Petit Samuel

CS1021 / Introduction to Computing I

Assignment #1
Simple Calculator

Step 1 – Console Input

Solution Description

This first step was to store a number entered in the console in R4, the tricky part being that it is only possible to get the ascii values for the key pressed one by one.

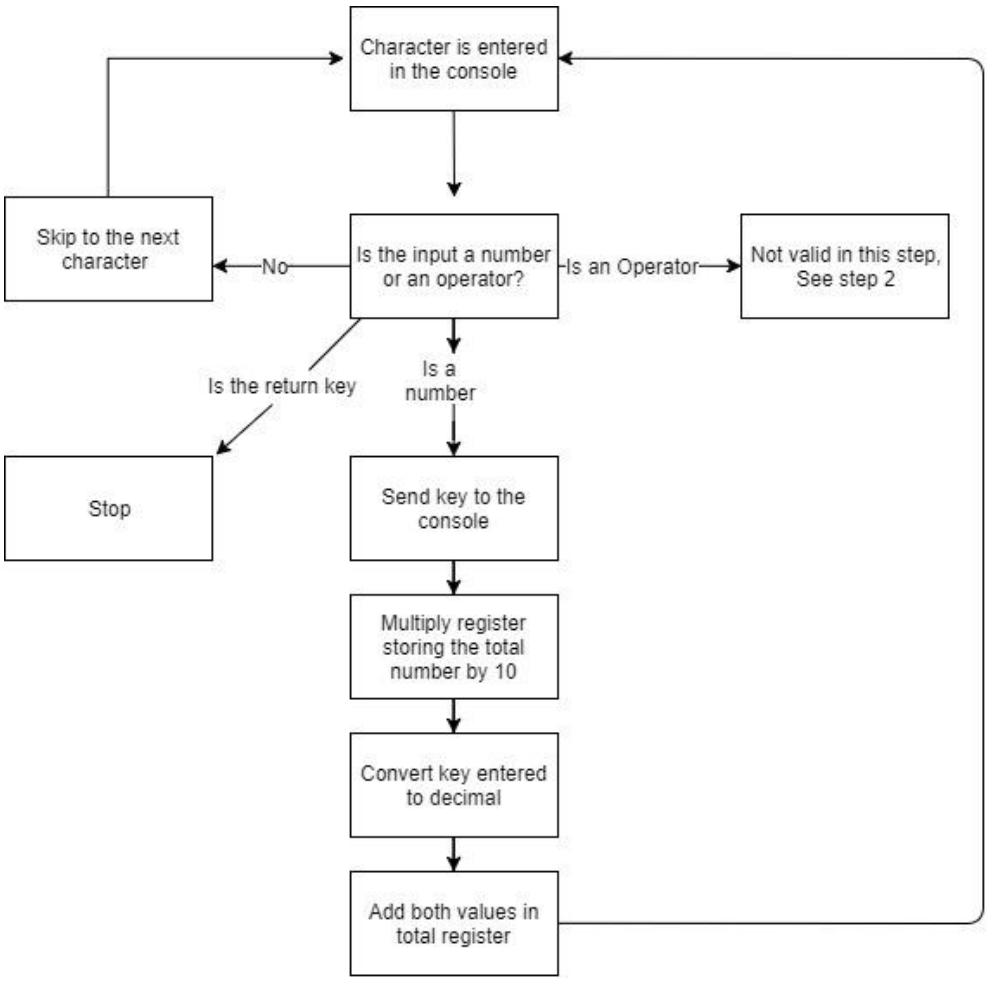
To make it possible, each time a number is entered, computations are made until a specific key is hit to stop the loop (in this case the return key).

The first step after a key is pressed, was to store the value in a register and convert the ASCII value to decimal which only is an easy subtraction:

Input = input – '0'

After that, to add the number to the previous stored, the value already stored is multiplied by 10 and then add the two values together. This was what was required. I added a series of if statements, checking if the characters pressed are actual numbers or operators, if they are not the character is not sent to the console and computations are ignored until a next character is entered in the console.

