**★★★★☆**

**題組：Problem Set Archive** **with Online Judge**

**題號：288: Arithmetic Operations With Large Integers**

**解題者：李承諺**

**解題日期：2025年5月15日**

**題意：**

給定一串數學式，算出答案。

**題意範例：**

1+2\*\*3-4\*5 -> -11

1+2\*\*3\*4+5 -> 38

**解法：** 大數運算，用陣列存數字

**解法範例：**

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。對於每一個數字，我們都用一個陣列來儲存，例如我們要儲存12345543211234554321，將會用下圖的方式來儲存數字。

我們用 Bignum(12345543211234554321) 表示這個數字。

接下來，讀入每一行的數學式後，我們用一個陣列儲存每一個數字，另

一張含有 螢幕擷取畫面, 行, Rectangle, 字型 的圖片

AI 產生的內容可能不正確。一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

AI 產生的內容可能不正確。一個陣列儲存符號。

接下來，根據符號的優先級處理運算，對於op[i]來說，我們會把運算完

的結果儲存到num[i]，並把num[i+1]刪掉。當然，也要把op[i]刪掉。刪

到最後面，num[0]就是我們要的結果。

最後，我們實際操作一次 1+2\*\*3-4\*5 = -11 。

一張含有 螢幕擷取畫面, 文字, 行, 圖表 的圖片

AI 產生的內容可能不正確。一張含有 螢幕擷取畫面, 文字, 數字, 平行 的圖片

AI 產生的內容可能不正確。

**討論：**

1. : 此題因為每一個數字最多有1000位，且最多有100個運算，所以如

果用 k 格陣列存放 k 個 digit，如果做高指數的運算，可能造成TLE。

1. : 這題用C++解非常痛苦，但如果用Python實作，只需要5行程式碼，所以選對程式語言非常重要。

**程式：**

#include <bits/stdc++.h>

using namespace std;

#define N 10000

#define width 4

class Bignum {

    public:

        Bignum(string str = "");

        Bignum operator + (Bignum &other);

        Bignum operator - (Bignum &other);

        Bignum operator \* (Bignum &other);

        Bignum operator ^ (Bignum &other);

        bool operator < (Bignum &other);

        void operator \*= (Bignum &other);

        friend ostream& operator<<(ostream &out, Bignum &other);

        bool sign;           // 0: 正數, 1: 負數

        int digit[1001];    // 每4位為一單位

        int cnt;           // 有效位數

        void CountDigit();

        bool IsZero();

        void decrement();

};

Bignum::Bignum(string str) : sign(0), cnt(0) {

    memset(digit, 0, sizeof(digit));

    if (str[0] == '-') {

        sign = 1;

        str.erase(0, 1);

    }

    for (int i = (int)str.size(); i > 0; i -= width) {

        int start = max(0, i - width);

        int size = i - start;

        digit[cnt++] = stoi(str.substr(start, size));

    }

}

void Bignum::CountDigit() {

    for (int i = 1000; i >= 0; --i) {

        if (digit[i]) {

            cnt = i + 1;

            return;

        }

    }

    cnt = 1;

}

bool Bignum::IsZero() {

    return (cnt == 1 && digit[0] == 0);

}

void Bignum::decrement() {

    if (cnt <= 0) return;

    digit[0]--;

    for (int i = 0; i < cnt; ++i) {

        if (digit[i] < 0) {

            digit[i] += N;

            digit[i + 1]--;

        }

        else break;

    }

    CountDigit();

}

void Bignum::operator\*=(Bignum &other) {

    \*this = (\*this) \* other;

}

bool Bignum::operator < (Bignum &other) {

    if (sign != other.sign) {

        return sign > other.sign;

    }

    if (!sign) {

        if (cnt != other.cnt){

            return cnt < other.cnt;

        }

        for (int i = cnt - 1; i >= 0; --i){

            if (digit[i] != other.digit[i]) {

                return digit[i] < other.digit[i];

            }

        }

    }

    else {

        if (cnt != other.cnt) {

            return cnt > other.cnt;

        }

        for (int i = cnt - 1; i >= 0; --i){

            if (digit[i] != other.digit[i]) {

                return digit[i] > other.digit[i];

            }

        }

    }

    return false;

