

University of Cincinnati
Department of Electrical & Computer Engineering and Computer Science

20 ENFD 112 – Fundamentals of Programming

LABORATORY 6: FUNCTIONS, ARRAYS

Spring 2008

1. Objective

The objectives of this assignment are to a) design a solution to a complex problem which has several subparts that must be integrated to achieve a desired result; b) make use of functions in a logical manner to facilitate a clear solution to the subparts.

2. Background

Matlab does not allow arbitrarily large integers. In fact, the maximum integer expressible by MATLAB is quite small: only about 10 digits maximum are represented. Larger integers become floats which means they are inaccurately represented. For example, even 30! (factorial of 30) cannot be represented accurately in MATLAB. But, there are numerous applications where large integers are required, for example modern cryptographic systems need integers of 100s of digits to function in a secure manner. This lab will get you to think about creating a feasible, although inefficient, solution to the problem in MATLAB.

In what follows we will be talking about creating and maintaining a new data type called `BigInteger`. A `BigInteger` is a character string of arbitrarily many decimal digits, one digit per character. A `BigInteger` is printable: that is, all the characters used in a `BigInteger` are taken from the following set:

`{'0','1','2','3','4','5','6','7','8','9','-'}`.

But in this lab we will dispense with the complexity of negative numbers and not deal with the `'-'` character.

3. Problems

You will build a collection of functions to support multiplying two `BigInteger`s. You will build a factorial function using this new multiply function and find some large factorials. The subproblems you will be working on are listing in the following subsections. Each subsection asks to build an independent function.

3.1 Conversion of integer to `BigInteger`

Write a function called `toBigInt` which converts a given integer `n` to a `BigInteger`. Use the following prototype:

```
function number = toBigInt(n);
```

Thus, if `n` has a value of 3371209, then the string '3371209' is returned by `toBigInt`. I have decided to give this one to you below, at no cost to you. It uses the built-in Matlab `int2str` function for converting integers to strings (do not use `num2str` because we are dealing exclusively with integers here).

```
% function to convert an integer to a corresponding string of
% character digits
function number = toBigInt(n);
    number = int2str(n);
end
```

Test your code before continuing.

3.2 Shift a BigInteger k decimal places to the left

Write a function called `shiftLeft` which takes a BigInteger `n` and an integer `k` as input and returns `n` with `k` zero characters appended to it on the right. Thus, if `n` is '345' and `k` is 3, the string returned by `shiftLeft` is '345000'. This operation will be used when multiplying two BigIntegers. This is the most straightforward function we ask you to write in this lab. It should require no more than 5 lines of code all together.

Test your code before continuing.

3.3 Add two BigIntegers

Write a function called `addBigInt` which takes two BigIntegers `n1` and `n2` as arguments and returns the string equivalent of `n1+n2` if `n1` and `n2` were numbers. For example, if `n1` is '345' and `n2` is '664' then the string 1009 is returned. This function should take 20 lines of code all together. You should use the algorithm for adding that you learned in grade school: namely, add a column of digits at a time, from right to left, plus a carry from the previous column, if necessary. You might consider the following hints if you are stuck.

1. Positive integers less than 256 may also be considered by Matlab to be characters but you need some MATLAB functions to help your programs understand this. For example, the character '0' is actually the number 48. You can write `3+'0'` at the MATLAB prompt and the result is 51. But suppose you do not want the 51 and want the corresponding character instead? Then use `char(3+'0')` and you will see the string 3 on the screen. Using `char` will be important to signify `char` type when adding elements to an array.
2. The result of adding two single digit integers (not characters) is either a single digit or a two digit number. Call the result `n`. In the later case, the most significant digit (on the left) is the carry to the next column to be added. You can find the carry with `floor(n/10);`. You can find the least significant digit (right digit) with `mod(n,10)`.

Test your code before continuing.

3.4 Multiply a BigInteger by a single character digit

Write a function called `multiplyLine` which takes a BigInteger `n` and a single character `d` representing a number between 0 and 9 and returns a string representing the multiplication of the numbers represented by `n` and `d`. For example, if `n` is '345' and `d` is '5', then the string '1725' is returned. This is similar to the addition of Section 3.3 except that multiplications take place

instead. There is still a carry to worry about, though. The MATLAB function `size(n)`; might come in handy for convenient indexing of the digits of `n`. This function should take less than 15 lines.

Test your code before continuing.

3.5 Multiply two BigIntegers

Write a function to multiply two `BigInteger` numbers. Use the following prototype:

```
function number = multiplyBigInts(num1, num2);
```

Thus, if `num1` is the string '87734' and `num2` is the string '63321' then the string '5555404614' is returned by `multiplyBigInts`. Use the grade school algorithm of repeatedly multiplying a digit `d` of `num1` by `num2`, shifting the intermediate result by the position of `d` from the right in `num1`, and adding all the shifted intermediate results obtained. This can be written in less than 10 lines of code using `shiftLeft`, `addBigInt`, and `multiplyLine`, which you already developed and tested. Again the `size` function might come in handy here.

Test your code before continuing.

