# Big Number Library

# Author


- Name: 于子緯
- Grade: 資工111級(大一)
- ID: 40747024S


# Contents

        - argument
        - return value
        - caution
    - Power
        - description
        - argument
        - return value
        - caution
    - Factorial
        - description
        - argument
        - return value
        - caution
    - Permutation
        - description
        - argument
        - return value
        - caution
    - Combination
        - description
        - argument
        - return value
        - caution
    - Print
        - description
        - argument
    - Set
        - description
        - argument
        - return value
        - caution
    - GCD
        - description
        - argument
        - return value
        - caution
    - LCM
        - description
        - argument
        - return value
        - caution
    - Extra-isPrime
        - description
        - argument
        - return value
    - Extra-SQRT
        - description

# Brief Introduction

this library can help you handle big number operation.
featuring with supporting negative number, and some extra function.

# Structure

```
typedef struct _BigNum_ {
    // size of data array, each element in array stores at most 9999
    int32_t size, *data;
    // check if this is negative
    bool nega;
} BigNum;
```

# Usage

all in `README.md`

# Functions

## Initialization

**description**

initial big number with assigned bit.

#### argument

```
int32_t bigNumInit( BigNum *pNum, int32_t n )
```

`n` indicate set `pNum` to `n` bits.

#### return value

return -1 if fail to initialize, or 0.

## Free

#### description

free a big number.

#### argument

```
void bigNumFree( BigNum *pObj )
```

free `pObj` .

## Comparison

#### description

compare two big number, equal, less than, or greater than.

#### argument

```
int32_t bigNumCmp( const BigNum *pObj1, const BigNum *pObj2 )
```

compare two big number `pObj1` , `pObj2` .

#### return value

return 0 indicates equal, -1 `obj1` is less than `obj2` , 1 `obj1` is greater than `obj2`

## Addition

#### description

addition between two big numbers.

#### argument

```
int32_t bigNumAdd( BigNum *pAns, const BigNum *pObj1, const BigNum *pObj2 )
```

`obj1` add `obj2` , and result store in `pAns` .

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the addition.

## Subtraction

#### description

subtraction between two big numbers.

#### argument

```
int32_t bigNumSub( BigNum *pAns, const BigNum *pObj1, const BigNum *pObj2 )
```

`obj1` subtract `obj2`, and result store in `pAns`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the subtraction.

## Multiplication

#### description

multiplication between two big numbers.

#### argument

```
int32_t bigNumMul( BigNum *pAns, const BigNum *pObj1, const BigNum *pObj2 )
```

`obj1` multiply `obj2`, and result store in `pAns`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the multiplication.

## Division

#### description

division between two big numbers.

#### argument

```
int32_t bigNumDiv( BigNum *pQuotient, BigNum *pRemainder
, const BigNum *pObj1, const BigNum *pObj2 )
```

`obj1` divide `obj2`, and quotient store in `pQuotient`, remainder store in `pRemainder`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pQuotient`, `pRemainder` may change, adaptive with the result of the division.

## Power

#### description

power operation between two big numbers.
powered by **Fast exponentiation algorithm**, time complexity $\Theta(\log N)$, where $N$ is the exponent.

#### argument

```
int32_t bigNumPow( BigNum *pAns, const BigNum *pObj1, const BigNum *pObj2 )
```

`obj1` to the power `obj2`, and result store in `pAns`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the power operation.

## Factorial

#### description

factorial of a big number.

#### argument

```
int32_t bigNumFactorial( BigNum *pAns, const BigNum *pObj )
```

compute factorial of `obj1`, and result store in `pAns`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the factorial operation.

## Permutation

#### description

compute permutations between two big numbers.

#### argument

```
int32_t bigNumPermutation( BigNum *pAns, const BigNum *pN, const BigNum *pK )
```

compute `pK` -permutations of `pN` , and result store in `pAns` .

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the operation.

## Combination

#### description

compute Combinations between two big numbers.

#### argument

```
int32_t bigNumCombination( BigNum *pAns, const BigNum *pN, const BigNum *pK )
```

compute `pK` -combinations of `pN` , and result store in `pAns` .

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the operation.

## Print

#### description

print big number decimal, binary, hexadecimal.

