1. **Introduction / Purpose / Intent**

For this assignment we were tasked to write 2 functions for the Lab2.c program provided to us, toDecimal() and fromDecimal().

The first function, toDecimal(), is to return the decimal value of a number in a specific base and accepts 2 arguments; *base*, the integer value of the number system to convert from (2, 10, 16, etc), and *number*, the number stored as characters to be converted into decimal.

The second function, fromDecimal(), is to return the string result of converting a decimal number to a specific base and accepts 3 arguments; *base*, the integer value of the number system to convert to (2, 10, 16, etc), *decimal*, the integer decimal value to convert to, and *number*, a string location to contain the resulting conversion.

These functions must be written without using any C library functions, but may use the 2 provided functions digitChar(), which returns the character for the digit with value in base, and digitValue(), which returns the decimal value of a digit character. In the Lab2.c file that was provided, each number is represented as a character string using characters ‘0’-‘9’ and ‘A/a’-‘Z/z’ to represent the decimal values of 0-9 and 10-35 respectively. digitChar() returns characters in this range and digitValue() uses this range as input arguments.

1. **Process**

The first step in this lab was to read and analyze the provided Lab2.c code to determine how the portions I could not change functioned. I began by analyzing how the main() function operated. Firstly, main() accepts 3 arguments, *base*, *num1*, and *num2*, where base is the integer value of the number system of num1 and num2, and num1 and num 2 are values in the base number system to be added together. Once executed, main() checks for these 3 arguments. Next main() converts the number base provided, stored as a char in word[1], to a decimal value stored in an integer variable **base** using the function that we will be writing, toDecimal(), with the defined variable **DECIMAL** as the argument for base. This is where I began to work the first function and writing pseudo-code.

Int DECIMAL = 10;

Char Word[1] = any#

int base = toDecimal(DECIMAL, word[1]);

Convert word[1] to its value in the target base

We were provided the function digitValue() which accomplishes this task, however only accepts a single digit and cannot accept the value stored in **DECIMAL**. This pseudo-code works if word[1], which contains our base for the addition, only contains a single character, limiting the program to bases 2-9. The new pseudo-code is generated;

Int DECIMAL = 10;

Char Word[1] = any#

int base = toDecimal(DECIMAL, word[1]);

Determine how many digits word[1] has

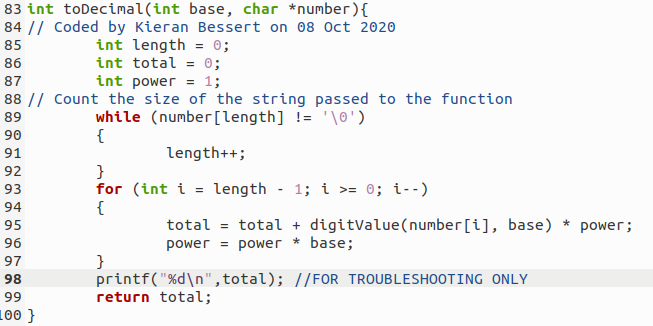
Convert each digit in word[1] individually using digitValue()

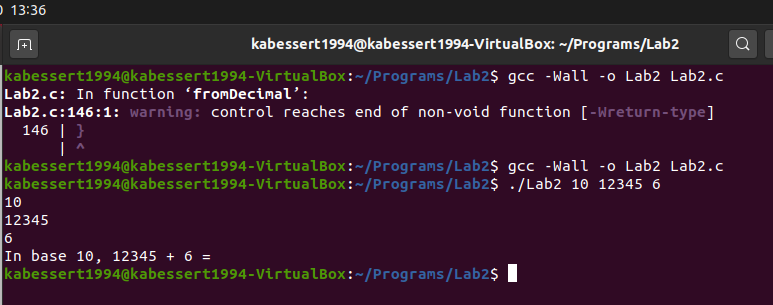
Store each digit in the proper sequence

To accomplish the first task, I declared an variable int **length** and set up a simple while loop to increment **length** once for each digit in word[1]. I chose the while loop instead of a for loop because I needed the variable length to be usable in the whole scope of the toDecimal() function.

To accomplish the second step I set up a for loop that walked through digits of word[1] and converted that character digit to an integer using the provided digitValue() function. I declared the variable int **total** to hold the results of digitValue().

