ECE 241: Data Structures and Algorithms- Fall 2022

Project 2: FunWithBigNumbers (Stack+Doubly-Linked List)

Due Date: **Deadline: see website, class policy and moodle for submission**

Description

In few applications in numerical approximations, or in cryptography with the computation of large prime numbers, it is essential to handle numbers with 'infinite' precision. In almost all programming languages, the standard native types such as int and float are limited in size. For example, an int variable in Java, C, (etc.) has a maximum absolute value of 2,147,483,647=$2^{31}-1$ (largest known prime until 1855), and primes generated for cryptography purposes can easily be several hundred digits. The situation is similar for floating point numbers (stored in double precision with "only" 16 digits digital accuracy). To overcome these limitations, a common approach is to implement a custom number class, that supports addition, subtraction, multiplication and division of big numbers. These big numbers are commonly stored using lists (where each item of the list stores a particular digit). For example the following integer: `i=123456789123456789123456789`
can be written as: `ilist=[1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9]`

In fact, Python already provides an arbitrary precision arithmetic for int. The list implementation of the integer is implicit, as soon as the number becomes too big (greater than $2^{31}-1$), python switches automatically its internal representation to a list (without you, seeing it). So the number can expand indefinitely. For example, you could easily compute `i**2` with the number above and python will return a bigger number (this is not the case with most other languages). Python does not provide this feature with floating numbers where the number of representative decimal digits stays stuck at 16. To simplify a bit this project, however, we are not going to consider floating big numbers but integer big numbers. It means that we are going to develop our own big number class for integer regardless of the fact that this functionality is redundant for Python.

To make the project more relevant, we are going to use a Doubly Linked-List to store the big numbers (which will end up being more efficient than the Python list). We will develop our own number class, called **BigNumberLL** using a **Doubly-Linked-List** and for **integer only**, that can be used for arbitrary size comparison, addition, subtraction, multiplication (division is optional/bonus) of big numbers. As a result, the only limitation on the size and precision of a big number will be the available memory on your computer, which means that your class will easily be able to handle numbers that are hundreds of digits long.

For this project, we will be using a (simple) RPN Stack calculator approach (seen in class/activity). It means that you will be able to push/pop/peek/... big numbers to/from the stack. The simple RPN calculator is implemented using a Python list (array) where items will be big number objects.

The project includes multiple files:

- `Stack.py` file containing basic stack class operations (provided).
- `StackCalc.py` file containing extended operations on Stack using multiple methods for RPN calculations (provided).
- `rpn.py` contains the main method allowing RPN calculations on big numbers (provided).
- `BigNumberLL.py` file containing the double-linked list to represent the big numbers, with associated methods to operate on big numbers (To complete).

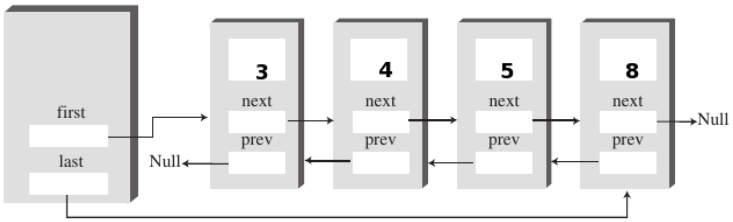
Submission/Grading Proposal

Only **one file** `BigNumberLL.py` must be submitted on moodle. Only one submission by group of two. Write your name on top of the file. This project will be graded out of 100 points:

1. Your program should implement all basic functionality/Tasks and run correctly (90 pts).
2. Overall programming style with comments (including function header doc-string) (5 pts).
3. Pre-submission deadline for Preliminary (answering first question) (5pts). The program does not have to run correctly for pre-submission.

How to represent BigNumbers?

You will be using a doubly linked list (double-ended as well). For example: the number **3,458** can be implemented as follows:



The class `Link` (below) is already implemented inside the file `BigNumberLL.py` where the `data` attribute will be used to represent an integer number (a digit between 0 and 9).

```
class Link:
  def __init__(self, data):
      self.data = data
      self.prev = None
      self.next = None

  def __str__(self):
    return str(self.data)
```

To start with, in the constructor of the class `BigNumberLL` you could define the following attributes:

```
self.first = None
self.last = None
self.size=0
self.positive=True
```

Note that we have added the attribute `positive` that stores the sign of the big number (Remark: the correct terminology should be "non-negative" rather than positive, since the zero is included in our positive set).

