

EE/CSE 474 Lab Report
Project 1



Radleigh Ang
Abdul Katani
Daniel Snitkovskiy

ABSTRACT

In this lab our main goal was to get a better understanding of the C language and the *BeagleBone Black* microprocessor. We achieved this goal by editing the given programs, utilizing the given debugger, and building our own applications.

First, we edited the given program for Project 1a, created a working directory in the *BeagleBone Black* via terminal, compiled it using the GNU Compiler Collection (gcc), and finally running the executable. After getting a solid foundation, we began exploring the C language further by modifying the program code in 5 ways:

1. Parametrizing the two delay() functions in the two for loops.
2. Replacing the two for loops with the given two functions
3. Passing the delay by parameter instead of having it passed by value
4. Consolidating the two functions into one that is parametrized by the character to be displayed and the value of the delay.
5. Separating the two functions into two files.

Second, we added the given program for Project 1b and Project 1c to the *BeagleBone Black*, compiled these with the debug flag on, and proceeded to pinpoint and repair several bugs within the source code via the GNU Debugger (gdb).

Lastly, we built our own applications according to the Application 1-3 specifications given in the lab manual. The objective was to display “ABCD” to the *BeagleBone Black* console such that:

1. All four letters would display on the console in one second increments.
2. Each individual letter would display in one second increments.
3. A and C would display in one second increments, while B and D would display in two second increments.

After finishing project, a more thorough understanding of the Linux development environment, C, and programming on the *BeagleBone Black* was achieved.

TABLE OF CONTENTS

1. INTRODUCTION	4
2. DESIGN SPEC	4-5
2.1 Working with the existing project file	4
2.2 Working with GDB	4
2.3 Building our Own Applications	5
3. SOFTWARE IMPLEMENTATION	5-10
3.1 Project1a	5-8
3.1.1 Question 7	5-6
3.1.2 Question 8	6
3.1.3 Question 9	7
3.1.4 Question 10	8
3.1.5 Question 11	8
3.2 Application 1	9
3.3 Application 2	10
3.4 Application 3	10
4. DESIGN TESTING	11
5. PRESENTATION AND RESULTS	11-16
5.1 Project1a	11
5.2 Working with the Debugger	12-14
5.2.1 Debugging Project1b	12
5.2.2 Debugging Project1c	13-14
5.3 Application 1	15
5.4 Application 2	15-16
5.5 Application 3	16
6. ANALYSIS OF ANY ERRORS	17-18
7. SUMMARY AND CONCLUSION	19
8. APPENDICES	19
9. CONTRIBUTIONS	19

1. INTRODUCTION

The purpose of this lab was to become familiar with the C programming language and Linux based development on the *AM3358BZCZ100 BeagleBone Black* microprocessor. The primary tools that were employed were the a *Ubuntu 16.04.2 LTS Work Station*, *GCC 4.6.3 Debian compiler*, and *GDB 7.4.1 Debian debugger*.

2. DESIGN SPEC

This section shows the requirements for Project 1 that are to be presented in the report.

2.1 Working with the existing project file

7. Modify the program (project1a-2017.c) to parameterize the two delay() function calls in the two for loops so that they will support different user specified delays rather than the single hard coded value as they are now.

Note for Step 7: The source code project1a had already implemented question 7's requirement.

8. Modify the program so that each of the two respective for loops is replaced by the following functions.

```
void f1Data(unsigned long delay1);
```

```
void f2Clear(unsigned long delay2);
```

9. Modify the program so that the delay parameter is passed into the two functions by pointer reference rather than by value.

10. Modify the program so that the two functions are replaced by a single function. The function should be able to be called with the character to be displayed and the value of the delay.

11. Modify the program so that the function you wrote in part 10 is in a separate file. Your program will now be composed of two files.

2.2 Working with GDB

Use the debugger to identify what the problem is for files project1b-2017.c and project1c-2017.c. Indicate how you found it with the debugger. Correct the problems and using your debugger, prove that you have, indeed, fixed the problems.

