CMPSC 200 Final Report: Katie, Maya, and Paige

For our final project, we have used the C programming language in order to create a translator and a binary calculator. Our program can successfully translate numbers between their binary, decimal, and hexadecimal forms. This program also features a calculator that takes in decimal numbers, converts them into binary, and performs either binary addition, subtraction, multiplication, or division.

Motivation

Our motivation for this project was to utilize the knowledge we gained in class to develop a functional program. While the initial idea was to simply develop a translator, we were inspired to take it a step further and demonstrate our knowledge of mathematical operation in binary as well.

Background

A translator between binary, decimal, and hexadecimal is a simple idea that has been done many times before. For example, this website by MathIsFun.com is an example of a previous implementation of this idea. Unlike this example, we used the programming language C for our implementation. Similarly, binary calculators have been done before as well. For example, this website by Calculator.net. Again, a difference between our implementation and theirs is that we used C. We also developed a program that performs as both a translator and a calculator in one. Since these are concepts that use tested algorithms, the solutions to these problems will give the same answers when the implementation is correct.

Overview of Files:

- Main.c file executes completed functions correctly
- DecimalToBinary.h properly converts decimal number into their binary form
- DecimalToHex.h properly converts decimal number into their hexadecimal form
- BinaryToDecimal.h properly converts binary number into their decimal form
- BinaryToHex.h properly converts binary number into their hexadecimal form
- HexToDecimal.h properly converts hexadecimal number into their decimal form
- HexToBinary.h properly converts hexadecimal number into their binary form
- Add.h file adds numbers in their binary form
- Subtract.h file subtracts numbers in their binary form
- Multiply.h multiplies numbers in their binary form
- Divide.h divides numbers in their binary form

The Main File

The main file, main.c, is where all of the files are run. Main.c imports all of the calculator and translator functions, which are named as .h files. Main.c has one function, int main(). In this, the user is asked a series of prompts in order to allow the user to select the specific calculator or translator that they want. Once this occurs, the user is asked to enter the binary, decimal, or hexadecimal value(s) required for the function. The only issue I had while coding this section was making sure all of the different prompts and logic that happens as a result of the response was nested correctly. After adding the calculator section, it got a bit more complicated and confusing to me, but I was able to solve this by breaking it down and using comments to help myself with it more.

Converter Functions:

Decimal to Binary Converter

The Decimal to Binary converter, stored in DecimalToBinary.h, is functional at this point. It has one function inside of it, d2b(), which takes in a decimal number as an int parameter and finds its binary equivalent. First, this function finds the amount of bits that should be in the binary number while the bits are being constructed backwards. After this, the bits are reversed, putting them in the correct order. After this, the program checks to see if any zeroes from the end of the binary number were removed, and adds all of the zeroes back on, producing the correct value. After this, the result is returned to be printed by the user. I didn't have any major problems with this code besides getting the trailing zeroes to be added back on. Through experimenting though I was able to come up with the idea to use the pow() function to add the difference of the original binary count during creation to a reversed binary count to re-add the difference in necessary binary values.

Displayed below is an example of the user interaction and output of using the decimal to binary converter:

Binary to Decimal Converter

The Binary to Decimal converter, stored in BinaryToDecimal.h, is functional at this time. This file has one function inside of it, b2d(), that receives a binary value stored inside of an unsigned long. This function calculates the amount of values in the binary and stores that in count. Count then is decremented in a while loop until it is equal to 0. In this while loop, the value is calculated, with the result being calculated using the formula res = (res + q) * 2 each iteration until the last one, where it is only added with q. After the loop is ended, the result is returned to be used or printed out to the user. I had no major issues with this code, and had an easy time converting it from a calculation done by hand to code.

Displayed below is an example of the user interaction and output from using the binary to decimal converter:

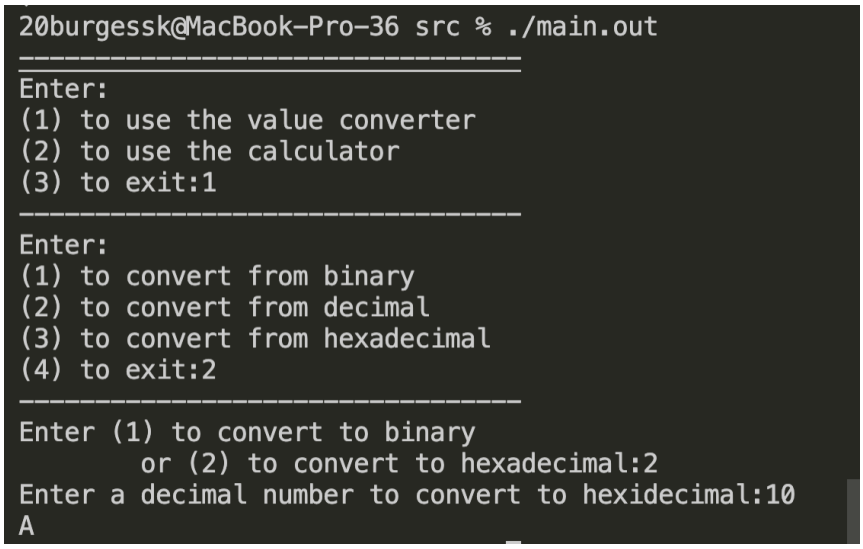
```
20burgessk@MacBook-Pro-36 src % ./main.out
---------------------------------
Enter:
(1) to use the value converter
(2) to use the calculator
(3) to exit:1
---------------------------------
Enter:
(1) to convert from binary
(2) to convert from decimal
(3) to convert from hexadecimal
(4) to exit:1
---------------------------------
Enter (1) to convert to decimal or (2) to convert to hexadecimal:1
Enter a binary number to convert to decimal:1010
10
```

