Sam Dinkelman

10/12/18

CIS247

Lab 2 Report

Introduction:

The intent of this lab was to add two numbers together that are in the same base. The purpose of the first function we have to write for this lab were to turn a number in a different base into a number in base ten or decimal base. The purpose of the second function we have to write for this lab was to convert a decimal number in base ten into any other base.

Process:

For this lab I started with the toDecimal function and wrote the code below to accomplish its task of converting a number in another base into the decimal base.

I began thinking about how to accomplish this task and started with creating a for loop to iterate(i) through the pointer array of character we would pass to it.

for ( i = 0; i < strlen(number); i++){

I figured that while we were counting up through this array doing our calculations we would have to be decrementing through the powers at each position in the array to perform calculations. So I set a variable named **a** equal to the string length(strlen) and inside my for loop I would decrement that value by one every time the loop iterated.

**a =strlen(number)**

for (i = 0; i < strlen(number); i++){

**a--;**

This will pass the value thats stored in the pointer array at index position **i** and the base we enter when the program is run into the digitValue function. Which was previously written and returns the digit value of the number pass into it. Then I store this value in a variable named **c** to later do calculations on.

c = digitValue(number[i], base);

The next line of code starts the computations by multiplying the variable **c** which holds what digitValue returned by **base** to the power of **a** and stores that computed value in j for later calculations.

j = c \* (power(base, a));

When I was first writing this program I started off importing the math.h library to use the function pow to compute the power. Since this program specified that we cannot use any library functions, with the exception of string length(strlen), I created a power function so that I could use it to compute the values. I’m going to include my whole power function below and not go into too much detail explaining it.

int power(int x, int n){

int i, result = 1;

for(i = 0; i < n; ++i){

result \*= x;

}

return result;

}

These next two lines contain an if statement that has a condition that says when **a** is not equal to -1 add the variable **j** to **z** and then store the result of that back into **z**. This makes it so that the program will compute the first value of a number in its given base, compute it to decimal and then add it into a new variable **z** and then add the value for the next element that gets passed into the array into **z**. We then return **z** at the end of this function.

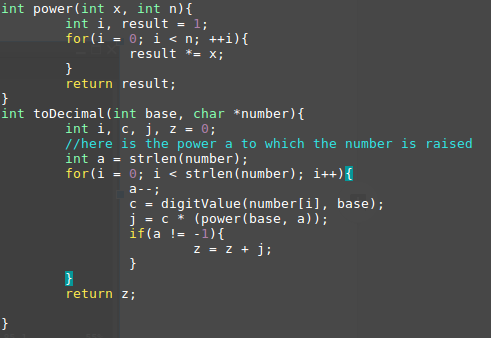
if(a != -1){

z = z + j;

}

The reason for having a condition with a not equal to -1 is since **a** is decrementing I want it to reach the last number in the string of number which would be stored as the zeroth element of the pointer array. Admittedly I could have also done an if statement that says while **a** is greater than or equal to zero do this computation and the two are logically the same when running the program. But this worked and I wrote it first so I kept it.

Here is my toDecimal function in full:



Okay fromDecimal was the next function we were tasked with writing and it was interesting to begin thinking about. The first time I tried this function I stored the computed values into the pointer array number but realized I would have to return this value later and so I had to change that and create a new array to pass the computed values into. So my function starts with some variable declarations and my array declaration which I set to be the value of WIDTH since that was defined at 20 and should be the longest string it gets passed.

Int r, i = 0;

Char newAR[WIDTH]

Next up I wanted to have a while loop since I tried it with a for loop and I had a bunch of issues, I’m sure the function could be completed this way but I preferred the while loop. So my while loop with run as long as decimal is not equal to zero. I set it this way so that when the value computes and divides the decimal by the base it will stop this while loop when there is no decimal value left to compute.

while(decimal != 0){

After this I pass digitValue the result of decimal mod base for its parameter value and then the base for its parameter base. So it does the converting and returns a character for the digit value in the given base. I then pass the result of this into **r**.

r = digitChar(decimal % base, base);

Then I compute the value of decimal divided by the base and store that result back into decimal. This is because we don’t really care about getting the value of decimal what we really need is that mod decimal by base so that we store the correct decimal value. This computation is only so that when the loop runs again we aren’t doing computations on the same value over and over again.

Decimal = decimal / base;

Next I take my new array, aptly named **newAR**, that I created and pass into it the value of **r** at the index position **i** of the new array. I do this because the value computed in **r** stores the value of the remainder and if I go through one number of the string at a time and push it onto the end of this new array I will have the correct value in decimal from any base.

newAR[i] = r;

And then lastly for this while loop I iterate **i** so that the next time it runs through the while loop it stores **r** at the next index of the new array.