2.3 Building our Own Applications

Application 1

Using what you have learned from the example programs in the previous exercises, write a program that will display the letters: A B C D on the BeagleBone console and flash them together at approximately a one-second rate.

Application 2

Modify the program in Application 1 to print then erase the letters A then B then C then D at approximately a one-second rate.

Application 3

Modify the program in Application 1 to flash the letters A and C at a one-second rate and the letters B and D at a two-second rate.

3. SOFTWARE IMPLEMENTATION

The following shows the pseudo-code implementation for every program that we have attached.

3.1 Project1a

3.1.1 Question 7 (Source code in q7.c)

The given source code defines a function called “delay” that takes in a user-specified delay and uses this to display the numbers 0-9 in descending.

main():

```
    initialize i and k = 0;
    print(the value of i is: “ ”, i);
    add_output_to_console();
    while (true) {
        for(i from 9 to 0) {
            print “i”;
            add_output_to_console();
            delay(specified_delay);
        }
        print_carriage_return();
        add_output_to_console();
        for(i from 0 to 9) {
            print “ ” // two spaces
            add_output_to_console();
            delay(specified_delay);
        }
        print_carriage_return();
        add_output_to_console();
    }
```

```

delay(specified_delay):
    initialize i and j;
    for(i from specified_delay to 1) {
        for (j from 0 to 99999) {
            // do nothing
        }
    }

```

3.1.2 Question 8 (*Source code in q8.c*)

Here, we've separated the two for loops in "main()" into "f1Data()" and "f2Clear()" in order to improve modularity.

```

main():
    while(true) {
        f1Data(specified_delay);
        f2Clear(specified_delay);
    }

```

```

f1Data(specified_delay):
    for(i from 9 to 0) {
        print "i";
        add_output_to_console();
        delay(specified_delay);
    }
    print_carriage_return();
    add_output_to_console();

```

```

f2Clear(specified_delay):
    for(i from 0 to 9) {
        print " " // two spaces
        add_output_to_console();
        delay(specified_delay);
    }
    print_carriage_return();
    add_output_to_console();

```

delay(specified_delay): Reference section 3.1.2 for pseudo-code implementation.

3.1.3 Question 9 (Source code in q9.c)

Here we pass the delays by reference rather than by value in order to reduce the total amount of bytes copied during each function call.

main():

```
    initialize delay1 and delay2;
    initialize pointer1 and pointer2 to delay1 and delay 2, respectively.
    while(true) {
        f1Data(pointer1)
        f2Clear(pointer2)
    }
```

f1Data(pointer1):

```
    for(i from 9 to 0) {
        print "i";
        add_output_to_console();
        delay(pointer1);
    }
    print_carriage_return();
    add_output_to_console();
```

f2Clear(pointer2):

```
    for(i from 0 to 9) {
        print "  " // two spaces
        add_output_to_console();
        delay(pointer2);
    }
    print_carriage_return();
    add_output_to_console();
```

delay(delay_ptr):

```
    initialize i and j;
    initialize delay to dereference(delay_ptr);
    for(i from delay to 1) {
        for (j from 0 to 99999) {
            // do nothing
        }
    }
```

3.1.4 Question 10 (Source code in q10.c)

Here, “f1Clear()” and “f2Data()” are merged into one function in order to reduce the size of the overall program.

main():

```
    initialize delay1 and delay2;
    initialize pointer1 and pointer2 to delay1 and delay 2, respectively;
    while(true) {
        for(i from 9 to 0) {
            printChar(pointer1, i); // print “i” w/ pointer1
        }
        for(i from 0 to 9) {
            printChar(pointer2, ' ') // print a space w/ pointer2
        }
        print_carriage_return();
    }
```

printChar(delay_ptr, char_to_print):

```
    if(char_to_print is a non-space character) {
        append '0'; // correct ascii character
    }
    print(char_to_print);
    add_output_to_console();
    delay(delay_ptr);
```

delay(delay_ptr): Reference section 3.1.3 for pseudo-code implementation.