Preliminary [20pts]

Let us test the big number representation using your linked-list. Run `rpn.py` and enter various numbers into the Stack. This is what you should obtain for various inputs:

```
Welcome to the RPN calculator ('quit' to quit)
-----------------------------------------------
$ 1
[1] 1
-----------------------------------------------
$ 123
[1] 1
[3] 123
-----------------------------------------------
$ 1234
[1] 1
[3] 123
[4] 1,234
-----------------------------------------------
$ 123456789123456789
[1] 1
[3] 123
[4] 1,234
[18] 123,456,789,123,456,789
-----------------------------------------------
$ -3560568
[1] 1
[3] 123
[4] 1,234
[18] 123,456,789,123,456,789
[7] -3,560,568
-----------------------------------------------
$ 0000023
[1] 1
[3] 123
[4] 1,234
[18] 123,456,789,123,456,789
[7] -3,560,568
[2] 23
-----------------------------------------------
$ -000003456
[1] 1
[3] 123
[4] 1,234
[18] 123,456,789,123,456,789
[7] -3,560,568
[2] 23
[4] -3,456
-----------------------------------------------
$ quit
Thanks for using the RPN calculator
```

We note:

- The stack is displayed in reverse order (top at the bottom).
- The constructor of `BigNumberLL` accepts one input String argument that may contain a negative sign (first character).

- When the big number is displayed, the number of digits that composed the number (i.e. size of the Linked-List) is also displayed at first between square brackets.
- If the user enters meaningless zero on the left, they are removed from the Link-list before displaying.
- To ease the reading of the big number, a comma "," is placed every three digits.

Basic methods to implement

- `insertLast`: If you read a String, character by character from left to right, you need to be able to insert each integer (converted from the character) inside the linked-list while keeping the ordering of a queue. For example: 3,458, do `insertLast(3)`, `insertLast(4)`, etc. This method can be used inside your constructor.
- `insertFirst`: reverse the order of insertion. This method is not needed for this preliminary part of the project but it is a kernel method which is needed by the next tasks. While doing an operation (for example addition), you often perform it step by step from the right to the left. The resulting integer needs to be inserted in the list using `insertFirst` to make sure that the ordering of the final number is correct.
- `deleteFirst`: it is used by the `trimFront` method.
- `trimFront`: if the BigNumber list contains a lot of zeros on the left, you can trim this number (remove the zeros) and reduce its size. This method can be used inside your constructor.
- `__str__`: Convert the big number list into a string that can be used to print by the Stack. A negative sign must be added at the beginning of the String if the number is negative (i.e. 'positive' flag equal to False). The size of the linked-list is also displayed at first between square brackets. For full credit, you will also include a comma "," to separate all the thousands. For example: 3458 is "3,458"; 345 is "345", 12349875 is "12,349,875". Hint: various techniques possible. You could scan the linked-list from right to left, keep track of the number of digits, and check the remainder while dividing by 3.

All following Tasks 1, 2, and 3 can be implemented in any order. Task 4 and 5 should be the last two you implement.

Task-1 Comparison math operators >,<,= [15pts]

When you enter one of these operators in your prompt, the Stack will pop two big numbers, perform the comparison and display `True` or `False` in the prompt (not in the Stack). This comparison should work for any signs as well. Examples:

```
[1] 1
[2] 12
[3] 123
[4] 1,234
[18] 123,456,789,123,456,789
[7] -3,560,568
[2] 23
[4] -3,456
-----------------------------------------------
$ >
True
[1] 1
[2] 12
[3] 123
[4] 1,234
[18] 123,456,789,123,456,789
[7] -3,560,568
-----------------------------------------------
$ <
False
[1] 1
[2] 12
[3] 123
[4] 1,234
-----------------------------------------------
$ =
False
[1] 1
[2] 12
-----------------------------------------------
$ <
True
-----------------------------------------------
$
```

These are just few examples, make sure every possible cases work.

Basic methods to implement

- `__gt__`: this is an operator overloading method for $>$ that will compare two Bignumbers x and y (x is self) and return `True` if $x > y$, or `False` otherwise. Hint: (i) you can first check if the numbers have opposite signs (if so you are done), (ii) if they have the same sign, you can compare their number of digits (`size`), (iii) if they have the same number of digits (and same signs) you need to scan the linked-lists from left to right and compare digit by digit.
- `__lt__`: this is an operator overloading method for $<$ that will compare two Bignumbers x and y (x is self) and return `True` if $x < y$, or `False` otherwise. Hint: You could check if $-x > -y$ (again using the $>$ method), if `True` it means that $x < y$.