}

Bignum Bignum::operator + (Bignum &other) {

    Bignum result;

    if (sign == other.sign) {

        result.sign = sign;

        for (int i = 0; i < max(cnt, other.cnt); ++i) {

            result.digit[i] += digit[i] + other.digit[i];

            result.digit[i + 1] += result.digit[i] / N;

            result.digit[i] %= N ;

        }

    }

    else {

        if(!sign){

            Bignum tmp = \*this;

            tmp.sign = other.sign;

            result = other-tmp;

        }

        else{

            Bignum tmp = other;

            other.sign = sign ;

            result = \*this - tmp;

        }

    }

    result.CountDigit();

    return result;

}

Bignum Bignum::operator - (Bignum &other) {

    Bignum result;

    if (\*this < other) {

        result = other - \*this;

        result.sign = 1;

        return result;

    }

    for (int i = 0; i < cnt; ++i) {

        result.digit[i] = digit[i] - other.digit[i];

        if (result.digit[i] < 0) {

            result.digit[i] += N;

            digit[i + 1]--;

        }

    }

    result.CountDigit();

    return result;

}

Bignum Bignum::operator \* (Bignum &other) {

    Bignum result;

    result.sign = (sign != other.sign);

    for (int i = 0; i < cnt; ++i) {

        for (int j = 0; j < other.cnt; ++j) {

            result.digit[i + j] += digit[i] \* other.digit[j];

        }

    }

    for (int i = 0; i < cnt + other.cnt; ++i) {

        result.digit[i + 1] += result.digit[i] / N;

        result.digit[i] %= N;

    }

    result.CountDigit();

    return result;

}

Bignum Bignum::operator ^ (Bignum &other) {

    Bignum result("1");

    Bignum exp = other;

    while (!exp.IsZero()) {

        result \*= \*this;

        exp.decrement();

    }

    return result;

}

ostream& operator << (ostream &out, Bignum &other) {

    if (other.cnt == 0 || (other.cnt == 1 && other.digit[0] == 0)){

        return out << 0;

    }

    if (other.sign){

        out << '-';

    }

    out << other.digit[other.cnt - 1];

    for (int i = other.cnt - 2; i >= 0; --i){

        out << setw(width) << setfill('0') << other.digit[i];

    }

    return out;

}

int main() {

    string input;

    while (cin >> input) {

        string temp = input;

        vector<Bignum> number;

        vector<string> op;

        char \*ptr = strtok((char\*)input.c\_str(), "+-\*");

        while (ptr) {

            number.push\_back(Bignum(ptr));

            ptr = strtok(NULL, "+-\*");

        }

        input = temp;

        ptr = strtok((char\*)input.c\_str(), "0123456789");

        while (ptr) {

            op.push\_back(ptr);

            ptr = strtok(NULL, "0123456789");

        }

        for (int i = op.size() - 1; i >= 0; --i) {

            if (op[i] == "\*\*") {

                number[i] = number[i] ^ number[i + 1];

                op.erase(op.begin() + i);

                number.erase(number.begin() + i + 1);

            }

        }

        for (int i = 0; i < op.size(); ++i) {

            if (op[i] == "\*") {

                number[i] = number[i] \* number[i + 1];

                op.erase(op.begin() + i);

                number.erase(number.begin() + i + 1);

                --i;

            }

        }

        for (int i = 0; i < op.size(); ++i) {

            if (op[i] == "+") number[i] = number[i] + number[i + 1];

            else if (op[i] == "-") {

                if(number[i].sign != number[i+1].sign){

                    if(!number[i].sign){

                        number[i+1].sign = number[i].sign;

                        number[i] = number[i] + number[i+1];

                    }

                    else{

                        number[i+1].sign = number[i].sign = 0 ;

                        number[i] = number[i] + number[i+1];

                        number[i].sign = 1;

                    }

                }

                else{

                    number[i] = number[i] - number[i + 1];

                }

            }

            op.erase(op.begin() + i);

            number.erase(number.begin() + i + 1);

            --i;

        }

        cout << number[0] << endl;

    }

    return 0;

}

// 98654\*192363-98765\*\*128+5\*\*987

// 0\*\*0

// 98654\*192363-98765\*\*128+5\*\*987