3.1.5 Question 11 (Source code in q11.c, printChar.c, and printChar.h)

Here, we separate the “printChar()” into a separate file, allowing future clients to import this function into their source code.

q11.c:

main(): Reference section 3.1.4 for implementation.

delay(delay_ptr): Reference section 3.1.3 for pseudo-code implementation.

printChar.h:

// Contains function prototype for printChar()

printChar.c:

printChar(delay_ptr, char_to_print): Reference the “printChar()” function in section 3.1.4 for pseudo-code implementation.

3.2 Application 1 (*Source code in app1.c*)

The following shows the pseudo-code implementation of Application 1.

main():

```
declare letters[size_of_letters] = "ABCD";
while(true) {
    display(letters, size_of_letters);
    print_carriage_return();
    delay(one_sec);
    undisplay (size_of_letters);
    print_carriage_return();
    delay(one_sec);
}
```

delay(specified_delay): Reference section 3.1.2 for pseudo-code implementation.

display(letters, size_of_letters):

```
initialize i;
for (i from 0 to size_of_letters - 1) {
    print letters[i]; // i'th character of letters;
    add_output_to_console();
}
```

undisplay(size_of_letters):

```
for (i from 0 to size_of_letters - 1) {
    print " "; // one space
    add_output_to_console();
}
```

3.3 Application 2 (*Source code in app2.c*)

The following shows the pseudo-code implementation of Application 2.

main():

```
declare letters[size_of_letters] = "ABCD";
while(true) {
    for (int i = 0; i < SIZE; i++) {
        display(pointer_to_letters[i], 1); // i'th character of letters
        print_carriage_return();
        delay(one_sec);
        undisplay(i + 1); // i'th character + a space
        delay(one_sec);
    }
    print_carriage_return();
}
```

delay(specified_delay): Reference section 3.1.2 for pseudo-code implementation.

display(letters, size_of_letters): Reference section 3.2 for pseudo-code implementation.

undisplay(size_of_letters): Reference section 3.2 for pseudo-code implementation.

3.4 Application 3 (*Source code in app3.c*)

The following shows the pseudo-code implementation of Application 3.

main():

```
declare letters[size_of_letters] = "ABCD";
declare spaces[size_of_letters] = "    "; // four spaces
while(true) {
    for (i from 0 to size_of_letters) {
        if(0th iteration) {
            spaces = "A C "
        } else if(2nd iteration) {
            spaces = "ABCD"
        } else { // 1st or 3rd iteration
            spaces = "    "; // four spaces
        }
        display(spaces, size_of_letters);
        print_carriage_return();
        delay(one_sec);
    }
}
```

display(letters, size_of_letters): Reference section 3.2 for pseudo-code implementation.

4. DESIGN TESTING

In this section, we describe the strategies that were employed during testing and debugging.

Testing:

1. Compile target program to test (call this “test.c”) via gcc.
2. Run target program (call this “test”) within bash.
3. Task a group member to inspect output of the console and compare said output against expected output.
4. If there are timing requirements, task 1 group member to signal auditorily (e.g. tapping) at each desired time step (e.g. 1 second intervals). Meanwhile, task another group member to verify that the output meets the timing requirements.

Debugging:

1. If incorrect output:
 - a. Set a breakpoint on all functions that print to the console.
 - b. Inspect variables containing output to be printed to the console before and after assignment.
2. If incorrect timing:
 - a. Exaggerate delays (multiply by large scalars) to pinpoint the cause and effect of delays in specified locations.
 - b. Binary search to approximate timing. That is, if “too slow”, cut the delay in half; if “too fast” double the delay.

5. PRESENTATION AND RESULTS

In the following sections, we present a proof of concept for our proposed designs, as well as the answers to the questions posed in the lab manual.

