

Feedback on #1A: Programming is available:









Look under a **new** assignment called
Assignment #1: Programming feedback
on connex.

 CSC 225: 201709 A01: Assignments

Assignment List

Select an assignment to view details, start working or edit you work.

Assignment title

-   [Assignment 2B: Written questions](#)
-   [Assignment 2A: Programming](#)
-   [Assignment #1: Programming feedback](#)
-   [Lab #4](#)

The grade I will record is given by the TOTAL line:

```
MARK for question 1 (readBigInteger) 12 out of 15
MARK for question 2 (printBigInteger) 15 out of 15
MARK for question 3 (reverse) 30 out of 30
MARK for question 4 (plus_plus) 15 out of 15
MARK for question 5 (plus) 25 out of 25
MARKS: -5 taken off: Your code does not have enough
comments.
```

The final total is the sum of these (or 0 is the sum is negative) +10:

TOTAL: 102

I am giving you a chance to resubmit Assignment #1A because many students did not do very well.

Resubmissions are due at the same time as assignment #2A.

Your current mark is p_1 where

$p_1 = m_1 = 0$ if no submission or not close to compiling
or

$p_1 = m_1 + 10$ where m_1 is the mark according to the grading scheme.

On your resubmission, I will compute a mark m_2 out of 100.

Set $p_2 = m_1 + (m_2 - m_1) * 0.7$

Your mark for the assignment: $\text{Max}\{p_1, p_2\}$

Ex. If $m_1 = 0$ and $m_2 = 100$, $p_2 = 70$.

Make sure you check your lists!

Lists should have valid values for n, start, rear and should be null terminated. Write some checklist code to check your lists.

An error like this:

```
Running Question 4
```

```
QUESTION 4: -----
```

```
Testing plus_plus
```

```
Test the increment method:
```

```
Value should be 1 is      1
```

```
Error- missing null terminator after 1 items.
```

```
MARK for question 4 (plus_plus) 0 out of 15
```

can arise if you forget to increment n.

Some problems with the programs:

Bad indentation.

Not enough comments:

Use comments at the tops of methods to explain their functionality. But also include comments at each step of the algorithm to explain what you are doing.

Don't forget to turn off your debugging messages!

To make the code more readable:

Do not use else with the base case of reverse:

```
if (n==1) return;
```

No else here- it makes deeper indentation unnecessarily.

This is the natural way to express the base case.

Do not put long comment at the ends of lines.

Use white space around blocks of code/comments.

To make it easier to debug:

Do not return if an error condition occurs (e.g. n is 0 in reverse or a list ends prematurely). Ideally you should print an error message and exit. Or just let the program crash: if you just return you make it harder to debug your program by burying bugs deeper inside it.

In reverse you could use:

```
if (n < 1)
{
    System.out.println("Error- " +
        "n in reverse is " + n +
        " but should be at least 1.");
    System.exit(0);
}
```

Do not use: `if (n <= 1) return;`

Do not use extra space unnecessarily:

Do not use a lot more variables than you need.

Do not convert data to strings in this program.

Do not store the data in an array before putting it into a list.

Relink the list cells you have instead of making new ones.

Don't ask for new space you do not need (e.g. making two new lists before checking the base case for reverse).

Make the code as fast as you can:

In your loop of reverse: just move one pointer through the list (e.g. `list1.rear`).

Do not use a linear time approach to something that can be done in $O(1)$ time: assigning `n` and `rear` for `list2`, concatenating the two lists together.

Do not use `reverse` in `readBigInteger` or `plus` or `plus_plus`: These can be implemented to run in $O(n)$ time but using `reverse` makes it take $\Theta(n \log n)$ time.

Try to make the code as elegant as possible.

Do not use a method to do a partial task.

E.g. to do some but not all of the work for breaking a list into two sublists.

Use meaningful variable names. Try to avoid using a lot more variables than you need.

You don't need to use "this".

If a pointer should already be null don't reset it to null.

For example:

Concatenating list2 followed by list1.

After you set:

```
rear = list1.rear;
```

Do not do this:

```
rear.next = null;
```

Doing things like this "just in case" can cover up bugs and dig them deeper into your program.

To make code easier to read and check for correctness:

Start reverse with the base case:

```
if (n==1) return;
```

Initialize variables just before you use them instead of at the top of the method.

Do not use l (ell) as a variable name- it looks too similar to the number one.

It is almost always bad style to use

```
while (true)
{
    do some stuff
}
as a loop.
```

Use a for loop instead of a while loop if it is the natural construction.

Loops usually count up and not down.

Exception: in maxSort:

```
for (end= n-1; end >= 1; end--)
```

For loops usually have an initial value assigned.

Don't add unnecessary extra special cases.

For example: the case when $n=2$ in reverse.

Do not use code to bypass your base case:

Instead of

```
if (n >= 2)
    list1.reverse(level+1);
```

Just use:

```
list1.reverse(level+1);
```


The reverse method should not call another method that does the actual recursion.

You do not need a linked list as a parameter.

There is a linked list that is the object associated with the reverse method.

Running out of input is not an error.

Do not exit when we are out of input.

Make sure you return null (and not an empty linked list) when we are out of input.

Do not try to convert the big integer values to integers to add them.

The whole point of the software is that the values could be too big to be stored in one integer.

Do not use math floor/ceiling methods.

You can use:

```
int floor= n/2;
```

```
int ceil= n- n/2;
```

to get the same thing with much less overhead and without converting to doubles.