1. **Introduction / Purpose / Intent**

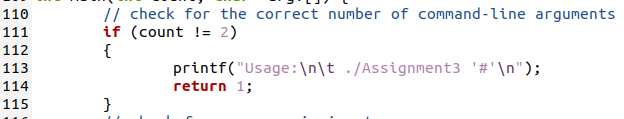
For this assignment I was tasked to create a C program to display an integer number in the style of a seven segment digital display, using only the space, underscore, and vertical bar characters. The design of each of the digits in this manner was provided to me at the beginning of the lecture page on canvas. The program is to take an integer as an argument at the command line when the program is run. It is required for the program to check that this command line argument is present, and that it is an integer number. In the event that the argument is not present or the input is not an integer number, the program is to terminate with an error message. There are 2 optional extensions to the exercise; to have it able to handle negative integers and to enable scaling of the display.

The assignment page on canvas details a possible solution to this task by separating each digit, in digital form, into 3 zones consisting of top, middle and bottom. It is required to produce and utilize 6 functions in addition to the main() function. These functions are to display one or all of the digits in a number for each of the 3 zones. The coding standards I am required to abide by are; to use meaningful names that give the reader a clue as to the purpose of the variable or function being names, avoid the repeated use of numeric constants, use comments at the start of the program to identify the purpose of the program along with the author and date it was written, use comments at the start of each function to describe the purpose of the procedure and the purpose of each parameter to the procedure, use comments at the start of each section of the program to explain what that part of the program does, and to utilize consistent indentation.

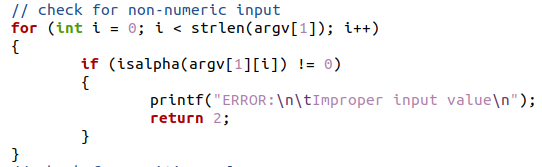
1. **Process**

The first step in this lab was to lay out the shell of my program. I knew that I would be utilizing inputs in main, so I drafted the main() function with an argv and argc variable, as well as copy and pasted the required functions from the canvas assignment page as comments outside of main(). I created the 6 function declarations each named for the number of digits it would display, one or all, and for which section it would be working, top, middle, or bottom. These functions all were declared as void as they would not need to return a value. Initially I left them accepting no value either and turned to work on my requirements in main.

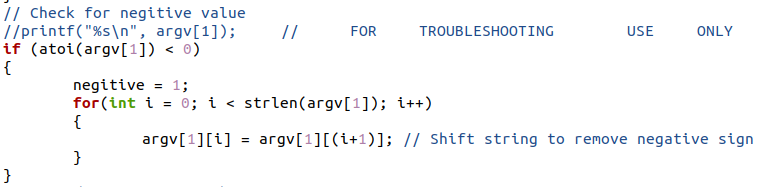
One of the first and easiest things to achieve was to check for the correct number of command line arguments. I opened an older assignment I had written and completed and copied the “if” statement at the beginning of its main. This statement simply checks to see the number of arguments present at the time the program runs. If that number is not 2, meaning a call to the program and an input value, it produces an error message and terminates the program. Very little editing of the code was required, just to edit the printf() statement to display the correct error message.