Here is the step by step process:



Testing Methodology

As I did not make the program take into account negative numbers I only tested positive ones, I first tested values with 1, 2, 3 and 4 decimals, then I tested 0 and then I tested adding random characters in between typing a number:

Input	Value in R4	Value in Decimal	Correct Value
1	0x1	1	1
12	0xC	12	12
345	0x159	345	345
1367	0x557	1367	1367
-12	0xFFFFFEE0	4294967008	-12
9*	0x54	84	9
12azerty1	0x79	121	121
0	0	0	0
0abc12	0xC	12	12
9	0x9	9	9

I tested first values that contained 1, 2, 3 and 4 decimals to verify that the algorithm worked properly using a normal input and all outputs for these are correct. I then tested a negative value to show that I did not take those into account, so it is normal for this program to process it wrong.

Then I tested a number followed with an operator and the output is incorrect as it did treat the operator as a number and did the same process as it would have with a number, this is incorrect and should not be happening.

I then tested whether it would skip computations for other characters (other than numbers and operators) and in this case all outputs are correct. The output for 0 is also correct.

Finally, the reason I tested 9 and the multiply value was because my algorithm will skip certain characters and those two are at the end of this checking (the pseudo code for this checking is the following:

If (input < '(' OR input > '9') {

Skip to next number}

It turns out those values are not skipped and the output for 9 is correct.

Step 2 – Expression Evalutation

Solution Description

This step kept the first one and extended it: it needed to store 2 numbers separated by an operator, and then do the computations in order.

Number1 <operator> Number2

Or in my program:

R5 = R6 <R10> R4

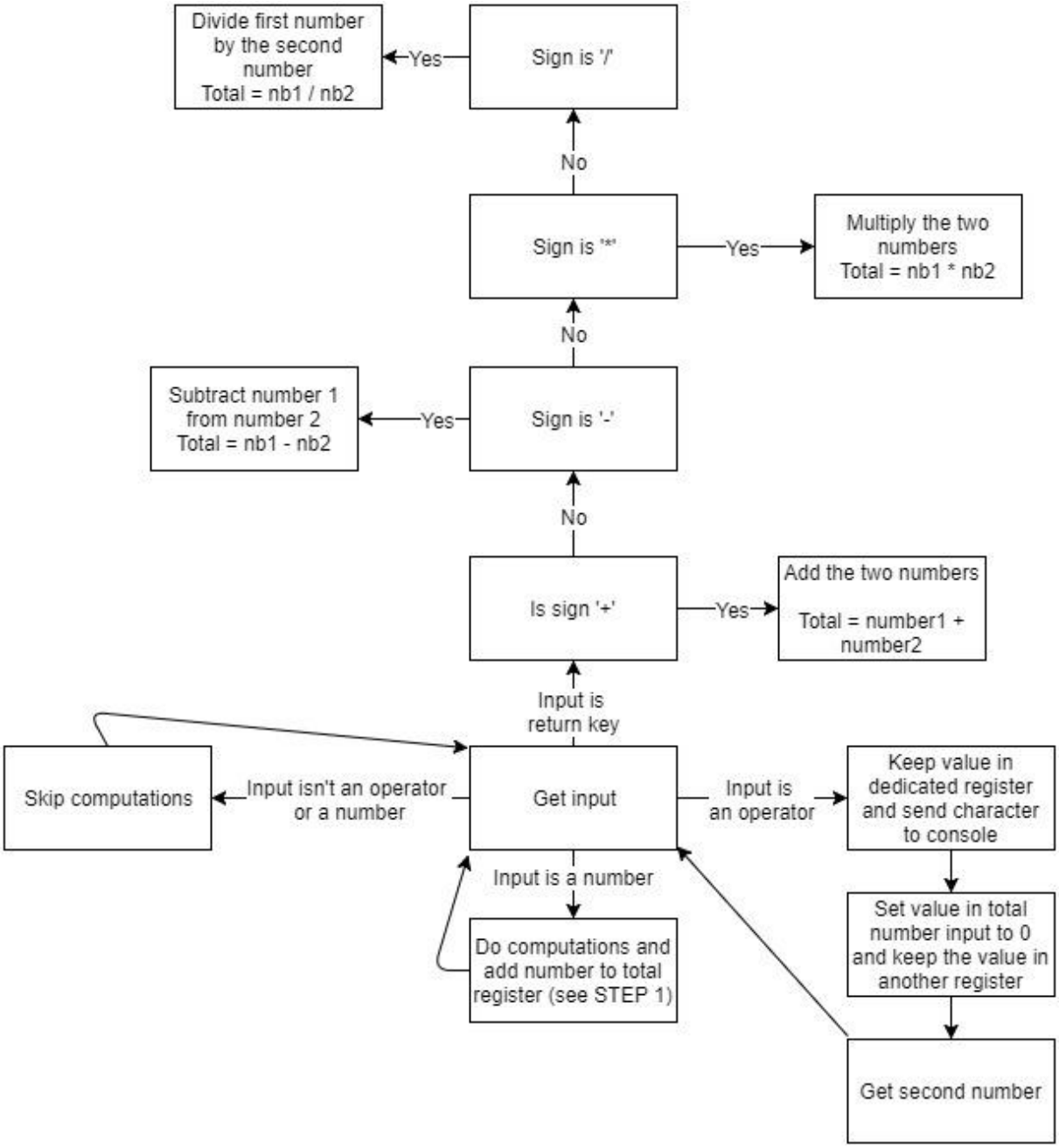
The program needed to work with the addition, subtraction and multiplication and I added division.