- `__eq__`: this is an operator overloading method for == that will compare two Bignumbers x and y (x is self) and return `True` if $x == y$, or `False` otherwise. Hint: Check if both the results of $x < y$ and $x > y$ are `False`.

Task-2 Operation + [15pts]

As you may have guessed, we need to perform the addition of two big numbers. For the time being (for this task), we will consider that the two big numbers are positive (including zero). Here five tests examples by increasing level of difficulty:

```
Welcome to the RPN calculator ('quit' to quit)
-------------------------------------------
$ 123456
[6] 123,456
-------------------------------------------
$ 654321
[6] 123,456
[6] 654,321
-------------------------------------------
$ +
[6] 777,777
-------------------------------------------
$ 111
[6] 777,777
[3] 111
-------------------------------------------
$ +
[6] 777,888
-------------------------------------------
$ 2222111
[6] 777,888
[7] 2,222,111
-------------------------------------------
$ +
[7] 2,999,999
-------------------------------------------
$ 222
[7] 2,999,999
[3] 222
-------------------------------------------
$ +
[7] 3,000,221
-------------------------------------------
$ 8000221
[7] 3,000,221
[7] 8,000,221
-------------------------------------------
$ +
[8] 11,000,442
-------------------------------------------
$
```

The first `+` operation uses two numbers of the same size with 'no carry' involved. The second and third `+` operation use numbers of different sizes (still 'no carry'). The fourth `+` operation has

'a carry' that is involved it means that if you start the addition from the right 9+2=11, you get 1 and carry the one to the next digit on the left, and so on. In the fifth `+` operation, the 2 numbers have the same size but the output is larger in size by one (because of the 'carry').

How to proceed?:

- You need to complete the method `__add__` (operator overloading method). Make sure you get the algorithm correct on paper before programming. You can scan the 2 linked-lists from right to left, add the digits and take care of the 'carry'. Hint: 15%10 will give you 5 (we are working in base 10) and 15//10 will give you 1 (the carry).
- Implement your method step by step following the tests example above. There will be partial credit for each successful tests.
- Hint: As a reminder the method `insertFirst` can be used to create your new big number by inserting the calculated digit at the beginning of the linked-list (to make sure that the ordering of the final number is correct).

Task-3 Operation - [15pts]

You need to perform the subtraction of two big numbers. For the time being (for this task), we will consider that the two big numbers are positive (including zero). At first, you will also consider that number 1 is greater than number 2. Here are three test examples by increasing level of difficulty.

```
Welcome to the RPN calculator ('quit' to quit)
-----------------------------------------------
$ 123456
[6] 123,456
-----------------------------------------------
$ 111
[6] 123,456
[3] 111
-----------------------------------------------
$ -
[6] 123,345
-----------------------------------------------
$ 888
[6] 123,345
[3] 888
-----------------------------------------------
$ -
[6] 122,457
-----------------------------------------------
$ 121413
[6] 122,457
[6] 121,413
-----------------------------------------------
$ -
[4] 1,044
-----------------------------------------------
```

The first - operation does not involve a 'carry', the second - operation has a 'carry', and in the third - operation the output is smaller in size than the the original big numbers.

How to proceed?:

- You need to complete the method `__sub__` (operator overloading method). Make sure you get the algorithm correct on paper before programming. You can scan the 2 linked-lists from right to left, subtract the digits and take care of the 'carry'. Hint: let us compute (36-17) by steps: (i) perform (6-7)=-1; (ii) since negative add 10 (-1+10)=9 (9 will be your first utmost right digit) and carry 1 to the next digit; (iii) perform (3-(1+carry))=1 (1 will be your next digit on the left); (iv) result is 19.
- Implement your method step by step following the example tests above. There will be partial credit for each successful tests.
- You may want to use the `trimFront` method on the result to remove any leading zeros if any.
- For this task, you also need to consider the subtraction between two big (positive) numbers where the first number is smaller. The result will then be negative. Hints: At the beginning of the __sub__ method, the __gt__ method could come handy here, if the big1 number is smaller than big2, just call the method `__sub__` 'in reverse', and return with new big number after changing its sign to negative. Example of execution:

```
Welcome to the RPN calculator ('quit' to quit)
------------------------------------------------
$ 12345
[5] 12,345
------------------------------------------------
$ 35235262462
[5] 12,345
[11] 35,235,262,462
------------------------------------------------
$ -
[11] -35,235,250,117
------------------------------------------------
$
```

