# Multi-digit multiplication

Concepta Njolima

February 6, 2022

**Abstract**

Although multiplying single digits by other single digits is easy, multiplying multi-digit numbers requires more steps. In this paper I will walk through an algorithm for multiplying two multi-digit numbers building off the algorithm for multiplying a multi-digit number and a single digit. Then, I will implement and test this algorithm using Octave software.

## 1 Introduction

Multiplying single digit numbers, say 2 multiplied by 1, is quite a simple task. However, the product of 96 and 3 might require more steps because you do not memorize such products. For example, to calculate 96 * 3, we can let 96 be the number and 3 the multiplier. A possible solution is multiplying the digit in the ones place with the multiplier, that is 6 * 3 = 18. Since 18 is greater than 10, we carry 1 to the tens place value and keep 8 as the product in the ones place value. Then multiplying the tens place value which is 9 * 3 = 27 and adding the carry to get 27 + 1=28. Since 28 is in the tens place, we get 28*10= 280. Then adding the two results, 280 + 8 =288. Therefore 96 * 3=288.

This step-by-step process of solving a problem is an algorithm. We just developed an algorithm for multiplying a multi-digit number with a single digit. But the question remains, how do we multiply two multi-digit numbers?

## 2 Multi-digit multiplication example

To calculate the product of 96 and 20, we build on the single-digit example above and view this problem as two single-digit multiplications; where we add the product of 96 and 0 and the product of 96 and 2(taking into account the place values of the multipliers). The algorithm is as follows:

- Starting with the least significant digit in the first number, that is 6 and the least significant digit in the second number which is 0. We multiply 6 by 0 resulting into 0. Since 0 is less than 10, the least significant digit of the product will be 0
- Then, in the tens position of the first number, we multiply 9 by the ones position of the second number, 0, to get 0. To account for the place value, we multiply 0 by 10 to get 0.
- Adding the results in the ones and tens place values, we get 0 + 0=0. Hence 96 * 0 = 0.
- Then, we use the digit in the tens place of the second number as the multiplier. Starting with the ones place, we get 6 * 2 = 12.
- Since 12 is greater than 10, we carry 1 and the result in the ones place is 2. To account for the place value, we multiply 2 by 10 to get 20.
- Then in the tens place of the first number, 9 * 2 = 18. Adding the carried value, we get 18 + 1= 19. To account for the place value, we multiply 19 by 100 to get 1900.
- Therefore 96 * 20 =1900 + 20 + 0 = 1920.

The vertical representation of this algorithm is shown below:

$$\begin{array}{r} 9\,6 \\ \text{x}\quad 2\,0 \\ \hline 0\,0 \\ +1\,9\,2 \\ \hline 1\,9\,2\,0 \end{array}$$

## 3 Flow Chart description

The algorithms explained above can be represented using a flow chart. Figure 1 in this section shows the flowchart for the multi-digit multiplication algorithm.