The way I approached this was to store those two numbers in separate registers, and keeping the value of the sign to then do the computations.

So first I added an if statement, so that if either the addition, subtraction, multiplication, division character is pressed, the value for that key is kept, the first number is set to zero and the previous value kept in another register. Then the computations are done depending on what the sign register has stored.

The problem with that solution is that if multiple operators are pressed, the values stored in the values and sign registers will most of the time be incorrect.

Skip to next page to visualise how my program works



Testing Methodology

Input	Value in R5	Value in Decimal	Correct Value
1+1	0x2	2	2
19+87	0x6A	106	106
1763+543	0x902	2306	2306
9+0	0x9	9	9
0+0	0x0	0	0
-3+2	0x5	5	-1
12az&é”2+48po	0Xaa	170	170

First, I tested the addition, so I first tried adding random numbers with 1, 2, 3 and 4 decimals. I then tried adding numbers to 0 as well as adding 0 to 0 and all of these have correct outputs in R5.

I then tested adding a negative number to a positive one: -3+2 and the output was 5 instead of -1. That is because my program makes it so that when any operator is entered, the register dedicated to keeping the value of that operator is changed to that last operator and whatever was stored in both number registers are changed: whatever was stored in number1(R6) will now store what was in number2(R4) and number2 is set to 0 waiting for input from the user. There clearly is a better way to deal with this situation than how I did.

I also tested if my input checking still worked by typing random characters in between an addition and the output is correct.

Input	Value in R5	Value in Decimal	Correct Value
0-0	0x0	0	0
1-1	0x0	0	0
19-10	0x9	9	9
100-50	0x32	50	50
-50-100	0xFFFFFFCE	4294967246 (N=1 so value really is 50)	-150
20-50	0xFFFFFFE2 (N = 1)	-30 (checked as if multiplied by -1 R5 = 30)	-30
10a-5’(;	0x5	5	5

I then moved on to test the subtraction using the subtraction using mainly the same approach: first subtracting with and by null values. I also tested subtracting a value by itself and all these outputs were correct. Subtracting with and by any regular will output correct answers (see step 3 to see how I handled negative numbers). However, subtracting with negative numbers will not work. It is also noticeable that random characters that are not numbers nor operators are still not taken into account which is a good thing.

