Javed Ahamed & Mark Finkelstein
Project #1 Design Document

Design Description:

Shared between Lmult and Ldiv:

Size (num length): number number -> number
This function returns the number of digits in a positive number **num**, while using **length** to hold this number until it is returned.

Ten-Exp (power): number -> number
This function returns 10^**power**.

Ten-Exp-Helper (power multiple): number number -> number
This function is a tail-recursive helper function to Ten-Exp that raises 10^**power** by multiplying 10 by **multiple**, and recursively calling with power and multiple.

Right-Justify (string width): string number -> string
This function right justifies **string** within a string of size **width.**

Line (width): number -> string
This function returns a line composed of –'s that is length **width.**

Valid-Check (input): number -> display
This checks to see if the inputs to lmult and ldiv are valid, a.k.a making sure they are not a list and are positive numbers.

Multiplication only:

Lmult(factor1 factor2): number number -> display
This is the main driver for multiplication, and takes in two numbers, **factor1** and **factor2** and multiplies them.

Multsteps (factor1 factor2 exp maxsize): number number number number -> display
This function shows the steps taken in multiplying **factor1 factor2** by using **exp** to hold the exponent to raise 10 to, to traverse between the ones, tens, hundreds, etc. places and using **maxsize** to note the biggest length of **factor1 factor2**.

Largest (list): (number) -> number
This function returns the largest number in a list of numbers.

Division only:

Ldiv (divd divs): number number -> display
This is the main driver for division, it divides **divd** by **divs** and outputs the result to the screen.

Size-Zero (num length): number number -> number
This is the same as size, however if the **num** is 0 this function returns a **length** of 1.

Print-Remainder (divd divs): number number -> display
This function displays the remainder when **divd** is divided by **divs**, and does not show it if the remainder is 0.

Make-List (num length): number number -> (number)
This function turns a number **num** of length **length** and inserts each digit of the number into a list and returns the list.

Zero-Check (num): number -> boolean
This is the check to make sure we don't divide by 0.

Start-Up (divd divs quot shift): number number number number -> display
This function is what Ldiv calls at the beginning to display the starting portions of the division. This includes drawing the **divd, divs, quot,** and the |.

Finish (first divd divs maxsize): number number number number -> display
This function is what Ldiv calls at the end to display the finishing portions of the division. This includes drawing the final line and the remainder.

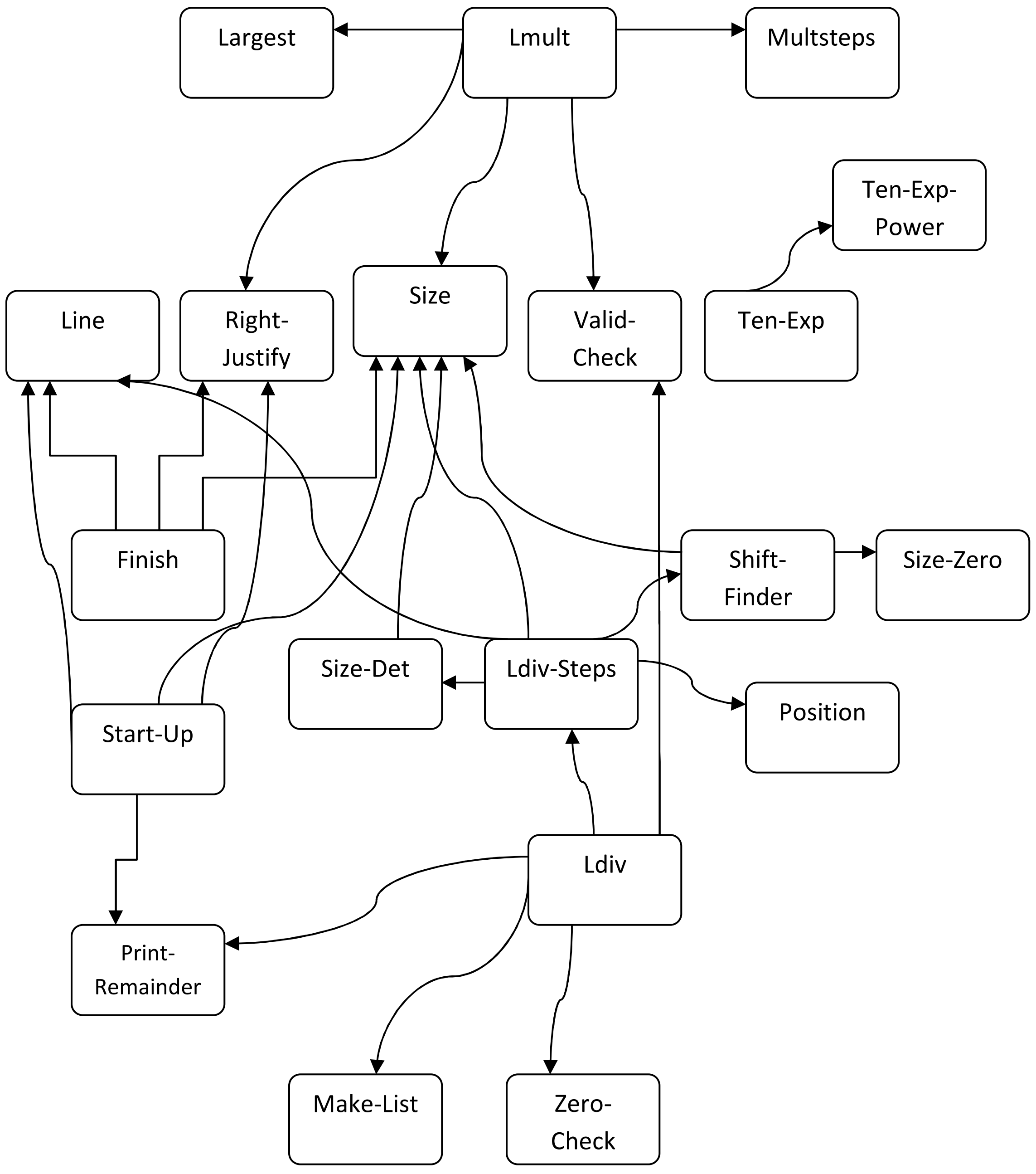
Ldiv-Steps (shift divd divs quot-size lqout): number number number number (number) -> display
This function shows the division steps of Ldiv, by dividing **divd** by **divs**, using the size of the quotient, **quot-size** as both a counter and an exponent value. It uses the quotient as a list **lquot** as well.