The final step needed to place the value of the digit in the right position, like in the char string word[1] but as a single int. I accomplished this by declaring a new variable, int **power** = 1, multiplying the result of digitValue() by the power, adding it to **total**, and finally multiplying **power** by **base** for every digit’s position. The first iteration of this loop placed the digits in the reverse order, so I had to alter the loop to ensure the proper sequencing as shown below.





I returned to main() to continue analyzing how the program was written after the initial toDecimal() call to convert the base. Main() continues to some error checking and then moves to use toDecimal() twice more with the other input variables to be added together, word[2] and word[3] as character strings, storing them in variables **num1** and **num2**. With all of the integer variables now in base 10, main() sums the values of **num1** and **num2** together and stores it in an int variable **result**. Finally main needs to convert this result back to characters in the correct base, so it calls the final function to write fromDecimal() with the inputs of **base**, **result**, and a character string variable **text** to store the characters we will be returning with space for 20 characters. I began the next function’s pseudo-code.

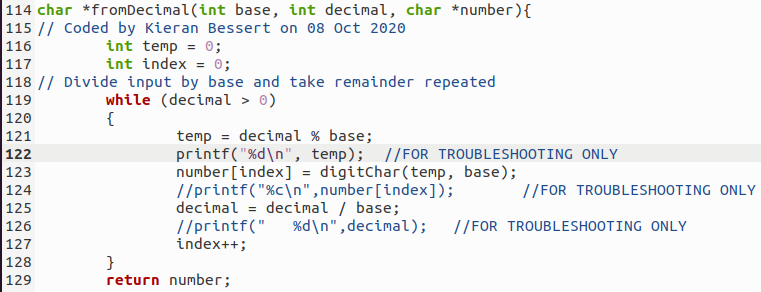
char \*answer = fromDecimal(base, result, text);

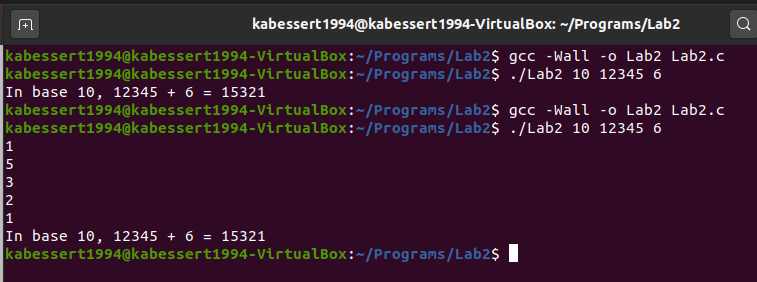
Isolate each digit

Use provided function digitChar() to convert each digit to a character

Store those digits in a string **text**[] in the appropriate position

To complete the first task of isolating each digit, I decided to inverse the power process I had used in the previous function by using the modulus operator with the **result** and **base** variables and storing the result in a new int variable **temp**. Now that the first digit was isolated I can utilize it as input for the provided function digitChar() along with **base**, storing the return of digitChar() into the first position of string **text**[] using a new variable int **index** = 0 to. I would need to loop this process for all of the digits within **result**, so I decided to divide **result** by **base** which removed the 1s position that we had already stored and made the 10s position in **result** the new 1s position. I placed the code into a while loop that tested the value of **result** > 0. This worked through all of the digits of **result** one at a time placing them in **text**[**index**], incrementing index after every loop. This method concluded successfully, however the result is in the reverse order due to the method chosen for processing the data.





1. **Testing**

All throughout the coding process many printf(); statements were used to output variables in various locations including in loops and after loops. Their use can be seen in the above images where for toDecimal() one is being used to display the **total** variable, after looping, that is being returned prior to the completion of the second function and for fromDecimal() it is displaying the value of the modulus stored in the **temp** variable before the loop continues. There are other commented out statements, as well as ones that have been removed from these screenshots through the various stages of writing and troubleshooting. All of these statements have been removed for the final version submitted along with this report.

Next was reversing the results of the returned string. To do this I wrote the following pseudo-code;