Input	Value in R5	Value in Decimal	Correct Value
0*0	0x0	0	0
0*1	0x0	0	0
9*1	0x9	9	9
10*10	0x35	53	53
198*376	0x1A3	74448	74448

-3*5	0Xf	15	-15
9*-2	0x2	2	-18
98abc*aldj3	0x126	294	294

For multiplying I used the same method and here are my observations: multiplying by zero and normal numbers works perfectly. Multiplying by negative numbers will not work: if the first number only is negative the output will have the wrong sign, if the second number is negative or both are the output will not have the correct value no matter the sign. Random characters are still ignored.

Input	Value in R5	Value in Decimal	Correct Value
0/0	0x0	0	
1/0	0x0	0	
9/2	0x4	4	4
250/125	0x2	2	2
19872/2	0x26D0	9936	9936
-10/5	0x2	2	-2
90/-45	0x2D	45	-2
1ABC2Etr/é&6	0x2	2	2

Finally, for dividing I used the same method as above. Dividing 0 will output zero and dividing by zero will skip computations (which is correct). Divisions are will end up with a remainder output the correct result and regular divisions with no remainder will also output correct result. However, the outputs for negative numbers are incorrect for the same reason as above: if the first number is negative, the output will have to opposite sign. If the second number is negative, then the output will have the incorrect value and possibly an incorrect sign.

Step 3 – Displaying the result

Solution Description

Once the result to output was stored in a register we then had to send it to the console. The problem with that is that it is only possible to send to the console one number at a time (using its ASCII value), and that the result to output was stored in hexadecimal.

The first step was first to send to the console “=” once the return key was hit. And then to check whether the value is positive of negative, if it is negative then the total number is multiplied by -1 and print the character “-“ to the console.

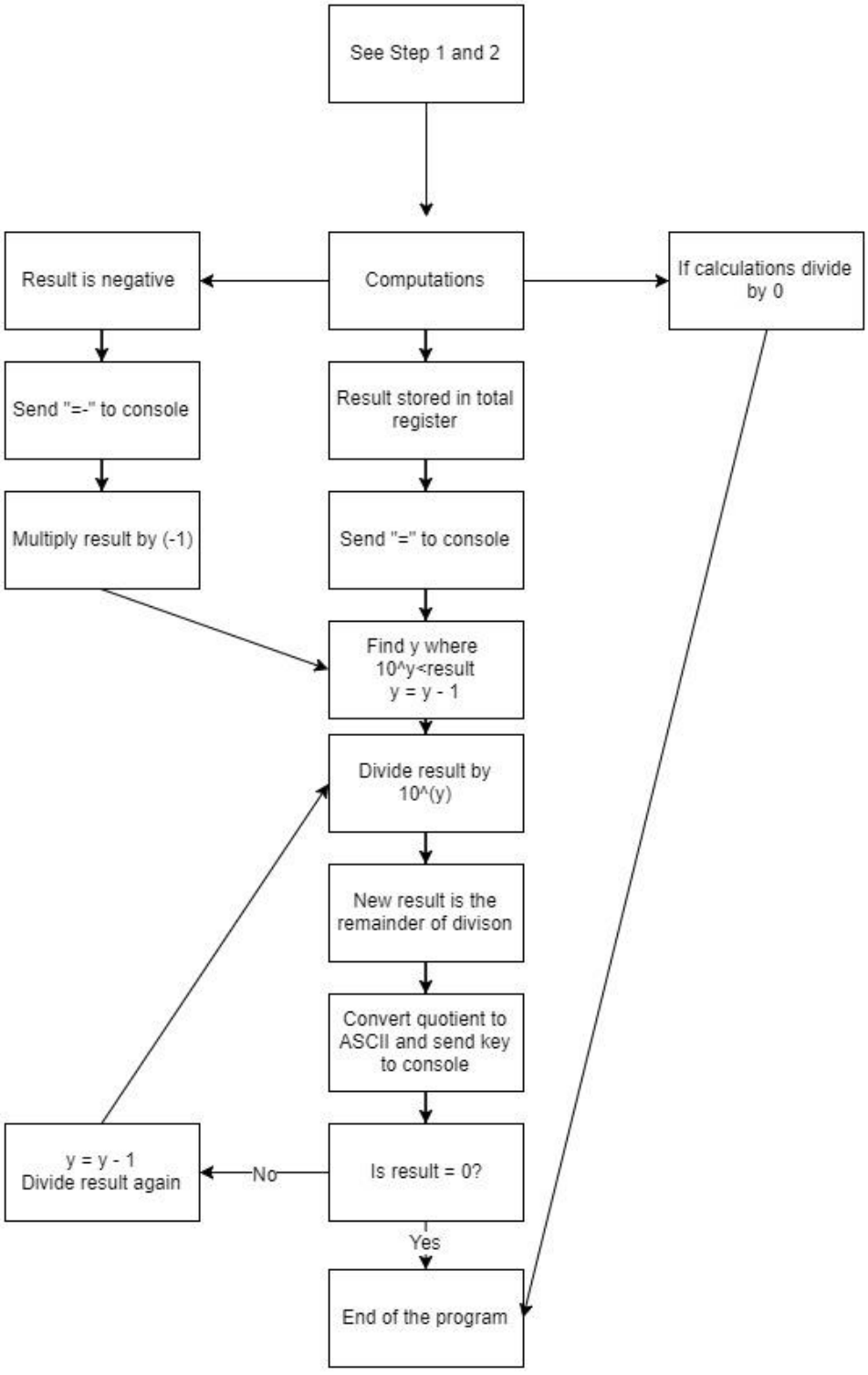
After that, to print out the number one by one I decided to divide the total by the correct power of 10, send the quotient to the console and the remainder becoming the new total and keep doing that until the total is equal to zero.