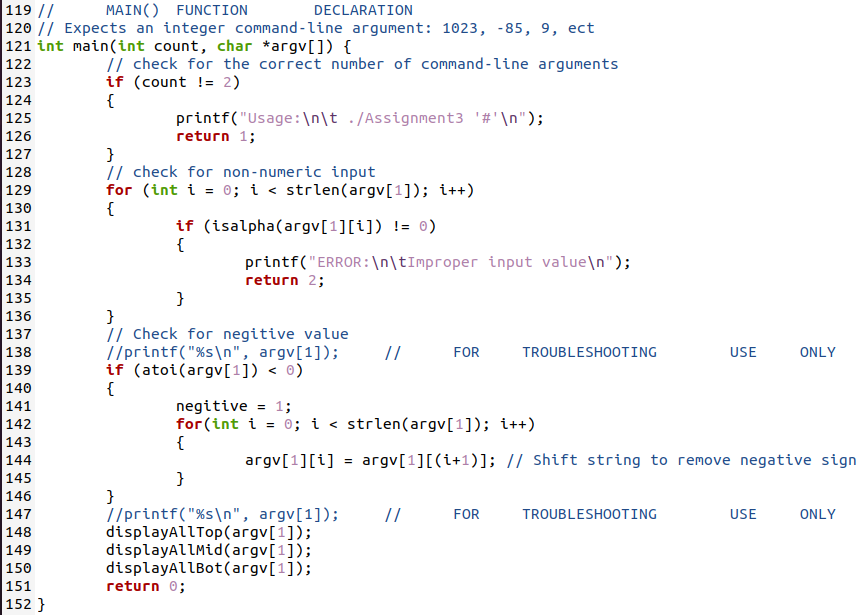
Following this check for the correct number of arguments, I was going to check for the valid input. Initially I was going to utilize the function isdigit() from the ctype.h library, but quickly I found that it only returns a 1 for values greater than 0. This meant that the input value 0 and the input value for any alphabetical character would be 0. This wouldn’t work for what I needed to accomplish. Within the same library I found the function isalpha(), which returns 0 for any values that is not alphabetic characters. I constructed a for loop with a nested if statement to check character by character of the input string. If the isalpha() function returned anything that was not a 0 it would terminate the program and produce an error message as instructed. This worked to my advantage because I wished to attempt to handle negative values, and the negative symbol is not an alphabetical character. This meant that the return value for “-“ was the same as any numeric digit, and would operate in the same way.



Next I wanted to check the input for negative values by utilizing the atoi() function in an if statement. If the value of the input was less than zero one it was turned into an integer by atoi(), then I would print to the command line a negative symbol and remove the negative sign. To do this I constructed a “for” loop that marched through the input string and moved each character in the array up one slot. This would overwrite the negative sign in the input so I could work with a simple string of numbers for the rest of the program. Later I realized that a single negative sign would not suffice as input here, so I decided to create a global variable that would be set to 0. If ever a negative value was input, this variable would be changed to 1. Later in the program I would test if this variable was still equal to zero, and if not it would print in the negative sign in correct format of digital display. The negative value needed to be a global variable, as I would be checking the value of the variable inside 3 of the 6 functions to output the sign in the correct format to the terminal. I wanted the sign to be outside of the loops that would produce each of the digits outputs, and so it ended up inside each of the displayAllX() functions. This ensured that the symbol would be output in the proper location and in the proper format.

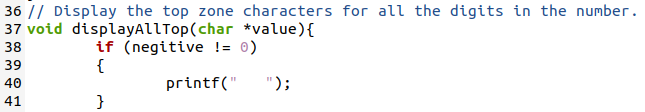


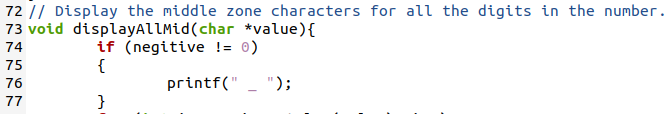
Now that each of the checks had been complete in the beginning of main, I would work on the 6 functions. However before I left main I placed 3 calls to the displayAllX() functions for each of the sections in the order they should be print. I also provided argv[1] as their input, which shaped how I would develop the function definitions. With these calls to the displayAllX() functions, main() was complete and required no further tweaking.

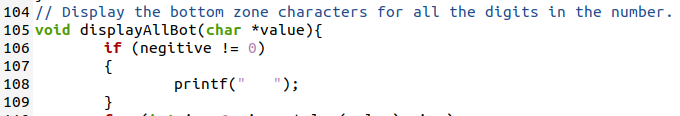


I began to tackle the functions by copy and pasting the format they would be printed in from the assignment page into my program as a comment. This would give me a visual reference as to the shape and design of each of the digits for the remainder of the writing of the program. I edited the declaration of each of the functions to accept “char \*value” as input, leaving their return type as void because no manipulation would occur. As said above, I had placed within each of the 3 displayAllX() functions if statements to check the value of the negative global variable. Since only one call to each of these 3 functions would exist, theoretically only a single negative sign could exist as well.