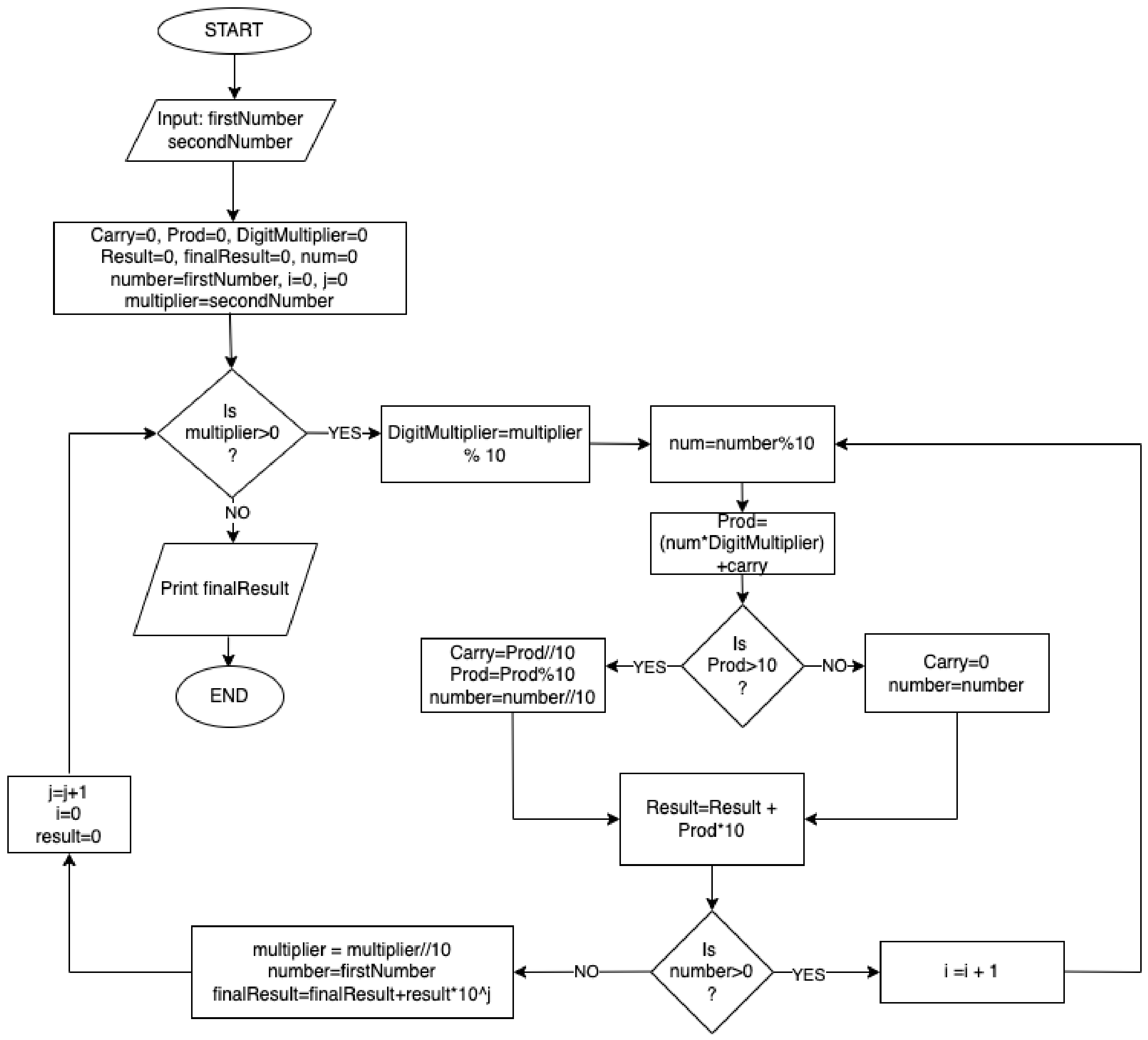

START
Input: firstNumber secondNumber
Carry=0, Prod=0, DigitMultiplier=0
Result=0, finalResult=0, num=0
number=firstNumber, i=0, j=0
multiplier=secondNumber
Is multiplier>0 ?
YES
NO
DigitMultiplier=multiplier % 10
num=number%10
Prod= (num*DigitMultiplier) +carry
Print finalResult
END
Is Prod>10 ?
YES
NO
Carry=Prod//10
Prod=Prod%10
number=number//10
Carry=0
number=number
j=j+1
i=0
result=0
Result=Result + Prod*10
Is number>0 ?
NO
YES
multiplier = multiplier//10
number=firstNumber
finalResult=finalResult+result*10^j
i =i + 1

Figure 1: This is a flow chart showing multi-digit multiplication

We input two digits to multiply named firstNumber and secondNumber. Then we initialize the variables we will use in the algorithm like carry, Prod, result, finalResult, i, j, DigitMultiplier, and num to zero. Additionally, we set temporary variables number to firstnumber and multiplier to secondNumber. If multiplier is greater than 0, we set the DigitMultiplier to the modulus of multiplier and 10. Then, number becomes the the modulus of num and 10 and Prod equals the sum of num multiplied by DigitMultiplier and carry. If Prod is greater than 10, carry is set to the floor division of Prod and 10, Prod then becomes the modulus of Prod and 10 and number becomes the floor division of number and 10. Then the result is incremented by Prod multiplied by 10. If the Prod is less than

10, carry is set to 0 and number equals number. Then the result is incremented by Prod multiplied by 10.

Then we check if number is greater than 0. If yes, we increment the counter i by 1. If no, we set the multiplier to the modulus of the multiplier and 10, number to firstNumber, and finalResult to the sum of finalResult and the product of result and 10 power j. Finally, we increment j by 1, reset i and result to 0. Then repeat the process of checking if the multiplier is greater than 0.

If the multiplier is less than 0, we print the finalResult and end the algorithm.

