Matthew Cummings
December 2, 2021
CISS 247 Computer Systems
Assignment 3 Report

Introduction
For this assignment, we were to write a C program to display a given number using only the underscore "_", vertical bar "|", and space characters. In addition, we were given certain coding standards to adhere to. I have not chosen to partake in the extra credit extensions to my code, and the name of my final file will be "display7.c".

1. A function to display the top zone characters for all the digits in the number.
2. A function to display the top zone characters for one digit.
3. A function to display the middle zone characters for all the digits in the number.
4. A function to display the middle zone characters for one digit.
5. A function to display the bottom zone characters for all the digits in the number.
6. A function to display the bottom zone characters for one digit.

Process
The first function to implement is main(), which will call all other helper functions, as well as handle user input in the form of command line arguments.

```
/*
 * argc: num of arguments
 * argv: array of arguments
 * argv[1]: number to print in 7 segment style
*/
int main(int argc, char *argv[]) {

    // check user entered number to display
    if (argc != 2) {
        printf("Error. Check arguments.\nUsage: ./display7.c 123456789\n");
        return 0;
    }
```

Here, my program will warn the user whenever they do not input the command line argument correctly. Only when 1 argument other than the program name is given will the rest of the program execute.

```
// find length of inputted number
int intLen = strlen(argv[1]);

// iterate over argv[1] and check it contains a number in every char.
// if argv[1] contains an int, then intChars will equal intLen. otherwise,
// the user has not correctly inputted an int in argv[1] and this
// program must terminate
int intChars = 0;
for (int i = 0; i < intLen; i++) {
    if (isdigit(argv[1][i])) {
        intChars++;
    }
} if (intChars != intLen) {
    printf("Please enter a positive number instead of \"%s\".\n", argv[1]);
    return 0;
}
```

Here the code checks that the argument that has been entered only contains integer digits by first finding the length of the argument as a string, counting all digit chars in said string, and comparing the two numbers. If they are not equal, i.e. there are more characters than digits, then the program will terminate with an error message. This is the only time the input is checked for incorrect input, but this is fine because the program will terminate before any code runs that is dependent on the input being sanitized.

Now the user input as a command line argument has been checked for unsupported characters, the rest of the code may be called from main().

```
// call functions to handle the printing of each zone
top_zone(argv[1], intLen);
mid_zone(argv[1], intLen);
bot_zone(argv[1], intLen);
```

These function calls send the number to display as a string in argv[1], and the length of the number to the 3 functions that handle different zones in the display. The length is sent so strlen() only has to be called the one time in main() and never again.

Functions top_zone(), mid_zone(), and bot_zone() are all functionally identical, with the only difference being which helper function is called from within. Naturally, top_zone() calls top_zone_num() and so on, with top_zone_num() handling the printing of the top zone for 1 number at a time. Here is only an example of top_zone(), but mid_zone() and bot_zone() only differ in calling mid_zone_num() and bot_zone_num(), respectively.

```c
/*
 * Purpose: prints all segments for the top zone by calling top_zone_num for
 *          each digit
 * userInt: full number user entered as a command line argument
 */
void top_zone(char* userInt, int intLen) {
    // prints full userInt
    // printf("top: %s\n", userInt);

    // iterate over top zone and print every digit's segments one at a time
    for (int i = 0; i < intLen; i++) {
        top_zone_num(userInt[i]);
    } printf("\n");
}
```

Each zone function uses the same for loop and intLen variable to iterate over the number and print out the applicable zone for each number.

```c
/*
 * Purpose: prints the correct segments for 1 digit in the middle zone
 * thisDigit: current digit being partially printed
 */
void mid_zone_num(char thisDigit) {
    // print current digit being displayecd
    // printf("%c", thisDigit);

    // match a given ASCII digit with the correct segments and print it
    if (thisDigit == '0') {
        printf("| |");
    } else if (thisDigit == '1' || thisDigit == '7') {
        printf("  |");
    } else if (thisDigit == '2' || thisDigit == '3') {
        printf(" _|");
    } else if (thisDigit == '4' || thisDigit == '8' || thisDigit == '9') {
        printf("|_|");
    } else {
        printf("|_ ");
    }
}
```

Shown above is my implementation for mid_zone_num(), which prints the middle zone of a number. The top_zone_num() and bot_zone_num() functions are structured similarly, except with their if-else trees modified to fit the different zones. Variable thisDigit is the digit of the inputted number being currently printed, and is checked checked against all possible middle zones to determine which combination of space, underscore, and vertical bar should be printed. Initially, I had intended to use a switch statement for the different cases of characters to print, but that implementation ended up

producing unintended behavior. Making this if-else tree for every digit fixed the unintended behavior. It is not pretty or elegant, but it gets the job done.

Testing

Testing involves running "./display7 1234567890" from terminal in the same directory the program is in.

Additional test cases, such as "./display7 12345a67890" and "./display7 1234567890 a" were also run to test input sanitation.

Results

```
matthew@mdesk-popos:~/Documents/CISS247/assignment3$ ./display7 1234567890
   _  _     _  _  _  _  _  _
  | _| _||_||_ |_   ||_||_|| |
  ||_  _|  | _||_|  ||_| _||_|
matthew@mdesk-popos:~/Documents/CISS247/assignment3$ ./display7 12345a67890
Please enter a positive number instead of "12345a67890".
matthew@mdesk-popos:~/Documents/CISS247/assignment3$ ./display7 1234567890 a
Error. Check arguments.
Usage: ./display7.c 123456789
matthew@mdesk-popos:~/Documents/CISS247/assignment3$ ./display7 1234567890 a^C
```

Conclusions

Given the coding conventions laid out on Canvas, and focusing on only one function at a time, made this assignment relatively simple to complete. Requiring a function for each zone and each number meant I did not have to design the overall structure of the program, instead I could focus on implementation. This reduced the amount of work significantly, making this assignment the easiest to implement, especially being at the end of the quarter and having all previous coding assignments to look back at. If I felt needed the extra points, then the additional functionality stipulated in the extra credit section looks entirely possible to implement; I just did not feel that was necessary, especially when I had the core functionality working 100% as expected. Overall, this assignment was a nice way to cap off the quarter, I only wish I had gotten the switch statements working like I wanted for printing.