I made an extra function to print big number in**random number system(2~16)** .
link: Extra-bigNumPrint.

#### argument

```
void bigNumPrintDec( const BigNum *pObj )
void bigNumPrintBin( const BigNum *pObj )
void bigNumPrintHex( const BigNum *pObj )
```

print `pObj` decimal, binary, hexadecimal.

## Set

### description

set big number by decimal, binary, hexadecimal string.

### argument

```
int32_t bigNumSetDec( BigNum *pObj, const char *decimal )
int32_t bigNumSetBin( BigNum *pObj, const char *binary )
int32_t bigNumSetHex( BigNum *pObj, const char *hex )
```

set `pObj` by decimal, binary, hexadecimal string.

### return value

return -1 if fail, or 0.

### caution

`pObj` **has to be initialized**.

## GCD

### description

greatest common divisor of two big numbers.

### argument

```
int32_t bigNumGCD( BigNum *pAns, const BigNum *pObj1, const BigNum *pObj2 )
```

compute gcd of `pObj1` and `pObj2`, and result store in `pAns`.

### return value

return -1 if fail, or 0.

### caution

the size of `pAns` may change, adaptive with the result of the operation.

## LCM

### description

least common multiple of two big numbers.

### argument

```
int32_t bigNumLCM( BigNum *pAns, const BigNum *pObj1, const BigNum *pObj2 )
```

compute lcm of `pObj1` and `pObj2`, and result store in `pAns`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the operation.

## Extra-isPrime

#### description

determine if the big number is prime.

#### argument

```
int32_t isPrime( const BigNum *pObj )
```

determine if `pObj` is prime.

#### return value

return 1 if `pObj` is prime, or 0.

## Extra-SQRT

#### description

compute the square root(floor integer) of a big number.
powered by **Binary Search**, time complexity $\Theta(\log N)$ , where $N$ is the value of `pObj` down below.

#### argument

```
int32_t bigNumSQRT( BigNum *pAns, const BigNum *pObj )
```

compute the square root of `pObj` , and result store in `pAns` .

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the operation.

## Extra-Nth Fibonacci Number

#### description

compute the square root(floor integer) of a big number.
powered by **Matrix Fast exponentiation algorithm**, time complexity $\Theta(\log N)$ , where $N$ is the value of `pN` down below.

#### argument

```
int32_t NthFibonacci( BigNum *pAns, const BigNum *pN )
```

compute Nth Fibonacci number, and result store in `pAns`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the operation.

## Extra-Log

#### description

compute the Log of a big number based the other big number.
powered by **Binary Search (and Fast EXponentiation algorithm because of using bigNumPow in this funciton)**, time complexity $\O(\log N \log M)$ , where $N$ is the value of `pX` down below.

#### argument

```
int32_t bigNumLog( BigNum *pAns, const BigNum *pB, const BigNum *pX )
```

compute the log of `pX` based `pB`, and result store in `pAns`.

#### return value

return -1 if fail, or 0.

#### caution

the size of `pAns` may change, adaptive with the result of the operation.

## Extra-isPalindrome

#### description

determine if the big number is palindrome.

#### argument

```
int32_t isPalindrome( const BigNum *pNum )
```

determine if `pNum` is prime.

#### return value

return 1 if `pNum` is prime, or 0.

## Extra-bigNumPrint

#### description

Print big number based on random number system(2 ~ 16)

**argument**

```
int32_t bigNumPrint( const BigNum *pObj, int32_t n )
```

print `pObj` in number system of base n.

**return value**

return -1 if fail, or 0.

# Appendix

預計要實作一個 `mymacro.h`
僅完成三個函式，目的在於能夠幫助使用者除錯

```
1 #define bigNumSetDec(x,y) bigNumSetDec(x, y, __FILE__, __LINE__, __func__)
2 #define bigNumSetBin(x,y) bigNumSetBin(x, y, __FILE__, __LINE__, __func__)
3 #define bigNumSetHex(x,y) bigNumSetHex(x, y, __FILE__, __LINE__, __func__)
```

在使用者進行 `SET` 時，若發生錯誤(如使用者沒有init足夠的空間)，可以依靠 `__FILE__`, `__LINE__`, `__func__` 輸出debug訊息。