconds would you like the message to flash?  ");
  // Gives the user time to enter input
  while (messageDelay == 0) {
    messageDelay = Serial.parseInt();
  }
  Serial.println(messageDelay);
}

void loop() {
  // Clears the screen
  tft.fillScreen(BLACK);
  // Pauses for the clear delay
  delay(messageDelay * 1000);
  // Resets the cursor and sets the text color and size
  tft.setCursor(0, 0);
  tft.setTextColor(GREEN); tft.setTextSize(2);
  // Prints the message
  tft.print(message);
  // Pauses for message delay
  delay(messageDelay * 1000);
}
```

Figure 11 – Application 6 on ATMega 2560

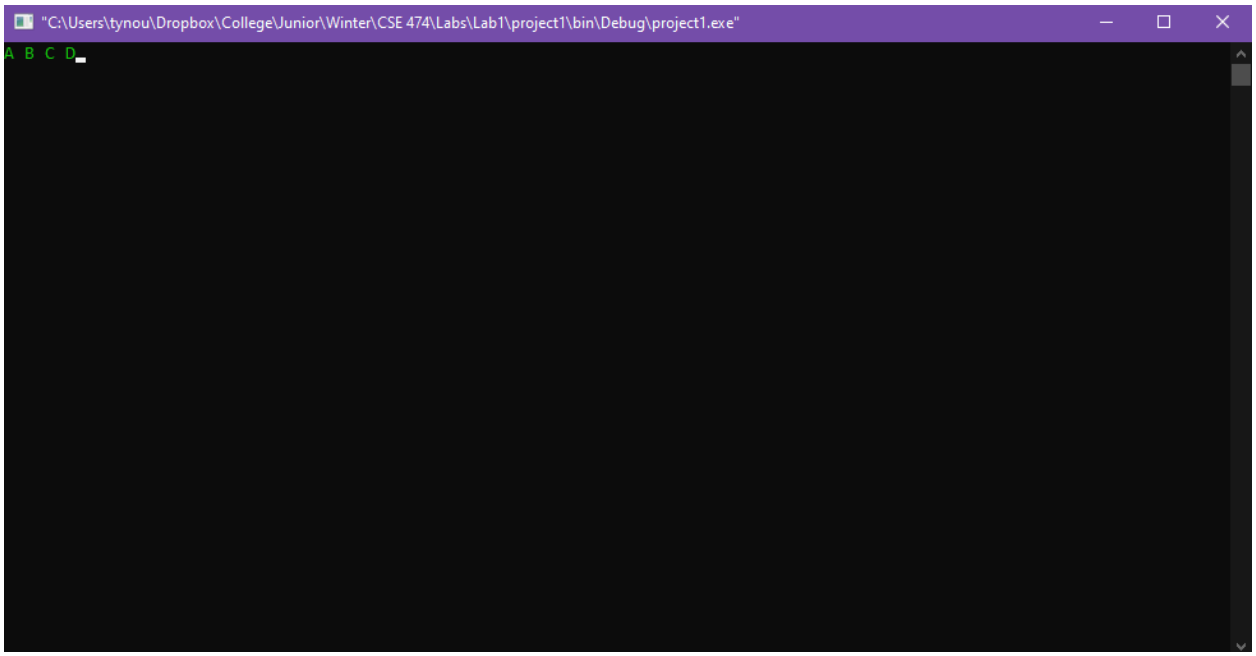

Figure 12 – Console of application 1