Decimal to Hexadecimal Converter

The Decimal to Hex converter, stored in DecimalToHex.h, is functional at this time. This file has one function inside of it, d2h(). The user is prompted to enter a decimal number inside of the function. This function then goes into a while loop, which is run while the decimal value is larger than 0. Every iteration, the number is modded 16 to get the remainder, and then divided by

16 in an integer to get the value of the division without the remainder or a decimal value attached. If the remainder is larger than 9, it is converted to the hex letter equivalent. Once this process is completed, the hex value is reversed to put it in the correct order and stored in the output array. This array is then printed in an if else nested inside of a for loop, in order to allow the character values to be printed using %c and the decimals to be printed using %d. I only had problems with the output printing, as the correct algorithm in order to get each value given to the correct print statement was a bit challenging for me to come up with at first.

Displayed below is an example of the user interaction and output from using the decimal to hexadecimal converter:



Hexadecimal to Decimal converter

The Hex to Decimal converter, stored in HexToDecimal.h, is functional at this time. This file has one function inside of it. h2d(). This function is very similar to the Decimal to Hex converter. Instead of going from decimal to letter values, this function inputs a Hex value, in which each value is checked to see if it is above 9 or not. If it is, that index in the array is converted to its corresponding integer value. After this, the mathematical function is applied, in which each number is added to the result and then multiplied by 16, except for the last iteration,

where it is just added to the total. This result is then returned to then be used or printed to the user. I didn't have any major problems with this section, only with finding the length of the array as well as creating an appropriate length for the hex to be inputted into.

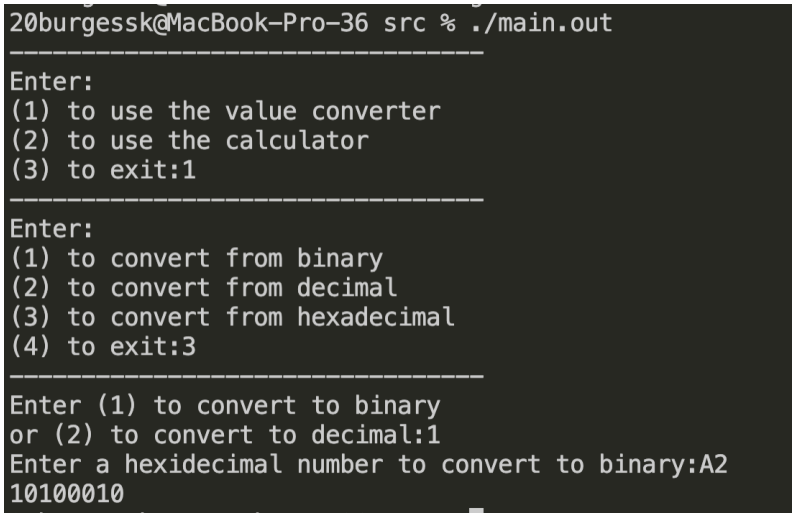
Displayed below is an example of the user interaction and output from using the hexadecimal to decimal converter:

```
20burgessk@MacBook-Pro-36 src % ./main.out
--------------------------------
Enter:
(1) to use the value converter
(2) to use the calculator
(3) to exit:1
--------------------------------
Enter:
(1) to convert from binary
(2) to convert from decimal
(3) to convert from hexadecimal
(4) to exit:3
--------------------------------
Enter (1) to convert to binary
or (2) to convert to decimal:2
Enter a hexidecimal number to convert to decimal:A2E
2606
```

Hexadecimal to Binary

The hexadecimal to binary converter starts by generating an empty string (list of chars) with a size given as a global variable of length 10. The same is performed with a binary array of size length * 4. The user can then enter a hexadecimal to be converted which is stored in the empty hex string. A variable len captures the length of the string and is used to iterate through that string. The various if statements match the hexadecimal character to its decimal representation. The final if statement (the else) deals with the case where the hexadecimal is not a character. Next it will send the list of decimals to the decimal to binary converter to get the final result.