Task-4 Product * [15pts]

For this task, and in order to make things easier, we will proceed in three steps

1- Scale operation

The program is expecting one big number and one factor (positive number form 0 to 9). It will perform the multiplication, and return the new big number. This operation can be performed using your RPN by entering at the prompt `s3` (to scale by 3), `s6` to scale by 6, etc. Some examples:

```
Welcome to the RPN calculator ('quit' to quit)
-----------------------------------------------
$ 123
[3] 123
-----------------------------------------------
$ s9
[4] 1,107
-----------------------------------------------
$ s3
[4] 3,321
-----------------------------------------------
$ s0
[1] 0
-----------------------------------------------
$
```

How to proceed?:

- You need to complete the method `scale` method (called by the `StackCalc` class). Again, you can scan your link list from right to left, proceed by step to account for the "carry".

2- Product operation between two positive numbers

Multiplication of two big positive numbers, examples:

```
Welcome to the RPN calculator ('quit' to quit)
-----------------------------------------------
$ 123456789
[9] 123,456,789
-----------------------------------------------
$ 123456
[9] 123,456,789
[6] 123,456
-----------------------------------------------
$ *
[14] 15,241,481,342,784
-----------------------------------------------
$ 123456789123456789123456789
[14] 15,241,481,342,784
```

```
[27] 123,456,789,123,456,789,123,456,789
-----------------------------------------
$ *
[40] 1,881,664,348,065,185,308,641,185,306,759,520,960,576
-----------------------------------------
$ 999999999999999999999999999999999999999999999999999
[40] 1,881,664,348,065,185,308,641,185,306,759,520,960,576
[53] 99,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999
-----------------------------------------
$ *
[93] 188,166,434,806,518,530,864,118,530,675,952,096,057,599,999,999,999,998,118,335,651,934,814,691,3
-----------------------------------------
$
```

How to proceed?:

- You need to complete the method `__mul__` (operator overloading method). Make sure you get the algorithm correct on paper before programming. Hint: You can use the method `scale` and `__add__` to perform successive (shifted) additions.

3- Generalization

Make sure that if one of the big numbers (but not both) is negative, so is the result.

Task-5 Generalization of +,- [10pts]

Your addition and subtraction should also work with negative numbers. You can handle the general cases in methods `__add__` and `__sub__` by taking advantage of this table:

sign of $b1$	+	+	-	-
sign of $b2$	+	-	+	-
$b1 + b2$	$\|b1\| + \|b2\|$	$\|b1\| - \|b2\|$	$\|b2\| - \|b1\|$	$-(\|b2\| + \|b1\|)$
$b1 - b2$	$\|b1\| - \|b2\|$	$\|b1\| + \|b2\|$	$-(\|b1\| + \|b2\|)$	$\|b2\| - \|b1\|$

Basically, you just need to add a bunch of if condition at the start of both methods (which are just handling positive numbers) and redirect the operations as presented in the table.

Some few examples:

```
Welcome to the RPN calculator ('quit' to quit)
-----------------------------------------
$ 987654321
[9] 987,654,321
-----------------------------------------
$ -123456789
[9] 987,654,321
[9] -123,456,789
-----------------------------------------
$ -
```

```
[10] 1,111,111,110
-------------------------------------------
$ -1111111120
[10] 1,111,111,110
[10] -1,111,111,120
-------------------------------------------
$ +
[2] -10
-------------------------------------------
$ -100
[2] -10
[3] -100
-------------------------------------------
$ +
[3] -110
-------------------------------------------
$ 100
[3] -110
[3] 100
-------------------------------------------
$ +
[2] -10
-------------------------------------------
$ 10
[2] -10
[2] 10
-------------------------------------------
$ +
[1] 0
-------------------------------------------
$
```

Extra Credit (bonus)

You can earn another 10 additional points by performing the true division. Indicate what you have done in comments as well. The TAs will not provide help for extra-credit.