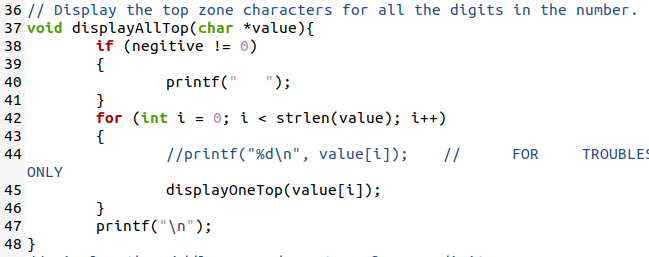


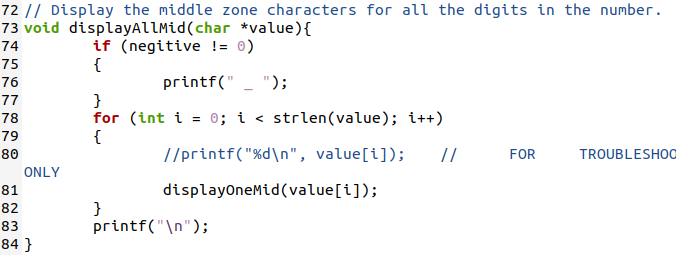


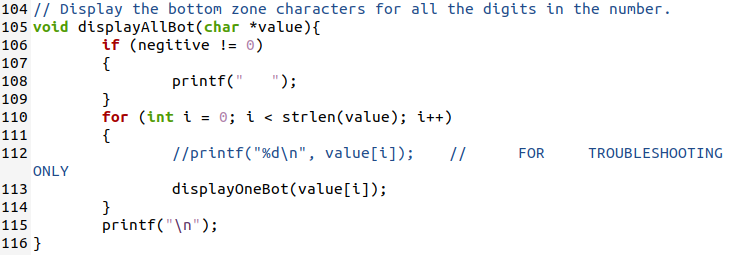




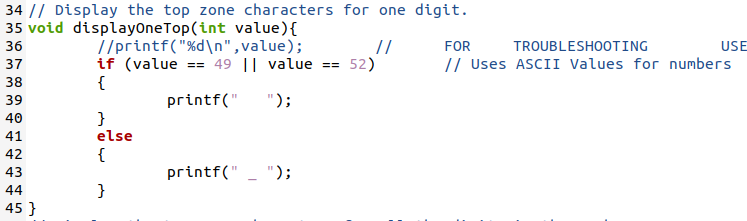
Since the input of the digits needed to be analyzed both together and separately to produce the desired design, I had the displayAllX() functions call their respective displayOneX() function in a for loop, which would step through the string array for each of the digits and use that digit as input for the displayOneX() function. The displayAllX() functions would also have a single printf(“\n”) statement for formatting after the loop had complete. In this way, the displayAllTop() function would begin, and digit by digit print out what was required for the top sections of those digits. Then once the loop had analyzed all of the digits, it would print a new line for the next section to begin the same process. The displayAllX() functions are almost identical, changing only the value printed if the negative variable is on and the call to the displayOneX() function that corresponds to the displayAllX() section.



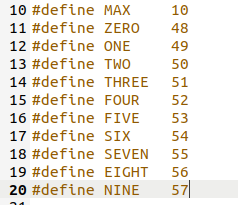


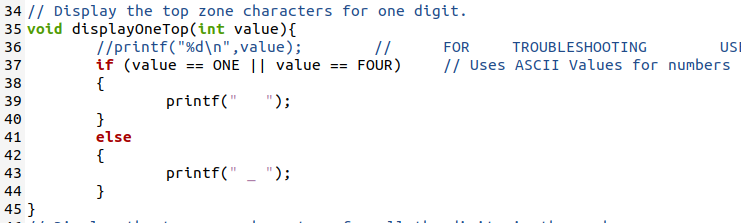


What I didn’t know when designing the displayAllX() functions is that when the digits are access individually, their ASCII integer values are passed rather than the correlating numeric character. This made things a bit messy when troubleshooting with my printf() statements, but will be discussed later in this report. I had to change the input for the 3 displayOneX() functions to “int value” and interpret the number. For the top section, there were only 2 potential values of formatting. This was accomplished with an if/else pair, where the tested if statement was the 2 values that needed no top, 1 and 4. Since the displayALLX() function would examine each digit individually, and pass it to the displayOneX() function as input one at a time, all this function needed to do was output to the terminal the correct top section of the digit. This also determined how the following 3 functions would be designed, using if/else statements for potential results.