## 4 Octave implementation

The code below implements the multi-digit multiplication algorithm explained above in Octave.

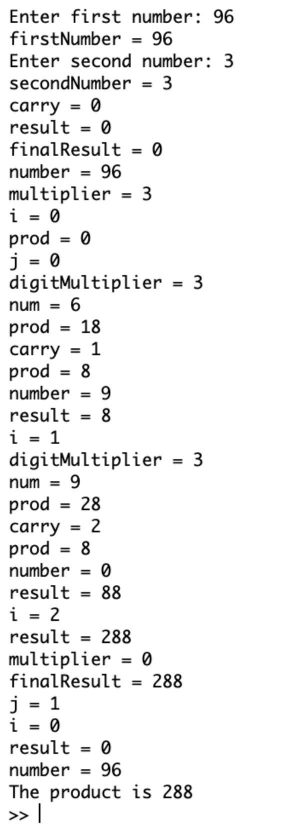
```
% Algorithm multiply two numbers

firstNumber=input("Enter first number: ")
secondNumber=input("Enter second number: ")

%initialize variables
carry=0
result=0
finalResult=0
number=firstNumber
multiplier=secondNumber
i=0
prod=0
j=0

while multiplier>0 % multiply by each digit in the multiplier
  while number>0 % multiply all the single digits in the number
      digitMultiplier=mod(multiplier,10)
      num=mod(number,10)
      prod=num*digitMultiplier+carry
      if prod>10
        carry=floor(prod/10)
        prod=mod(prod,10)
        number=floor(number/10)
      else
        carry=0
        number=floor(number/10)
      endif
     result=(prod*10.^i)+result
     i=i+1
    %endif
  endwhile
  if carry ~=0
    result=result+(carry*10.^i)
  endif
  multiplier=floor(multiplier/10)
  finalResult=finalResult+(result*10.^j)
  j=j+1 % move the next single-digit product result to the left by one place value
  i=0   % reset i
  result=0  % reset the single-digit product
  number=firstNumber  % set the number to the initial first number
endwhile
fprintf('The product is %i\n',finalResult)
```

We use while loops to check when the multiplier is greater than 0 and when the number is greater than 0 because these two cases needed to run for as long the specified conditions were true. A while loop implemented such a behavior better than a for loop because the number of times the loop runs varies based on the digits in the input numbers. The while loop that runs when number is greater than 0 is nested inside the while loop that runs when multiplier is greater than 0 because the latter condition needs to be fulfilled before the code in the former while loop executes.

Also, we use if conditionals to make decisions based on a condition, like checking if a variable has a value greater than 0. Using the if conditional allowed me to separate the order of execution of different processes using if and else parts of the conditional.

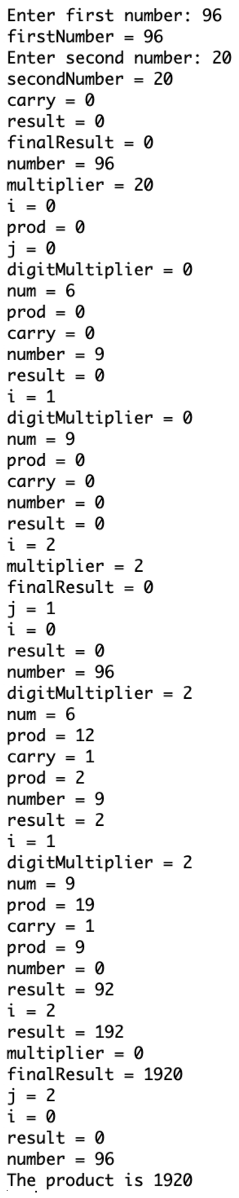
Then, counters i and j track the place values of the digits in the number and digit in the multiplier respectively.

## 4.1 Results from running the Octave implementation



```
Enter first number: 96
firstNumber = 96
Enter second number: 3
secondNumber = 3
carry = 0
result = 0
finalResult = 0
number = 96
multiplier = 3
i = 0
prod = 0
j = 0
digitMultiplier = 3
num = 6
prod = 18
carry = 1
prod = 8
number = 9
result = 8
i = 1
digitMultiplier = 3
num = 9
prod = 28
carry = 2
prod = 8
number = 0
result = 88
i = 2
result = 288
multiplier = 0
finalResult = 288
j = 1
i = 0
result = 0
number = 96
The product is 288
>> |
```