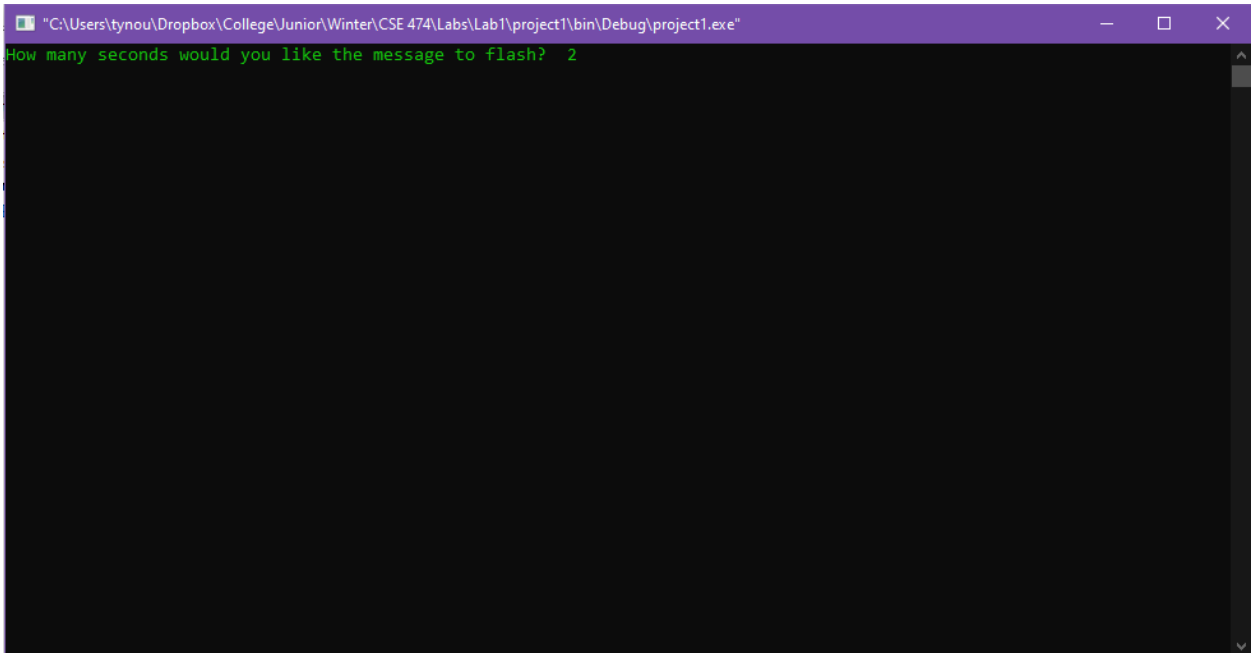

Figure 13 – Console of application 2-6 (asking the user)

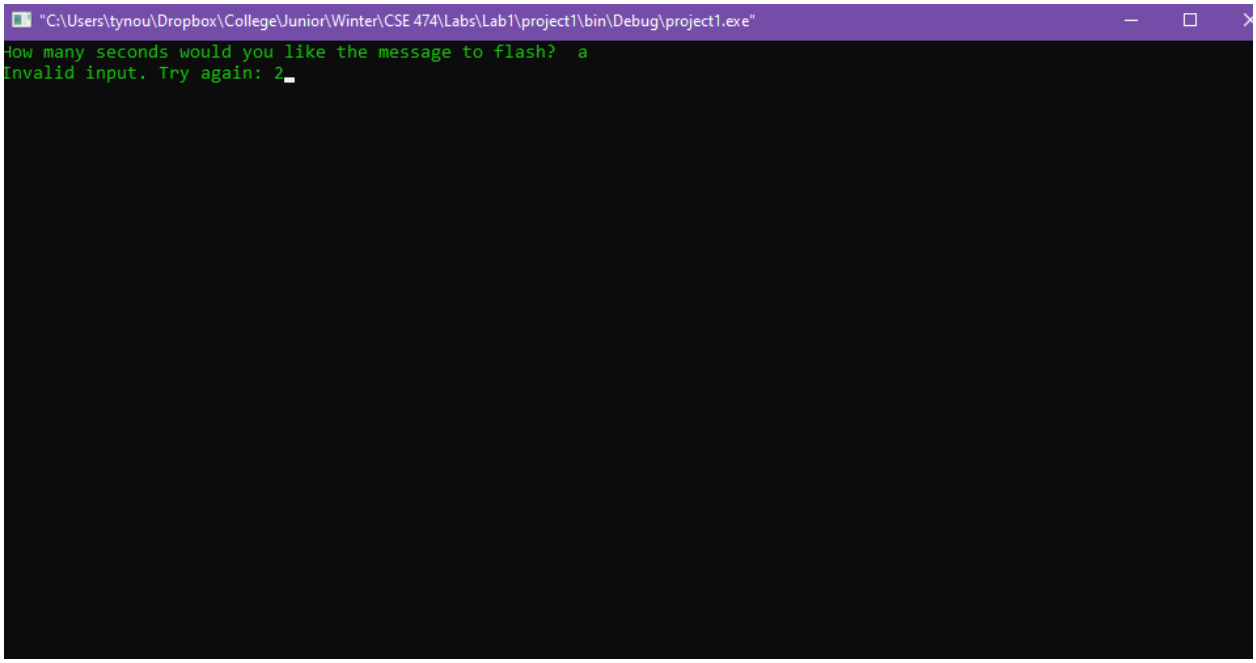

Figure 14 – Console of application 2-6 (invalid input)

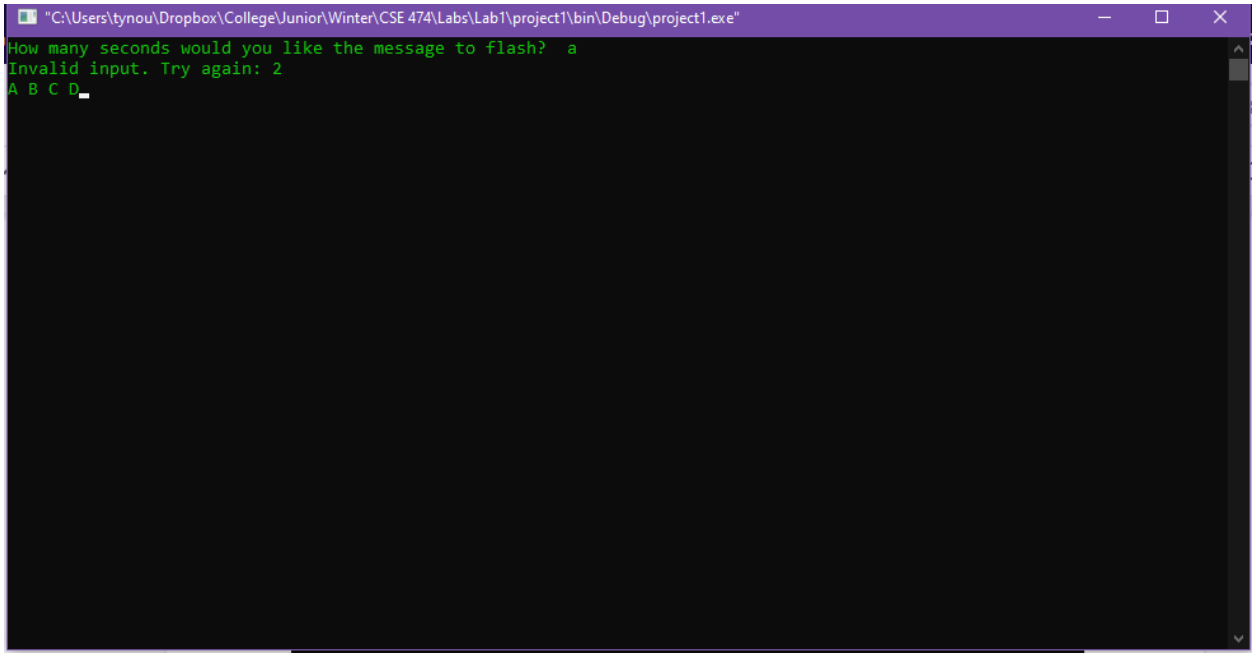

Figure 15 – Console of application 2-6 (output message and delay for given seconds)

Figure 16 – Output on ATMega 2560