5.1 Project1a

For each modification of Project1a, we were able to display 9 through 0 to the *BeagleBone Black* console and erase them with the appropriate delays. Below shows a sample execution.

```
The value of i is: 0
3 2 1 0
```

5.2 Working with the Debugger

The following subsections will explain the errors of the programs that were found by the GDB debugger and how these errors were solved.

5.2.1 Debugging Project1b

Problem:

This program was designed to fill an array called “myArray” with letters from A-F, in order and print its contents.

```
[(gdb) l 18
13             // fill with the ascii characters A..F
14             // 65 is the ascii value for A
15
16             myArray[i]= 65+i;
17         }
18
19         for (i = 0; i <= 5; i++)                // display the array
20         {
21             printf("%c \n", myArray[i]);
22         }
[(gdb) b 19
Breakpoint 5 at 0x4005b9: file project1b-2017.c, line 19.
[(gdb) p myArray
$2 = "ABCDE"
(gdb) □
```

Although the output is ostensibly correct, when using GDB, we realize that myArray’s contents are only storing A-E.. Thus, we can surmise that myArray isn’t big enough to store the string A-F.

Solution:

To fix this issue, we define myArray with a size of 6, rather than 5.

```
void main(void)
{
    int i = 0;

    char myArray[6];

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

        myArray[i]= 65+i;
    }
}
```

Proof of Fix:

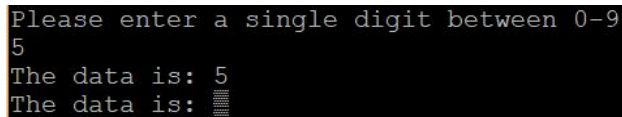
As we can see in the screenshot, myArray now has enough space to store A-F.

```
Breakpoint 1, main () at project1b-2017.c:19
19         for (i = 0; i <= 5; i++)
[(gdb) l 19
14             // 65 is the ascii value for A
15
16             myArray[i]= 65+i;
17         }
18
19         for (i = 0; i <= 5; i++)
20         {
21             printf("%c \n", myArray[i]);
22         }
23
[(gdb) p myArray
$1 = "ABCDEF"
(gdb) █
```

5.2.2 Debugging Project1c

Problem:

This program should print the ASCII character version of a user-specified integer provided via console input. However, as one can see from the below screenshot, the value printed does not match the user-specified integer.



```
Please enter a single digit between 0-9
5
The data is: 5
The data is: █
```

Using GDB, we track the address of myPtr (which should hold the user input) and the value it actually holds via breakpoints throughout execution.

```
Breakpoint 1, main () at project1c-2017.c:15
15         getData(myPtr);
[(gdb) p myPtr
$2 = (int *) 0x7fffffffdb4
[(gdb) p *myPtr
$3 = 32767
(gdb) █
```

Right before myPtr is passed into getData, we note that myPtr has an address which holds an undefined value (randomly set to 32767 as shown in the below screenshot).

```
Breakpoint 3, main () at project1c-2017.c:18
18         printf("The data is: %c \n", *myPtr);
[(gdb) p *myPtr
$7 = 32767
(gdb) █
```

After `getData` is executed, `myPtr` still holds the same value (see screenshot). Thus, when `getData` is executed, it does not change the value of `myPtr` as expected.

Within `getData`, we notice this line of code:

```
valuePtr = &tempValue;
```

This line of code changes `valuePtr`, which refers to `myPtr` in `main()`, to point to `tempValue`. Since `*valuePtr` gets `tempValue`, changing `*valuePtr` has no effect on `*myPtr` in the main function.

Solution:

To fix this issue, we simply comment out the offending lines (see screenshot below).

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

    // let valuePtr point to it
    // valuePtr = &tempValue;

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

Proof of Fix:

As we can see, the console output now matches the input from the user in both `getData` and in `main`.