3.6 Factorial of large numbers

Use `multiplyBigInts` and `toBigInt` to build a function for finding the factorial of a given integer `n`. The factorial function is normally expressed as follows:

```
% Factorial of integer n. The type of number is integer as well
function number = factorial(n);
    number = 1;
    for i=1:n
        number = number*i;
    end
end
```

But this solution wimps out at `n=21` whereupon a value of `5.1091e+19` is returned. But actually `21!` is `51090942171709440000`. With `BigIntegers` one can write:

```
% Factorial of integer n. Returns a BigInteger - no digit loss
function number = factorial(n);
    number = '1';
    for i=1:n
        number = multiplyBigInts(number, toBigInt(i));
    end
end
```

which looks very close to the original code but which gives, for example, the following output for `n=500`:

```
122013682599111006870123878542304692625357434280319284219241358838584
537315388199760549644750220328186301361647714820358416337872207817720
048078520515932928547790757193933060377296085908627042917454788242491
```

```
272634430567017327076946106280231045264421887878946575477714986349436
778103764427403382736539747138647787849543848959553753799042324106127
132698432774571554630997720278101456108118837370953101635632443298702
956389662891165897476957208792692887128178007026517450776841071962439
039432253642260523494585012991857150124870696156814162535905669342381
300885624924689156412677565448188650659384795177536089400574523894033
579847636394490531306232374906644504882466507594673586207463792518420
045936969298102226397195259719094521782333175693458150855233282076282
002340262690789834245171200620771464097945611612762914595123722991334
016955236385094288559201872743379517301458635757082835578015873543276
888868012039988238470215146760544540766353598417443048012893831389688
1639487469658817504506926365338175055478128640000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
```

That is some crazy number which is greater, by far, than the number of atoms in the universe!!!!
No more wimping out with these babies!!!

Test your code before continuing.

3.6 Number of zeros in the tail

Did you notice there is a relationship between the number of 0s at the end of `factorial(n)` and `n`? Plot the number of 0s as a function of `n` up to 500 and see what you get. From this plot, can you guess where the 0s are coming from (well, maybe you do not need a plot to see this but this is a long lab and you need to have some fun so go ahead)?

4. Analysis

If you get this without significant help you have definitely moved up a notch or two since you arrived at UC. There are many things we have to consider to get this right. For one thing, multiplication of two numbers `numb1` and `numb2` involves several intermediate steps of the following kind: (assume an accumulator variable is initialized to "0").

1. multiply `numb1` by a single digit of `numb2`.
2. shift the result to the left by the number of positions from the right the single digit of `numb2` is located.
3. add the result to the accumulator.

This suggests breaking the multiplication into three subfunctions with the following prototypes:

```
% Mutiply a given BigInteger numb1 by a single digit d of numb2.
% Assume '0' <= d && d <= '9' and n is non-negative.
function number = multiplyLine(numb1, d);
```

```
% Multiply a given BigInteger numb1 by 10^k (that is, shift the BigInteger
% k places to the left).
function number = shiftLeft(numb1, k);
```

```

% Add two BigIntegers.
% Assume numb1 and numb2 are non-negative numbers.
function number = addBigInt(numb1, numb2);

```

for taking care of the enumerated operations, respectively. In `multiplyBigInts` we do not worry about negative numbers - the ambitious student can make modifications to support them. The multiplication between the magnitudes of the input numbers (`numb1` and `numb2`), using the above functions, can be expressed as follows:

```

acc = '0';           % Accumulates the result
k = 0;              % Line multiply shifts k position
[x sz] = size(numb2); % Get the number of digits of numb2
for i=sz:-1:1       % Multiply ith digit of numb2 from left
    acc = addBigInt(acc, shiftLeft(multiplyLine(numb1, numb2(i)), k));
    k = k+1;
end

```

It should not be too hard to see how to multiply a `BigInteger n` by a single digit which is a character `d` between '0' and '9': convert `d` to a number by subtracting '0' from it; do the same for each digit of `n`; multiply in turn, from the right, a numerical digit of `n` with the numerical `d` and add the result to a carry from the previous multiplication (this gives, at most, a two digit number), use `mod` to compute the least significant digit of the result and `floor(result/10)` to compute the most significant digit (which is the carry for the next single digit multiply step), and add '0' to the least significant digit so it becomes a printable character again.

If you noticed that there is a lot of waste in the above you are sharp. Two places for improvement come to mind. First, why bother subtracting '0' when it is only added back later. Why not skip this step and use the numbers from 0 to 9 to represent each `BigInteger` digit? Second, by doing so, you can pack two digits into one character since only 4 bits are needed to uniquely represent all the numbers from 0 to 9. Then you multiply pairs of digits in a step of `multiplyLine` and you add pairs of digits in a step of `addBigInt`. So you can greatly economize on space and time but it will cost you more code. You are invited to try this but you are not required to do it.

Similarly, it should not be hard to see how to add two `BigIntegers`.

5. Help

Some things that might be helpful:

1. If `n` is a two digit number, `floor(n/10)`; gives the most significant digit of `n` and `mod(n,10)`; gives the least significant digit of `n`.
2. Two objects of type integer and character can be added. The result is equal to the ascii value of the character plus the integer. For example `3+'0'` is 51 because the ascii value of '0' is 48. To turn the result back into a character use, for example, `char(3+'0')`;
3. If `n` is an array, an element `a` can be appended to the front of `n` using `n = [a n]`; . An element `a` can be appended to the rear of `n` using `n = [n a]`;
4. To find the size of an array `n` (how many elements it has) use `[x sz] = size(n)`; . Then `sz` has the size.

5. To define an array `n` (an error message might say a variable is undefined) use `n = []`; which gives `n` zero size.
6. To index element `i` of array `n` use `n(i)` in some expression.

6. Submission

Submit `toBigint.m`, `shiftLeft.m`, `addBigInt.m`, `multiplyLine.m`, and `multiplyBigInts.m` on or before May 11 using blackboard. If you have an answer to the question of the number of zeros at the end of `factorial(n)` in terms of `n`, write it into `multiplyBigInts.m` as a comment. See the course webpage at

<http://gauss.ececs.uc.edu/Courses/HTML/E112.html>

for instructions.