Displayed below is an example of the user interaction and output from using the hexadecimal to binary converter:



Binary to Hexadecimal

This function has been fully implemented and tested now and correctly converts binary values into their hexadecimal form. The function first takes user input and stores it into an array. It then breaks this array down into groups of four bits each, temporarily storing them into an array of size four. The groups of four are then extracted from the arrays. Then, the previously defined b2d function is called to convert these 4-bit binary numbers into decimal form before they are stored in the hexadecimal array. Once this is finished, the function iterates through the hexadecimal array, converting the decimal to its corresponding letter where appropriate, and printing the values one by one on the same line.

Displayed below is an example of the user interaction and output from using the binary to hexadecimal converter:

```
20burgessk@MacBook-Pro-36 src % ./main.out
---------------------------------
Enter:
(1) to use the value converter
(2) to use the calculator
(3) to exit:1
---------------------------------
Enter:
(1) to convert from binary
(2) to convert from decimal
(3) to convert from hexadecimal
(4) to exit:1
---------------------------------
Enter (1) to convert to decimal or (2) to convert to hexadecimal:2
Enter a binary number to convert to hexidecimal:1111
F
```

Calculator Functions:

Add and Subtract

The implementation of the add and subtract functions, which are found in Add.h and Subtract.h respectively, have been fully completed and tested. When the user runs the main file and selects either of these options, they are prompted to enter the numbers in decimal form which theory would like to add or subtract. The main file then calls the d2b() function that was implemented in DecimalToBinary.h in order to convert the entered numbers into their binary forms. These binary numbers are then passed to the respective function, whether add or subtract was selected. The functions both take in the binary numbers and add each individual bit into an array from right to left. The add function then applies the logic to add each bit from right to left, carrying when necessary. The subtract function behaves similarly, but it will first find the two's complement of the second value before performing the addition. The functions both print what the subtraction looks like in binary form, then pass the result back to the main file where it is converted back into decimal form.

Multiplication:

This function takes as input the two unsigned longs from the main method. By using two while loops, it continuously performs the modulo operation to capture each bit into the existing

array of 16 bits starting from the end of the array. While this happens, a counter is implemented in order to measure how many bits will actually be required after the translation. These counters will also be utilized later in the file.

The next section of code runs the actual multiplication algorithm. Because the algorithm needs to execute until the multiplier is all 0s, the while loop runs until multiplier counter = 0. The first if statement checks if the last bit in the multiplier is equal to 1. If so, the multiplicand and product are added together. As of now, there is a separate method for add in multiplication that only deals with the inputs and does not have a separate array for output. That is because the second input itself must be modified.

Next, the multiplicand will be left shifted by 1 and the multiplier will be right shifted by 1. This code works by creating a temporary variable and switching the values in the list. The main difference between the right and left shift is that the right shift can go out of bounds. Therefore, an if statement is implemented to check if that is the case and deal with it accordingly. Finally, the multiplier counter is decremented and the process is repeated again.

Division:

The division function works in a similar way as the multiplication. However, it was much more difficult to implement given the fact that the decimals could not be converted into a list with a set size of 16. Thus, the program uses dynamic arrays. It also requires shifting two lists together and subtraction (which also requires one's complement and two's complement).

The main part of the code starts by counting the number of bits required when translating the unsigned long into the binary list. Next, four dynamic arrays are created based on the counter variables: The dividend uses size = dividend counter and the divisor, quotient and negative divisor use size = divisor counter. These specifications are based on the division algorithm.

Because subtraction will need to take place, the one's complement and two's complement will be performed on the multiplier. The main change made in these two methods comes from the fact that the multiplier list is dynamic. Therefore, a counter variable that specifies the length of the multiplier must be used instead of the global variable we used to use called size.

After computing the negative multiplier, the main division algorithm begins. The algorithm specifies that the computation will continue until the counter which is equal to the length of the divisor. The first step in the algorithm is to left shift the quotient and multiplicand by 1 together. This method took a bit of maneuvering. In short, it sets the size equal to the combined length of the quotient and multiplicand and literature that many times. The three if conditions break up the 'combined' arrays into three parts: the first is to shift quotient by 1, the second is to transfer the first bit in dividend to the last bit in quotient and the third left shifts the dividend.

Next, the multiplier is subtracted from the quotient. If the first bit in quotient is equal to 1, it sets the last bit in dividend to 0 and adds the divisor to the quotient. If not, it just sets the last bit in dividend to 1. Finally, the dividend count is decreased by 1. The list is then converted from binary to decimal.

Second driver file

The second driver file, which was supposed to be created using a randomized input file, which would be used to create an output file that can be used to check the correctness of the functions. This second driver file was very rewarding to work on, but due to the project due date quickly approaching and multiple other projects and papers due at the same time, this second driver file was not completed. If there was more time I would finish it, but due to the time crunch it unfortunately had to be cut.

Links

MathIsFun.com Translator Example:

https://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html

https://www.calculator.net/binary-calculator.html