1. Длинная арифметика

Длинная арифметика — это набор программных средств (структуры данных и алгоритмы), которые позволяют работать с числами больших величин, чем это возможно для стандартных типов данных.

Работая с «длинными» числами, чьи значения выходят за диапазон стандартных типов, нам придётся для них рассмотреть несколько важных операций, которые для базовых типов реализованы средствами языка[1]:

1. Ввод и вывод;
2. Сложение;
3. Вычитание;
4. Сравнение;
5. Полудлинное умножение – умножение «длинного» числа на «короткое»;
6. Длинное умножение – умножение «длинного» на «длинное»;
7. Полудлинное деление – деление «длинного» числа на «короткое»;
8. Длинное деление – умножение «длинного» на «длинное».

Рассмотрим несколько представлений «длинных» целых чисел с помощью массива.

1.1 Упрощенная модель «число=строка»

Является частным случаем следующей модели 10^n при n=1.

Первая наиболее интуитивно понятная модель – представление длинных чисел с помощью массива символов (строки). В данной модели нам нужно лишь посимвольно производить хорошо известные нам операции с разрядами десятичного числа.

ВАЖНО: Данная модель является наиболее простым представлением длинных чисел, хотя и **медленным** в работе. Она удобна для решения задач с большим TIMELIMIT, а также для улучшения понимания более сложных моделей. Также её хорошо использовать, если, к примеру, нужно определить, встречается ли в записи числа 3 идущих подряд нуля и т.д. – это достаточно просто выполняется строковыми функциями.

Задача №1. **Золото племени АББА (acmp.ru)**

(Время: 1 сек. Память: 16 Мб)

Главный вождь племени Абба не умеет считать. В обмен на одну из его земель вождь другого племени предложил ему выбрать одну из трех куч с золотыми монетами. Но вождю племени Абба хочется получить наибольшее количество золотых монет. Помогите вождю сделать правильный выбор!

Входные данные

Три натуральных числа через пробел. Каждое из чисел не превышает 10^{100}.

Выходные данные

---

[1] В некоторых языках и компиляторах - Python, Java, PascalABC (тип **biginteger**) и т.д. длинная арифметика также реализована средствами языка. Однако в Python за всё приходится платить скоростью выполнения действий, Java долго инициализирует библиотеки. В обоих этих языках имеются ограничения на размер «длинных чисел», который может оказаться недостаточным для конкретной задачи. PascalABC не включен в список разрешенных компиляторов для олимпиад по информатике. Для решения не олимпиадных задач с длинной арифметикой в C++ имеется множество реализаций, таких как GMP и т.д.

Одно целое число — максимальное количество монет, которые может взять вождь.

Примеры

№	Ввод	Вывод
1	5 7 3	7
2	987531 234 86364	987531
3	189285 283 4958439238923098349024	49584392389230983 49024

Решение

Данную задачу можно решить без алгоритмов длинной арифметики, используя лишь строковые функции. Из двух чисел больше то, в котором разрядов больше. Если количество разрядов совпадает, то сравнение чисел эквивалентно лексикографическому сравнению строк.

```
#include <iostream>
using namespace std;
string smax(string x, string y){
	int m = x.size(), n = y.size();
	if(m > n) return x;
	if(m < n) return y;
	return max(x, y);
}
int main(){
	string a, b, c;
	cin >> a >> b >> c;
	cout << smax(a, smax(b, c));
}
```

1.1.1 Сложение беззнаковых длинных целых

Чтение и вывод чисел в такой модели реализуется через строковый тип и в дополнительном описании не нуждается. Рассмотрим сложение двух неотрицательных чисел, каждое из которых сохранено в виде последовательности цифр.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	перенос					0	0	0	0	1	1	1	1	1	1	1	0	
2	число #1					2	6	7	5	8	9	6	3	2	7	3	5	
3	число #2				+							4	7	7	4	9	5	
4	результат					2	6	7	5	9	0	1	1	0	2	3	0	
5																		

Рис. 40. Поразрядное суммирование

Очевидно, при поразрядном суммированиичисел, которое начинается с конца строки, каждый разряд результата определяется по сумме трёх величин – переноса и соответствующих разрядов

операндов. В разряд записывается остаток от деления этой суммы на 10, а переносом в следующий разряд идёт частное от деления нацело этой суммы на 10.