I++;

After my while loop is finished I have to close off the array with a null terminating byte ‘\0’ so that it doesn’t overflow into memory when I reverse it next. This also has the effect of closing the array if the while loop never runs.

newAR[i] = ‘\0’

Now I have some more variable declarations that are going to be for reversing the string.

Int k = 0, z;

Next is a for loop for reversing the string so I set **z** equal to the string length(strlen) of the new array(newAR) minus one because of the null terminating byte I placed onto the end of the array. For the condition statement I said when **z** is greater than or equal to zero, subtract one from **z**. So this will work through the array(newAR) backwards to zero.

for(z = (strlen(newAR)-1); z >= 0; z--){

Here I set the pointer array number at index position **k** equal to the new array(newAR) at index position **z**. Thereby reversing the arrays into each other, as **k** counts up through its array number **z** counts down through its array newAR and it stores the value at each index position into each other.

Number[k] = newAR[z];

Then after this I update the index position of **k**.

k++;

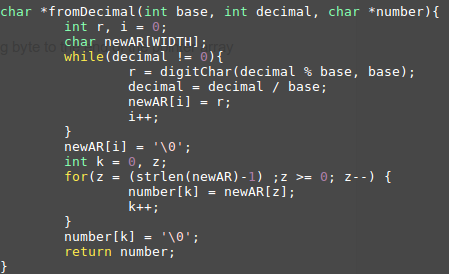
I close this for loop and then add a null terminating byte to the end of the pointer array number at index position **k**.

Number[k] = ‘\0’;

Lastly I return number.

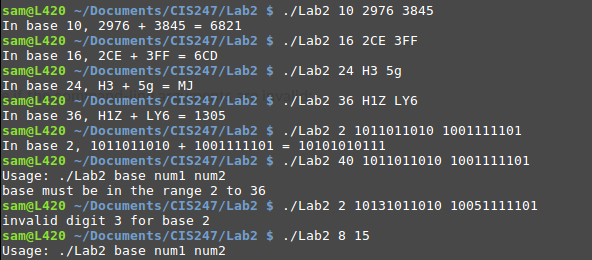
Return number;

Here is my fromDecimal code in full:

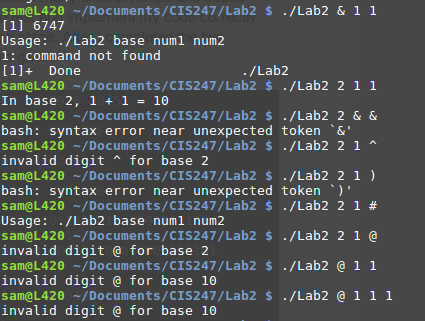


Testing:

For this Lab I did extensive testing throughout. I tested many different loops and added print statements just about everywhere in order to test and implement my code correctly so I could understand what was happening at each stage. After I completed the two functions I tested my code with all of the examples given.



I also tested entering symbols instead of numbers

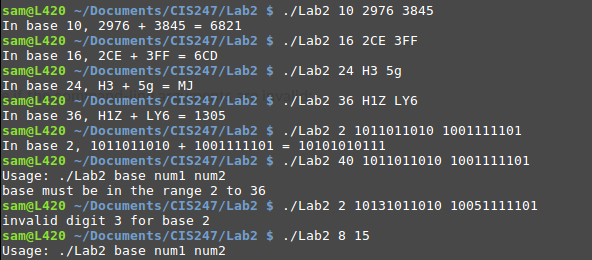


Here I actually got some interesting results the first one I used & symbol which will try and run the program as a background process in bash so that got a strange error but nothing unordinary. Most of these just told me the usage error message or told me it was an invalid digit I was entering. It was interesting when I entered @ for the base value it thought it was in base 10 but it still threw the error invalid digit message at me.

Overall the testing didn’t provide any bugs or errors except the messages written within the program.

Results:

My results from this lab show that the program I built worked as outlined in the instructions and stood up to very rigorous testing. I have run it against all of the examples given in the instructions and many other stress tests. The following screenshot shows the results of testing it against the scenarios provided:



Conclusion:

Overall I found this lab to be a good exercise to begin to think like a programmer. I enjoyed the use of arrays and having to manipulate them in order to calculate and store values. A lot was learned over the course of this lab and I feel as though I have a much deeper understanding of loops and arrays as well as functions and function calls. With this lab I really felt like I began to understand how to break up these larger problems into smaller ones that I can solve and piece together to accomplish a larger goal.

References and acknowledgements:

During this lab I worked with many of my peers to conceptualize the process that this lab required. They also provided me with one site courtesy of Trinity College CS department which breaks down the mathematical steps for converting to and from decimal bases, which will be cited below.

Morelli, R., & Lake, P. (2012, Spring). Conversions. Retrieved October 12, 2018, from http://www.cs.trincoll.edu/~ram/cpsc110/inclass/conversions.html