# Sep / 28 / 2020

## HW1 Sagar’s solution and explanations

* Incremental code (record commits)
* main ()
	+ Exceptions for invalid input : e.g. “-341343”
		- Use exception to make it easier for future changes
	+ Switch statement for operators
		- e.g. switch(op) {

case ‘+’:

 case ‘-’:

 … …

 Default:

 }

* bignum class
	+ Linking : #include “bignum.hpp”
	+ Write public variables/functions ahead of private variables
	+ Pass arguments/parameter using const

Bignum& operator+(const Bignum& other) const;

* + Default initialize
		- Default initialize a vector: a vector with 0 elements

e.g. zero bignum : for number 0

* + - Assignment : Bignum(Bignum&&) = default;
		- Copy : Bignum& operator = (const Bignum&) = delete;

 Deep Copy: copy.number = number

create a new variable with different address

* + - Default move: Bignum& operator=(Bignum&&) = default;
* Different from clone, move will move object from one to another, the previous one no longer valid
	+ Private constructor: Bignum(const uint32\_t num\_digits)
	+ Accessing digit(override indexing operator):

uint8\_t& operator[](uint32\_t idx) -- reference to change

uint8\_t operator[](uint32\_t idx) const; --access index

* + Int vs. uint:
		- uint\_8 element size in the vector
		- uint\_32 to represent the number of digits
	+ Transform : transform each ACIIchar -> int : char - ’0’

std::transform(strnum.begin(). strnum.end(), digits.rbegin(), [&strnum] (const char ch) ){

…… }

* + purge() :
		- Store the number in reverse order: since removing an element from the end ( pop\_back() ) is easier than remove from the beginning
	+ Exception : create a class exception.hpp link to Bignum
		- bignum::parse\_error(strnum)
		- bignum::negative\_number\_error
	+ Operators:
		- Principles: avoid creating extra variables during function
		- Operator compares:

< : comparing total digits number

(keep the invariant through code to ensure no extra digits in comparison)

 lexicographical comparison operator

(from most significant digit to last digit)

* + - Operator + :

While loop for merge two numbers

carry variable use bool to represent for precise

Inner function for carry + number[i] + other[i]

* + - Operator - :
		- Operator \* :

Grading method for multiplication 

Iterate over small number of the two numbers multiplying

large\*small

* + - Operator / and % :

Dividend / divisor = quotient … remainder

Return quotient for / operator, return remainder for % operator

Above is Simplifier than get the remainder by (dividend-quotient\*divisor)

* + gdb to debug
		- compile the executable
		- run the executable via gdb ./bignum + 3 2