So, the first step is to find the power of 10 that is higher than the total. Then I store in a register the correct power of 10 to divide the total and then execute what was said in the paragraph above.

Once the quotient for one number to output is loaded into R0, what is left is to convert that value to ASCII (+#'0') and send it to the console.

One last note : if I try to divide by 0, the program will skip all calculations and display and stop the program.

Finally, here is the diagram that Illustrates how this last step works for my program :



Testing Methodology

From the testing of previous stages I knew that, besides using negative numbers, the outputs are correct. So, I focused my testing on displaying the result only :

Input	Output	Correct Output
0*0	Calling start() ... 0*0=0	0*0=0
9-9	Calling start() ... 9-9=0	9-9=0
0+0	Calling start() ... 0+0=0	0+0=0
0/2	Calling start() ... 0/2=0	0/2=0
0/0	Calling start() ... 0/0	0/0
1009+3013	Calling start() ... 1009+3013=4022	1009+3013=4022
2000-5000	Calling start() ... 2000-5000=-3000	2000-5000=-3000
9876-7654	Calling start() ... 9876-7654=2222	9876-7654=2222
18b9*546a	Calling start() ... 189*546=103194	189*546=103194
98ABWq71/1&é09b	Calling start() ... 9871/109=90	9871/109=90
-1+10	Calling start() ... -1+10=11	-1+10=9
10--2	Calling start() ... 10--2=-2	10 - -2=8

First, I tested the outputs for adding, multiplying, substracting, diving with null numbers. All the outputs for these are correct. I then tried adding numbers that included zeros in the middle of the result to output, which still was correct.

I then tried if a subtraction of 2 positive numbers that resulted in a negative number to verify that my trick did work, and the output is correct, so it did.

I then tested normal multiplications and divisions typing random characters in the console in between numbers and operators and the output were still correct.

Finally, as my program does not get correct values for negative numbers for reasons explained in Step 2, it is normal for it not to output calculations using negative numbers correctly.

How to Improve this program

There are several ways in which this program can be improved. Here are a few :

- When it does skip random characters that are not numbers or operators, the characters ‘,’ and ‘.’ Are still included and they are not useful for this program.
- Making this program work with negative numbers
- Being able to use multiple operands and operators
- Output the decimals of the result of a division in the case where there is a remainder
- Instead of stopping the program in the case that there is a division by 0, print out to the console text before stopping (ex: “Impossible to divide by zero”).