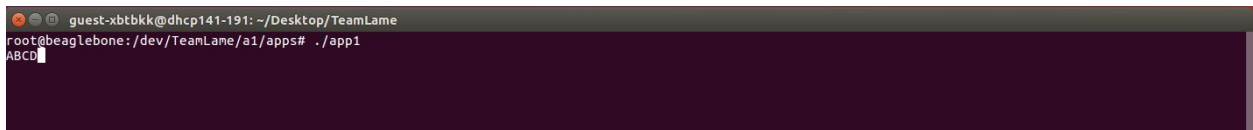
```
Please enter a single digit between 0-9
5
The data is: 5
The data is: 5
```

5.3 Application 1

The following sequence of screenshots illustrates a sample execution of Application 1 for one iteration of the while loop. Note that the time between each screenshot is approximately 1 second.

A terminal window with a dark purple background. The title bar shows 'guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame'. The prompt is 'root@beaglebone:/dev/TeamLame/a1/apps#'. The command './app1' has been entered, and the cursor is on the next line.

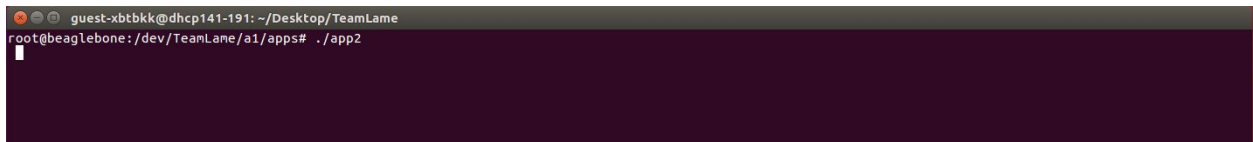
(blank console)

A terminal window with a dark purple background. The title bar shows 'guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame'. The prompt is 'root@beaglebone:/dev/TeamLame/a1/apps#'. The command './app1' has been entered, and the output 'ABCD' is visible on the next line.

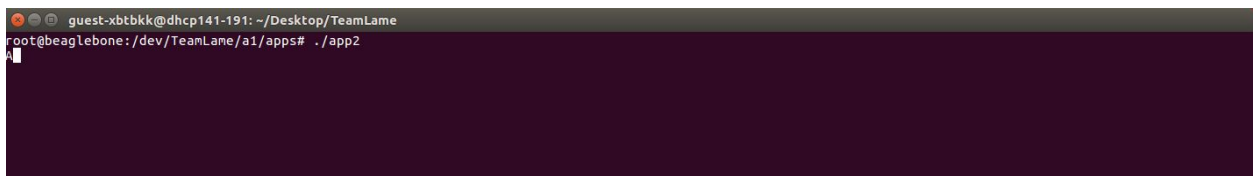
(“ABCD” flash on the console)

5.4 Application 2

The following sequence of screenshots illustrates a sample execution of Application 2 for one iteration of the while loop. Note that the time between each screenshot is approximately 1 second.

A terminal window with a dark purple background. The title bar shows 'guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame'. The prompt is 'root@beaglebone:/dev/TeamLame/a1/apps#'. The command './app2' has been entered, and the cursor is on the next line.

(blank console)

A terminal window with a dark purple background. The title bar shows 'guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame'. The prompt is 'root@beaglebone:/dev/TeamLame/a1/apps#'. The command './app2' has been entered, and the output 'A' is visible on the next line.

(“A” flashes on the console)

A terminal window with a dark purple background. The title bar shows 'guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame'. The prompt is 'root@beaglebone:/dev/TeamLame/a1/apps#'. The command './app2' has been entered, and the output 'B' is visible on the next line.

