CALC

Does addition, subtraction, and multiplication:

- **Addition:** Takes in 4 parameters, ARG1, ARG2, SignOfARG1, SignOfARG2
    - SignOfARG1, SignOfARG2 keep track of whether the inputs ARG1 and ARG2 are negative or not
    - Run Time: O(1) since all it does is add values
- **Subtraction and Multiplication:**
    - Does the same checks as the add function.
    - Run Time: Both O(1) since all they do is compute values

The way I designed my calc program is that it converts both inputs to their respective int which then is converted to decimal.

All computation is done on the converted decimal number. Once that is done there it looks at the output base and converts the decimal to the desired output.

So its input characters -> ints -> decimal -> calculations -> conversion to desired base.

Challenges I ran into:

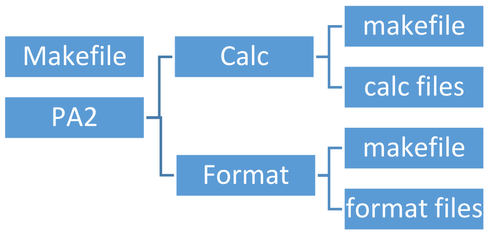
- Conversions from decimal to hex and binary, as well as output for hex and binary. My program is able to convert small numbers to binary without problem, but converting higher numbers would give me an error or the wrong output.
- Another problem was outputting the result as a string. To resolve this I added a function that takes the int result **intToChar** that takes in an int gets the size of it and takes each digit through using mod and prints it as a character. But it printed as many characters as its size. A **trim** function is called before it returns that trims all the extra leading characters. **Runtime for intToChar: O(n^2)** because there are two while loops that both run up until it reaches the size of the input value. **Runtime for trim: O(n) worst case** since it must go through each character in the string.
- **Efficiency:** Most functions run in **O(n)** time for small values. The function **toBin** runs in **O(n)** time for values less than a 1000. But this is not practical for values greater than a 1000 since it takes longer, so once the value goes past a 1000, it switches to a different faster algorithm that uses bitwise operations.
- **Calculations with different bases:** I found it easier to convert both arguments to the same base (in this case to decimal) and converting decimal to the desired output base.
- **Representing Negative Numbers:** In order to represent negative numbers, my **Add, Sub, Multiply** functions do a check that if the resultant value is less than 0, multiply it by -1, and print the minus sign (-) on the same line. So, if the output is -2 and the output base is binary then it would be outputted as "**-b11**"

FORMAT:

It is incomplete. It does the basic error checking. It is able to convert from 32 bit binary to an unsigned integer. Although, the output may or may not be correct depending on the input. Runtime efficiency **toInt** runs in O (n^2) since it goes through the binary string multiple times. Space efficiency is O(1) since it's only allocating space for a 32 bit binary integer.

Makefile

- **I kept the original folder structure with an extra file so in the root of Pa2 there is a makefile that ONLY calls the makefiles from calc and format. In other words the makefile does:**
    - **cd calc: make**
    - **cd format: make**
- **This makefile does not have a CLEAN but the makefiles in each directory has a CLEAN call.**


Makefile
PA2
Calc
makefile
calc files
Format
makefile
format files