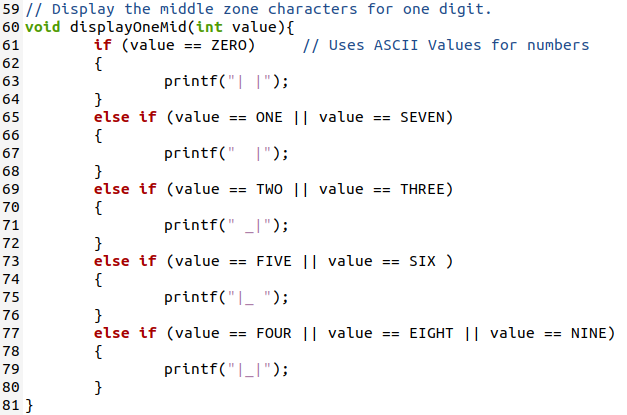


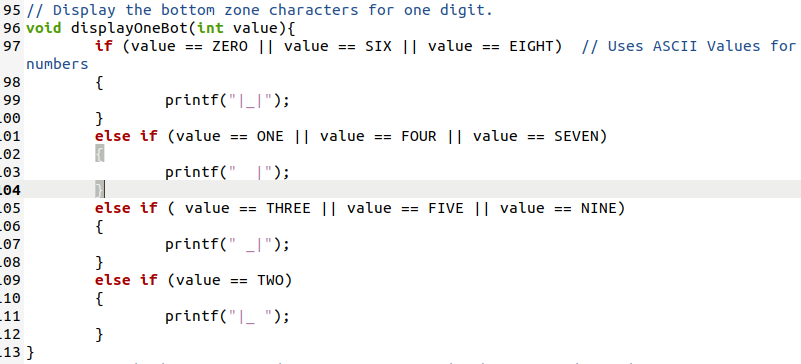
The value of the input for the displayOneX() functions was the ASCII integer value of that digit’s character, as discussed previously, so instead of comparing “value == 1” or “atoi(value) == 1” I compared the value to the corresponding ASCII value. This method does not conform to the coding style required in the assignment page that was listed in the introduction of this report, so I defined 10 statements to replace all of the uses of numeric constants. Each of the define statements holds the value of its correlating ASCII integer value. The numeric constants were replaced in the displayOneTop() function as can be seen below.





The same process used to define the displayOneTop() function was repeated for the middle and bottom displayOne functions. I generated “if” and “else if” pairs for each of the potential outputs for that section and placed a test for the value of the correlating digit against the input ASCII value passed to the function. This process took some trial and error to correctly display but ultimately was not difficult. With the generation of these functions all that was left to accomplish was the testing of the program itself.



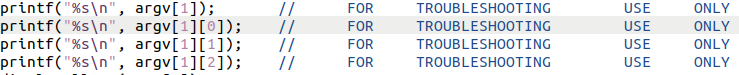


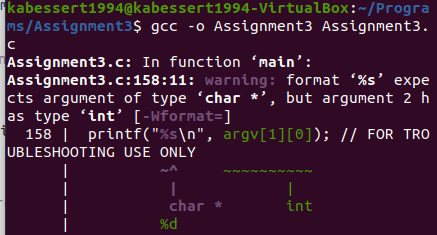
1. **Testing**

All throughout the coding process many “printf();” and “puts();” statements were used to output variables in various locations, including before and after variable assignment statements, before, in, and after loops, and in functions. These statements were used for the tracking of information and troubleshooting the overall logic and design of the program. These statements have been removed in the final submitted version of this code, although they remain in the working portion of this code.

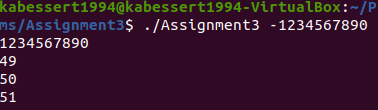


As stated earlier in this report an issue arose when I was writing and troubleshooting the displayOneX() functions from utilizing my troubleshooting printf() statements. The issue was that when the string would be printed to the terminal such as with “printf("%s\n", argv[1])” the entire string would be output correctly, however when I attempted to access each specific digit individually I would receive a warning and the code would not compile. Each digit is interpreted as an integer instead of a character or a string.