(“B” flashes on the console)

```
guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame
root@beaglebone:/dev/TeamLame/a1/apps# ./app2
C
```

(“C” flashes on the console)

```
guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame
root@beaglebone:/dev/TeamLame/a1/apps# ./app2
D
```

(“D” flashes on the console)

5.5 Application 3

The following sequence of screenshots illustrates a sample execution of Application 3 for one iteration of the while loop. Note that the time between each screenshot is approximately 1 second.

```
guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame
root@beaglebone:/dev/TeamLame/a1/apps# ./app3
```

(blank console)

```
guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame
root@beaglebone:/dev/TeamLame/a1/apps# ./app3
A C
```

(“AC” flashes on the console)

```
guest-xbtbkk@dhcp141-191: ~/Desktop/TeamLame
root@beaglebone:/dev/TeamLame/a1/apps# ./app3
ABCD
```

(“ABCD” flashes on the console)

6. ANALYSIS OF ANY ERRORS

In this section, we describe all of the errors that were encountered during development, and what was implemented in order to remedy these errors.

Filename	File Description	Error (s)	Cause (s)	Resolution
q7.c	Question 7	No errors.	No errors.	No errors.
q8.c	Question 8	No errors.	No errors.	No errors
q9.c	Question 9	1. Error: Initialization makes pointer from integer without a cast.	1. Trying to assign a pointer to a number	1. Initialize an integer and then another pointer to this integer.
q10.c	Question 10	1. Character to be printed formatted incorrectly. 2. Spaces rendered as "P" when printed.	1. Integers passed coerced as characters missing a '0' char at the end. 2. '0' char was being added to spaces.	1. Add '0' to the end of character to be printed. 2. Implement ternary assignment to add '0' char only if character to be printed is not a space.
q11.c	Question 11	No errors	No errors	No errors
printChar.c	Question 11	Same as q10.c	Same as q10.c	Same as q10.c
printChar.h	Question 11	No errors	No errors	No errors

Filename	File Description	Error (s)	Cause (s)	Resolution
app1.c	Application 1	1. Characters and spaces were printed consecutively rather than in one location (i.e. "A B C D " would be printed further and further along the console).	1. There was no carriage return between displaying/un-displaying characters.	1. Print a carriage return after each function call that involved printing to the console.
app2.c	Application 2	1. All characters other than 'A' would be printed on the second space within the console.	1. Undisplaying only erased 1 characters' worth, point are which the console printed/erased on the second space after the 0'th iteration.	1. Undisplay based on the index. That is, erase 1 character's worth for the first iteration, 2 character's worth for the second, and so on.
app3.c	Application 3	1. Printing AC and BD out of sync.	1. Delays were not correct.	1. Created a spaces string to represent the blank output of the console.

7. SUMMARY AND CONCLUSION

Through this project, we have gained proficiency in the C language, and the Linux development environment for deploying applications on the the *BeagleBone Black*.

To gain proficiency in the C language, we utilized and modified the provided code, as well as building our own programs to meet specified behavior. After working with the provided code, we learned: how to parameterize a function, how to merge/separate functions based on behavior, and how to practice effective debugging. To put these new skills into practice, we built Applications 1-3, which were three variations of one basic specification: display the letters “ABCD” in a specific order within a specific time interval. By using principles of modularity, we were able to use the same basic functions (display, undisplay, and delay) with few modifications across all three Applications.

All of these programs were executed on the *BeagleBone Black*, which gave us the necessary exposure to the deployment process for implementing future programs on this system.

8. APPENDIX

All information and requirements were taken from EE/CSE 474 Project 1, Introducing the Lab Environment: <https://class.ee.washington.edu/474/peckol/assignments/lab1/lab1Summer17.pdf>

All source code are included in “a1.zip”. Reference README.md in the root directory for build instructions and file locations.

9. CONTRIBUTION

Name	Contribution
Abdul Katani	<ul style="list-style-type: none">• Co-wrote q10-q11 in project1a.• Co-designed and co-tested applications 1-2.• Contributed to the lab report.
Radleigh Ang	<ul style="list-style-type: none">• Used GDB to debug project1b and project1c.• Added modifications to q7 - q9 in project1a.• Designed and tested application 3.• Set up and contributed to lab report.
Daniel Snitkovskiy	<ul style="list-style-type: none">• Co-wrote q10-q11 in project1a.• Co-designed and co-tested applications 1-2.• Contributed to the lab report.