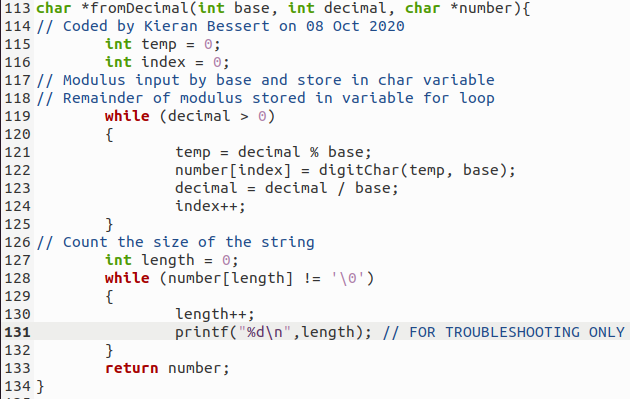
Get length of **result**

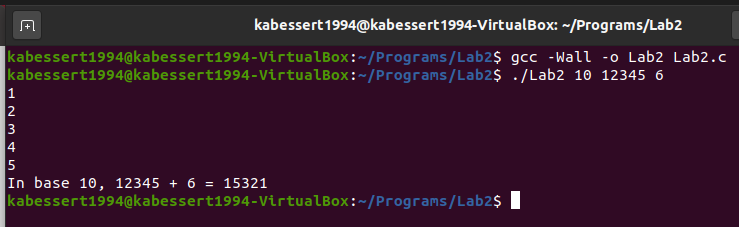
Place contents of **text**[] into a temp array in correct order

Copy temp array to **text**[]

Return **text**[]

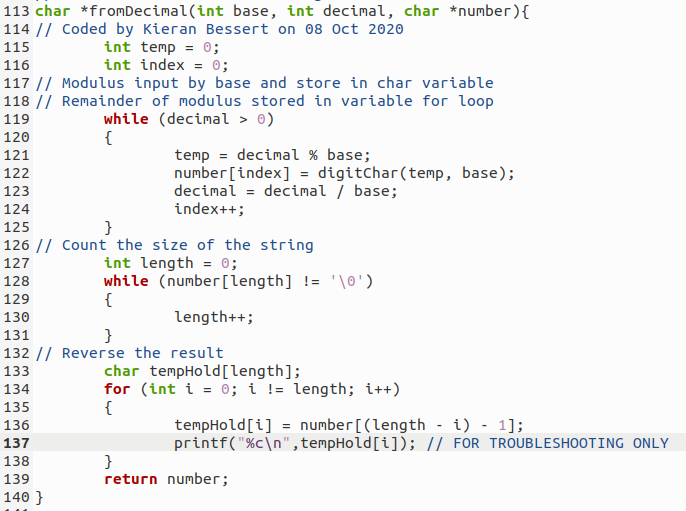
I had already written a while loop for counting string positions for toDecimal(), and at this point in the fromDecimal() function I had a string with content inside it in the form of **text**[], which held **result**, only backwards. Instead of writing new code to count the digits of the **result** variable I copied the while loop and **length** variable from toDecimal(). I then tested if it successfully counted the positions of the **text**[] string.

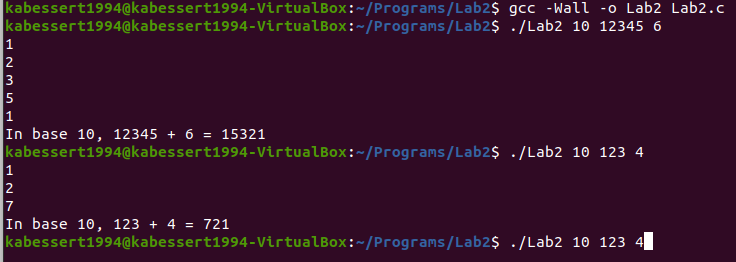




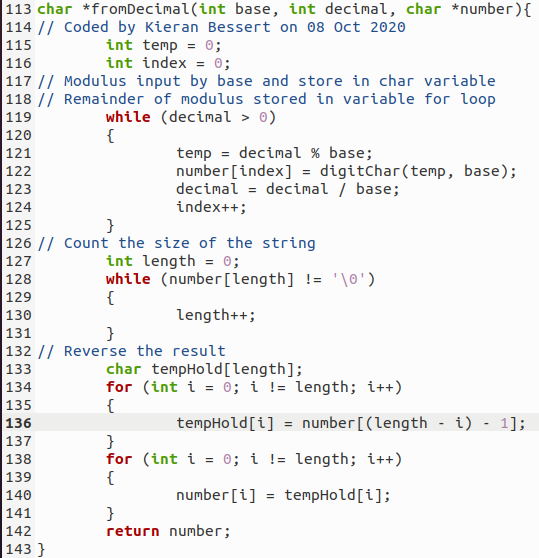
The function seemed to be functioning correctly, counting only the positions within **text**[] that held values, as **result** was stored inside **text**[] we expect length to be 5 in this case. Continued testing with variable input data lengths showed this to be affirmed.