(a) Product of 96 and 2 in octave



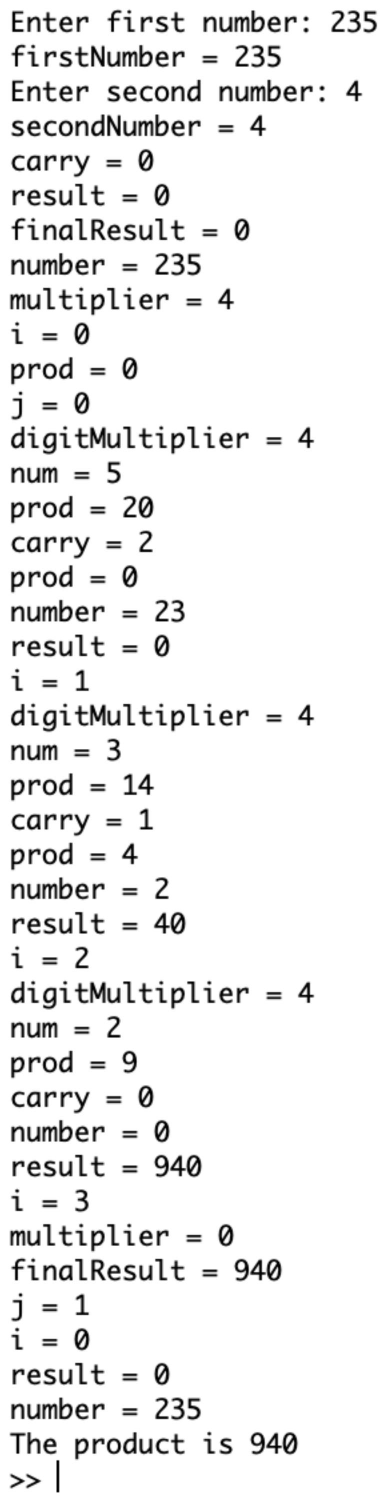
```
Enter first number: 96
firstNumber = 96
Enter second number: 20
secondNumber = 20
carry = 0
result = 0
finalResult = 0
number = 96
multiplier = 20
i = 0
prod = 0
j = 0
digitMultiplier = 0
num = 6
prod = 0
carry = 0
number = 9
result = 0
i = 1
digitMultiplier = 0
num = 9
prod = 0
carry = 0
number = 0
result = 0
i = 2
multiplier = 2
finalResult = 0
j = 1
i = 0
result = 0
number = 96
digitMultiplier = 2
num = 6
prod = 12
carry = 1
prod = 2
number = 9
result = 2
i = 1
digitMultiplier = 2
num = 9
prod = 19
carry = 1
prod = 9
number = 0
result = 92
i = 2
result = 192
multiplier = 0
finalResult = 1920
j = 2
i = 0
result = 0
number = 96
The product is 1920
```

(b) Product of 96 and 20

Figure 2: Results of multi-digit multiplication for different inputs

Entering the example in section 1 in Octave where the number is 96 and the multiplier is 2 yielded results shown in figure 2a. Then the product of 96*20 in Octave is shown in 2b.

It is worth noting that this algorithm extend to numbers with more than two digits as illustrated in 3 below.



```
Enter first number: 235
firstNumber = 235
Enter second number: 4
secondNumber = 4
carry = 0
result = 0
finalResult = 0
number = 235
multiplier = 4
i = 0
prod = 0
j = 0
digitMultiplier = 4
num = 5
prod = 20
carry = 2
prod = 0
number = 23
result = 0
i = 1
digitMultiplier = 4
num = 3
prod = 14
carry = 1
prod = 4
number = 2
result = 40
i = 2
digitMultiplier = 4
num = 2
prod = 9
carry = 0
number = 0
result = 940
i = 3
multiplier = 0
finalResult = 940
j = 1
i = 0
result = 0
number = 235
The product is 940
>> |
```

Figure 3: Extending the algorithm above to a three-digit number

## 5 Conclusion

In this paper, we develop and implement an algorithm for multi-digit multiplication. This algorithm only works for positive whole numbers. We assumed single-digit addition and multiplication and multi-digit addition. It does not compute products containing negative, decimal or fractional inputs.

### 5.1 Acknowledgements

This was my first time using Latex. Special thanks to Eleni Lazaridou who guided me on how to get started with Latex. I appreciate Kristi Nakaj and Immanuela Belaineh who reviewed my algorithm and suggested possible ways to improve it.