bigInteger.cpp

```cpp
// Hexadecimal Big Integer program with everything hard-coded, by 110064533
#include <bits/stdc++.h>
#define MAX_DIGIT 300 // currently, 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 bid 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;
```

```c
    }
    // 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
            }
        }
    }
}
```

```cpp
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:
     *   (1) a >= b
     *   (2) len(a) <= len(b) + 5
     *   otherwise the result would either be incorrect
     *   or exceeded constant execution time requirement. */
    strcpy(str1, "f1245AB3341Ff3461818881767676819EE");
    strcpy(str2, "ffa24387539639853800bbeCbcb494990");
    std::cout << "string a = " << str1 << "\n";
    std::cout << "string b = " << str2 << "\n";

    // 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 << "integer a = "; Print(a);
    std::cout << "integer b = "; Print(b);
    // testing addition
    Add(add, a, b);
    std::cout << "a + b = "; Print(add);
    // testing substraction
    Sub(sub, a, b);
    std::cout << "a - b = "; Print(sub);
    // testing multiplication
    Mul(mul, a, b);
    std::cout << "a * b = "; Print(mul);
    // testing integer divsion
    Div(quo, rem, a, b);
    std::cout << "a / b = "; Print(quo);
    // testing modulus
    // Div(quo, rem, a, b);
    std::cout << "a % b = "; Print(a);

    free(str1);
    free(str2);
    free(a);
    free(b);
    free(add);
    free(sub);
    free(mul);
    free(quo);
    free(rem);

    return 0;
}
```

```
> Executing task: g++ -g d:\C++\bigNumber3\bigInteger.cpp -o bigInteger.exe <

PS D:\C++> ./bigInteger
string a = f1245AB3341Ff3461818881767676819EE
string b = ffa24387539639853800bbeCbcb494990
integer a = f1245ab3341ff3461818881767676819ee
integer b = ffa24387539639853800bbecbcb494990
a + b = 1011e7eeba95956de6b9893d63332b1637e
a - b = e12a367abee68fadc4987c589b9c1ed05e
a * b = f0cc0ef5e2f7d593719ce283c6efb373d86a14d50f9f5c5ba42a6bae39ff8d173e0
a / b = f
a % b = 17c3b6455c31d593397d7e9767e1cca7e
PS D:\C++>
```