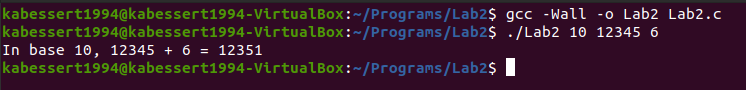
Next came reordering the contents of **text**[] in a temporary variable **tempHold**[] which used the **length** variable to set its size. I decided to use a for loop to step through the positions of **tempHold**[] and **text**[] in opposite orders so that the final character of **text**[] was placed in the first character position of **tempHold**[] and so on. Testing showered this to have succeeded.





**TempHold**[] now held **result** in the correct order, and initially I attempted to return **tempHold**[] and cut out the final line of pseudo-code before I realized that the memory location for **text**[] is what was being passed to the fromDecimal() function. I then set up a final for loop that replaced the contents of **tempHold**[] back into **text**[]. With this complete, the return value of **text**[] should contain **result** in the appropriate order, and finished out the function.



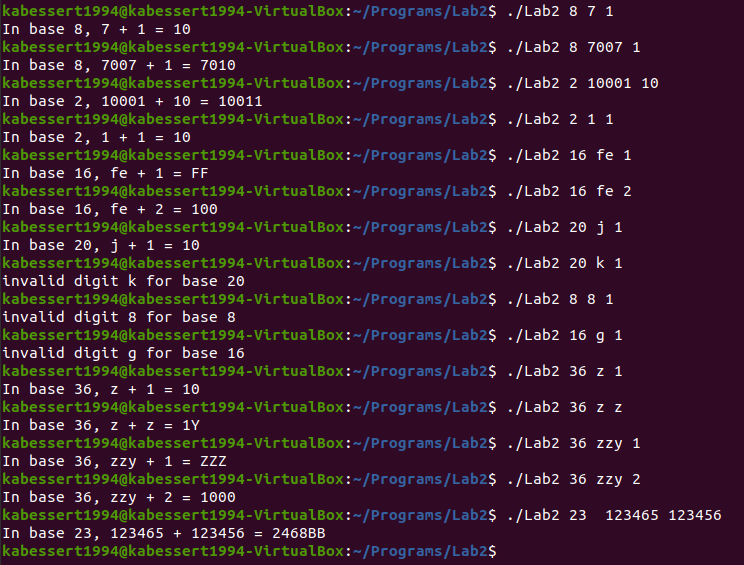


With this issue corrected, further testing of the program could continue. Up until this point I had almost exclusively been using base 10 for all of my calculations, as my thought process was that if a number system I was familiar with was incorrect then how could I expect a number system that I was not as familiar with to be correct. For follow on testing I began using bases I was less familiar with than base 10, such as 2, 8, and 16, and bases I was not familiar with, such as 20, 32, and 36. Further testing showed positive results.

I did not test the error checking of the rest of the program, as I was not responsible for coding it, however I did attempt “known bad input” and “known good input” checks to verify that values expected to be bad, such as a k in base 20, were correctly noticed and disallowed as input and values expected to be good, such as j in base 20, were correctly authorized. Testing for bases above 36 is neither desired, nor required, as the code in main() tests for valid range before any function call. Testing the addition of numbers with greater than 20 digits has been conducted and is unsatisfactory, thought the 20 digit limit is written outside of the scope of my responsibility, so further testing of this limit is neither required nor desired by the intent of the lab.

1. **Results**

The results of this lab are successful in completing the intent of Lab 2. Both functions were completed so that any numbers of 20 digits or less of any base 2-36 may be passed to the completed toDecimal() function and converted into decimal base 10, then are then return to main to be added together, then that result is used as input for the completed function fromDecimal() where the result is converted to characters and stored in the correct order. No calls to library functions were used within the limits of these functions, thought were used for troubleshooting when required. Below is screen-capture of a handful of results from this code. Included with this report is the completed lab2.c file.



1. **Conclusions**

Based on the results and intent of this assignment I conclude the intent was to familiarize me as a student with loops, functions, variable passing, C syntax, and complex problem solving for a language I am not familiar with. I had significant difficulty starting this assignment as I had not coded in about a quarter and did a lot of referencing to my books. I also know I did not code these solutions in the most efficient manner, but in a manner that was logical for me to understand that allowed me to insert troubleshooting code where I was unsure of what was happening.

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.

Douglas Slater assisted in the pseudo-code for producing a while loop that counted positions of strings during a lab session breakout room.