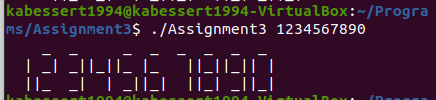


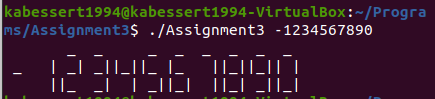


I changed the values in my printf() statements accessing each digit as instructed and the code compiled fine. I originally thought the program would now output each digit as the integer it was, in my example case I was using -123 and expected to see the digits 1, 2, and 3, however this was not the case. My logic at the time was that somehow somewhere in my code I was assigning the incorrect values into the array so that argv[1][0] now held the integer 49. I combed through my code and didn’t realize until after I had added and removed many more troubleshooting printf() statements that this logic did not make sense. It was then I discovered that these values correlated to the ASCII decimal integers of the digit’s characters. This is why atoi() did not work when accessing each individual digit, as the digit was already being displayed as an integer, and why the printf() statement required to be changed from string to digit.

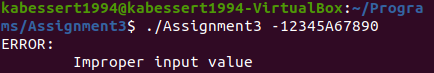


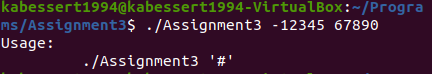
After the program was written I utilized the digit -123456789 to verify the layout of each of the digits. At first it was a chaotic mess of incorrectly formatted characters, but I returned to my displayOneX() functions and ensured the correct values were placed in the correct if/else tests.

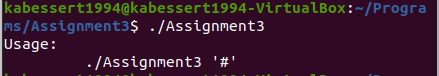


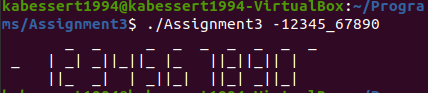


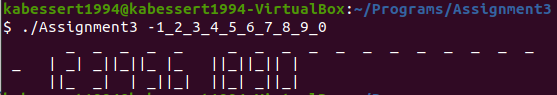
Now that all of the digits are displaying correctly and as can be seen the negative portion of the program is functioning correctly, I moved to testing different inputs. This included inputs with alphabetical characters, multiple inputs, no inputs, and inputs with strange characters inside. I found that the only condition that did not perform as expected was the introduction of non-alphabetical characters as can be seen below. At this time of testing the displayOneTop() function only checked for a value of 1 or 4 to place 3 consecutive spaces, and in all other cases to print 2 spaces with an underscore between them. Since the other functions did not have an else statement alone, they did not operate at all on these types of characters while the displayOneTop() does.



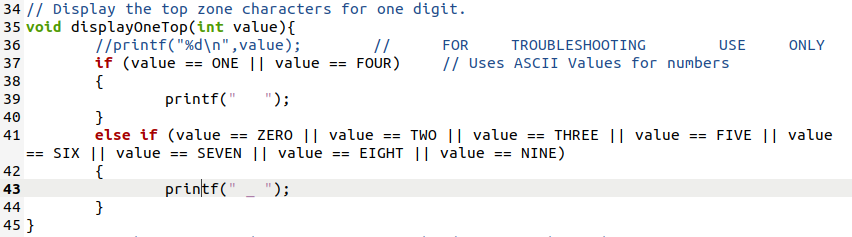


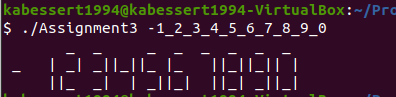






I modified the function from an “else” statement to an “else if” to test specifically for the values that required the formatting “ \_ “. With this addition the functions only display the numeric digits “0-9” as intended, ignoring all other input. This erroneous input is not caught by the test in main, as described in process, due to it only testing for the presence of alphabetical characters.





1. **Results**

The results of this program are successful in accomplishing the task to display each of the numeric digits 0-9 as digital 7 segment integers. The digits are each correctly formatted and no other characters are output in this fashion. I have also successfully implemented the ability to handle negative integer inputs.

1. **Conclusions**

Based on the results and intent of this assignment I conclude that I have successfully completed the intent of this assignment to the best of my ability. I adhered to all coding standards and the required programming style. I successfully implemented the 6 functions as required and completed the bonus extension of handling negative integers. The largest takeaway from this lab was the encounter I had where I attempted to pass a numeric character in a string array to a function and it was passed as the integer value of that character rather than the character itself.

1. **References / Acknowledgements**

C Programming Language, B. W. Kernighan & D. M. Ritchie, 2nd Edition, Prentice Hall, 1988.

C Programming: A Modern Approach, K.N. King, Norton, 2008.

[Programiz C function isdigit()](https://www.programiz.com/c-programming/library-function/ctype.h/isdigit)

[Programiz C function isalpha()](https://www.programiz.com/c-programming/library-function/ctype.h/isalpha)

[ASCII Table](https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html)