

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') {
            result[n - i - 1] = str[i] - '0';
        }
        else if (str[i] >= 'a' && str[i] <= 'z') {
            result[n - i - 1] = str[i] - 'a' + 10;
        }
        else if (str[i] >= 'A' && str[i] <= 'Z') {
            result[n - i - 1] = str[i] - 'A' + 10;
        }
    }
}

// interface functions
void Print(int *result) {
    // Print function, print out the big number in hex
    int i = MAX_DIGIT - 1;
    while (i > 0 && result[i] == 0) {
        i--; // skip leading zero, if any
    }
    while (i >= 0) {
        // print out 10~15
        if (result[i] == 10) {
            std::cout << 'a';
        }
        else if (result[i] == 11) {
            std::cout << 'b';
        }
        else if (result[i] == 12) {
            std::cout << 'c';
        }
        else if (result[i] == 13) {
            std::cout << 'd';
        }
        else if (result[i] == 14) {
```

```

        std::cout << 'e';
    }
    else if (result[i] == 15) {
        std::cout << 'f';
    }
    else {
        // print out 0~9
        std::cout << result[i];
    }
    i--;
}
std::cout << "\n";
}

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
    int i = MAX_DIGIT - 1;
    while (i > 0 && a[i] == b[i]) {
        i--;
    }
    if (a[i] - b[i] == 0) {
        return 0;
    }
    return (a[i] - b[i] > 0) ? 1 : -1;
}

void Add(int *result, int *a, int *b) {
    // Addition, result = a + b
    memset(result, 0, sizeof(int)*MAX_DIGIT);
    for (int i = 0, carry = 0; i < MAX_DIGIT; i++) {
        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
    memset(result, 0, sizeof(int)*MAX_DIGIT);
    bool sign = false;
    // case 1. a > B
    if (Compare(a, b) == 1) {
        for (int i = 0, borrow = 0; i < MAX_DIGIT; i++) {
            result[i] = a[i] - b[i] - borrow;
            // result < 0 means not enough, need to borrow
            if (result[i] < 0) {
                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++) {
            result[i] = b[i] - a[i] - borrow;
            // result < 0 means not enough, need to borrow
            if (result[i] < 0) {
                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
        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++) {
        if (a[i] == 0) {
            continue;
        }
        for (int j = 0; i + j < MAX_DIGIT; j++) {
            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++) {
        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) {
        // remain = a - b
        Assign(a, remain); // a = remain
        divide[0]++;
        for (int i = 0; i < MAX_DIGIT; i++) {
            // check for carry
            if (divide[i] == 16) {
                // if any element == 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));

```

