Big Number

```
// Hexadecimal Big Integer with everything hard-coded lol
#include <bits/stdc++.h>
#define MAX DIGIT 300 // max digit allowed is 300
void Hex2Int(int *result, char *str) {
    const int n = strlen(str);
    memset(result, 0, sizeof(int)*MAX_DIGIT);
    for (int i = n - 1; i >= 0; i--) {
        if (str[i] >= '0' && str[i] <= '9') {</pre>
            result[n - i - 1] = str[i] - '0';
        }
        else if (str[i] >= 'a' && str[i] <= 'z') {</pre>
            result[n - i - 1] = str[i] - 'a' + 10;
        }
        else if (str[i] >= 'A' && str[i] <= 'Z') {</pre>
            result[n - i - 1] = str[i] - 'A' + 10;
        }
    }
// interface functions
void Print(int *result) {
    int i = MAX DIGIT - 1;
    while (i > 0 && result[i] == 0) {
        i--; // skip leading zero, if any
    while (i >= 0) {
        if (result[i] == 10) {
            std::cout << 'a';</pre>
        }
        else if (result[i] == 11) {
            std::cout << 'b';</pre>
        }
        else if (result[i] == 12) {
            std::cout << 'c';</pre>
        }
        else if (result[i] == 13) {
            std::cout << 'd';</pre>
        }
        else if (result[i] == 14) {
```

```
std::cout << 'e';</pre>
        }
        else if (result[i] == 15) {
            std::cout << 'f';</pre>
        }
        else {
            std::cout << result[i];</pre>
        }
        i--;
    std::cout << "\n";</pre>
void Assign(int *result, int *source) {
    // Assign function, assign source to result
    memset(result, 0, sizeof(int)*MAX_DIGIT);
    for (int i = MAX_DIGIT - 1; i >= 0; i--) {
        result[i] = source[i];
    }
// arithmetic operators
int Compare(int *a, int *b) {
    // Compare function, if a > b return 1,
    // if a < b return -1, if a == b return 0</pre>
    int i = MAX DIGIT - 1;
    while (i > 0 && a[i] == b[i]) {
        i--;
    <u>if</u> (a[i] - b[i] == 0) {
        return 0;
    return (a[i] - b[i] > 0) ? 1 : -1;
void Add(int *result, int *a, int *b) {
    memset(result, 0, sizeof(int)*MAX DIGIT);
    for (int i = 0, carry = 0; i < MAX_DIGIT; i++) {</pre>
        result[i] = a[i] + b[i] + carry;
        carry = result[i] / 16;
        result[i] %= 16;
    }
}
```

```
bool Sub(int *result, int *a, int *b) {
    // Substraction, if a > b result= a - b return sign = false
    // if a < b result = b - a return sign = true</pre>
    memset(result, 0, sizeof(int)*MAX_DIGIT);
    bool sign = false;
    if (Compare(a, b) == 1) {
        for (int i = 0, borrow = 0; i < MAX DIGIT; i++) {</pre>
            result[i] = a[i] - b[i] - borrow;
            // result < 0 means not enough, need to borrow</pre>
            if (result[i] < 0) {</pre>
                borrow = 1;
                result[i] += 16; // lend result 16
            }
            // result >= 0 means enough or equal
            else if (result[i] >= 0) {
                borrow = 0;
            }
        }
        // a - b > 0 is positive, sign remains false
        return sign;
    }
    // case 2. a < b
    else if (Compare(a, b) == -1) {
        for (int i = 0, borrow = 0; i < MAX_DIGIT; i++) {</pre>
            result[i] = b[i] - a[i] - borrow;
            // result < 0 means not enough, need to borrow</pre>
            if (result[i] < 0) {</pre>
                borrow = 1;
                result[i] += 16; // lend result 16
            }
            // result >= 0 means enough or equal
            else if (result[i] >= 0) {
                borrow = 0;
            }
        }
        // a - b < 0 is negative, sign changed into true</pre>
        sign = true;
        return sign;
    // case 3. a == b
    return 0;
```

```
void Mul(int *result, int *a, int *b) {
    // Multiplication, result = a * b
   memset(result, 0, sizeof(int)*MAX_DIGIT);
    for (int i = 0; i < MAX_DIGIT; i++) {</pre>
       if (a[i] == 0) {
           continue;
        }
       for (int j = 0; i + j < MAX_DIGIT; j++) {</pre>
           result[i + j] += a[i] * b[j];
        }
    }
    // if result[i] >= 16 means overflow, need to add a carry
    for (int i = 0, carry = 0; i < MAX DIGIT; i++) {</pre>
       result[i] += carry;
       carry = result[i] / 16;
       result[i] %= 16;
   }
void Div(int *divide, int *remain, int *a, int *b) {
   memset(divide, 0, sizeof(int)*MAX_DIGIT);
   memset(remain, 0, sizeof(int)*MAX_DIGIT);
   while (Sub(remain, a, b) == false) {
       Assign(a, remain); // a = remain
        divide[0]++;
       for (int i = 0; i < MAX_DIGIT; i++) {</pre>
           // check for carry
           if (divide[i] == 16) {
               divide[i] = 0; // set it to 0
               divide[i + 1] += 1; // add the carry to next digit
           }
       }
   }
}
int main() {
    char *str1 = (char*)calloc(MAX_DIGIT, sizeof(char)); // input a
    char *str2 = (char*)calloc(MAX DIGIT, sizeof(char)); // input b
   // store str1 and str2 in array a and b
    int *a = (int*)calloc(MAX DIGIT, sizeof(int));
    int *b = (int*)calloc(MAX_DIGIT, sizeof(int));
```

```
// store the result of addition, substraction,
// multiplication, quotient, and remainder
int *add = (int*)calloc(MAX_DIGIT, sizeof(int));
int *sub = (int*)calloc(MAX_DIGIT, sizeof(int));
int *mul = (int*)calloc(MAX_DIGIT, sizeof(int));
int *quo = (int*)calloc(MAX_DIGIT, sizeof(int));
int *rem = (int*)calloc(MAX_DIGIT, sizeof(int));
// sample input
/* input constrains:
   (2) len(a) <= len(b) + 5
 * otherwise the result would either be incorrect
* or exceeded constant execution time requirement. */
std::cout << "a = " << str1 << "\n";</pre>
std::cout << "b = " << str2 << "\n";</pre>
// convert hex input char array into decimal int array
Hex2Int(a, str1);
Hex2Int(b, str2);
// check if we read the input string correctly
std::cout << "read a = "; Print(a);</pre>
std::cout << "read b = "; Print(b);</pre>
// testing addition
Add(add, a, b);
std::cout << "a + b = "; Print(add);</pre>
// testing substraction
Sub(sub, a, b);
std::cout << "a - b = "; Print(sub);</pre>
// testing multiplication
Mul(mul, a, b);
std::cout << "a * b = "; Print(mul);</pre>
// testing integer divsion
Div(quo, rem, a, b);
std::cout << "a / b = "; Print(quo);</pre>
// testing modulus
// Div(quo, rem, a, b);
std::cout << "a % b = "; Print(a);</pre>
free(str1); free(str2); free(a); free(b);
free(add); free(sub); free(mul); free(quo); free(rem);
return 0;
```