Size-Det (sub subdiv): number number -> number
This function determines the line length for division and also has a special case for 0 being the **sub** or subtrahend.

Position (length string): number string -> string
This function returns **length** spaces followed by **string**. It is used to buffer **string** with spaces before it is displayed.

Shift-Finder (top bottom): number number -> number
This function determines the amount of spaces to shift a result by, after analyzing the numbers **top** and **bottom** being subtracted.

Diagram of Function Calls:



Design Rationale:

During the design we tried to format the scheme code to follow the guidelines of high-quality code as closely as possible. Each variable was named for what it was meant to be so that scheme code can be read naturally. One can understand what one particular line of code is meant to do without understanding the actual program flow. This would help others ascertain the program flow and help with debugging. To improve readability, reduce code complexity, and reduce the size of function code, we devolved tasks onto other functions. The code was designed to avoid redundancy by abstracting tasks that we found were being used in multiple occasions such as incorrect input determination. To adhere to the idea of information hiding, the code was constructed so that all callers need not make deep assumptions about the functions called and just expect the correct output based on the input. To allow for extreme programming, we commented on all the code.

For lmult (multiplication):

For lmult, we decided to look at how we do multiplication in real life, which is by multiplying the ones, then tens, then hundreds digit etc. To do this we made a function called ten-exp that returned 10 to a certain power, and first started the lmult function drawing the two factors followed by a line. We then called multsteps, a helper function to lmult to recursively multiply each digit of factor2 by factor1, and used ten-exp to shift this result over in the intermediate step. We then drew the last line and answer, and had some special cases such as if factor2 was a single digit number.

For ldiv (division):

The output of long division was divided into 3 cases. The first was just the display of divisor, dividend, and quotient, with the remainder appearing optionally depending on its existence. This was encapsulated in **start-up**. The middle portion, which shows all the steps of the long division process, was encapsulated as a recursive function in **ldiv-steps**. The end, which depends on the output of **ldiv-steps**, outputs the line and the remainder, which is a special case since there is no carry after this step. This finishing step is encapsulated in **finish**.

The portion that shows the long division steps works as follows. The size of the quotient was used to determine the number of recursive calls, and therefor served as a counter. As an example, if the size of the quotient is 4, then there would need to be 4 recursive calls. A list containing the digit members of the quotient was used, where the digit at the top of the list (using car), is the one that accounts for the arithmetic at the local stage. To account for the position of the displayed ouput, the **shift**, which represents the number of spaces to the right of the text, was incremented based on the local calculation, so as to obtain the correct positioning needed locally. The size of the line is formatted so that it extends from the first digit above it to the last digit below it.