Рассмотрим задачу, иллюстрирующую применение такого подхода

Задача №2. **Длинные A+B**

Найти сумму чисел A и B

Входные данные

В двух строках записаны числа Аи B, не превышающие 10^{1000}

Выходные данные

Одно число – сумма A и B.

Решение

```
#include <iostream>
using namespace std;
int main(){
	string a, b, c = "";
	cin >> a >> b;
	int lena = a.length()-1, lenb = b.length()-1,
		maxl = max(lena, lenb), carry = 0, op1, op2;
	for(int i = 0;i <= maxl; i++){
		op1 = i<=lena ? a[lena-i]-'0' : 0;
		op2 = i<=lenb ? b[lenb-i]-'0' : 0;
		c = char((op1 + op2 + carry) % 10 + '0') + c;
		carry = (op1 + op2 + carry) / 10;
	}
	if(carry) c = char(carry + '0') + c;
	cout << c;
}
```

Задача №3. **2^N (acmp.ru)**

Необходимо вычислить значение 2^n.

Входные данные

В единственной строке входного файла INPUT.TXT записано натуральное число n ($0 < n < 1000$).

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести значение 2^n.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3	8
2	10	1024

3	72	47223664828696452 13696

Решение

Воспользуемся моделью представления «число=строка». Данное решение можно использовать для любого K^N, при $K \leq 9$

```
#include <fstream>
#include <algorithm>                    //описание reverse
#define BASE 2
using namespace std;
int main(){
    ifstream cin("input.txt");
    ofstream cout("output.txt");
    string ans="1", tmp;
    int power,len, carry, expr,last;
    cin >> power;
    for(int i = 1; i <= power; i++){
        tmp = ans;
        len = tmp.length();
        ans.clear();
        carry = 0;
        for(int j = 0; j < len; j++){      //умножаем столбиком
            expr = (tmp[j]-'0') * BASE + carry;
            last = expr % 10;              //последняя цифра
            carry = expr / 10;             //перенос
            ans.push_back('0' + last);
        }
        if(carry)                          //последний перенос
            ans.push_back('0' + carry);
    }
    reverse(ans.begin(), ans.end());       //переворачиваем строку
    cout << ans;
}
```

В строке **ans** хранится в символьной форме результат последнего умножения на BASE. Строка **tmp** является промежуточной и используется для временного хранения результата перед возведением в следующую степень.

Задача №4. **Длинный факториал**

Требуется вычислить факториал целого числа N. Факториал обозначают как N! и вычисляют по формуле:

N! = 1 * 2 * 3 * … * (N-1) * N, причем 0! = 1.

Так же допустимо рекуррентное соотношение: N! = (N-1)! * N

Входные данные

В единственной строке записано одно целое неотрицательное число N (N < 1000).

Выходные данные

Вывести одно целое число — значение N!

Пример

№	Ввод	Вывод
1	6	720
2	17	355687428096000

Решение

```
#include <iostream>
using namespace std;
string mult(string x, int y){   //полудлинное умножение через строку
    int carry = 0, t;
    string result = "";
    for (int i = x.size() - 1; i >= 0; i--){
        t = int(x[i] - '0')*y + carry;
        carry = t/10;
        result = char(t%10 + '0')+result;
    }
    while(carry){               //в разряде переноса остались цифры
        result = char(carry%10 + '0') + result;
        carry /= 10;
    }
    return result;
}

int main(){
    int n;
    string str = "1";
    cin >> n;
    for(int i = 2; i <= n; i++) str = mult(str, i);
    cout << str << endl;
}
```

ВАЖНО: Одним из преимуществ данной модели представления длинных чисел является простота операций деления

Задача №5. **Очень длинное число**

Учитель математики Сергей Иванович хочет проверить, какой остаток дает число 19202122…979899 (подряд записаны двузначные числа от 19 до 99) при делении на 3^1, 3^2, 3^3 и 3^4. Сергей Иванович подозревает, что число делится без остатка на все указанные делители. Помогите ему проверить, прав ли он.

Подсказка: признак делимости по сумме цифр как для чисел 3 и 3^2=9 в случае 3^3 и 3^4 не применим.

Выходные данные

YES если Сергей Иванович прав, NO– если он ошибается.

Решение

Сохраним число Сергея Ивановича в строковый поток **stringstream st**, а затем будем его посимвольно считывать. Воспользуемся модификацией деления методом «уголка».

```
 1920212223...99| 81
-162| 23...
  300
 -243
   57 и т.д.
```

Проиллюстрируем это на примере табличного процессора

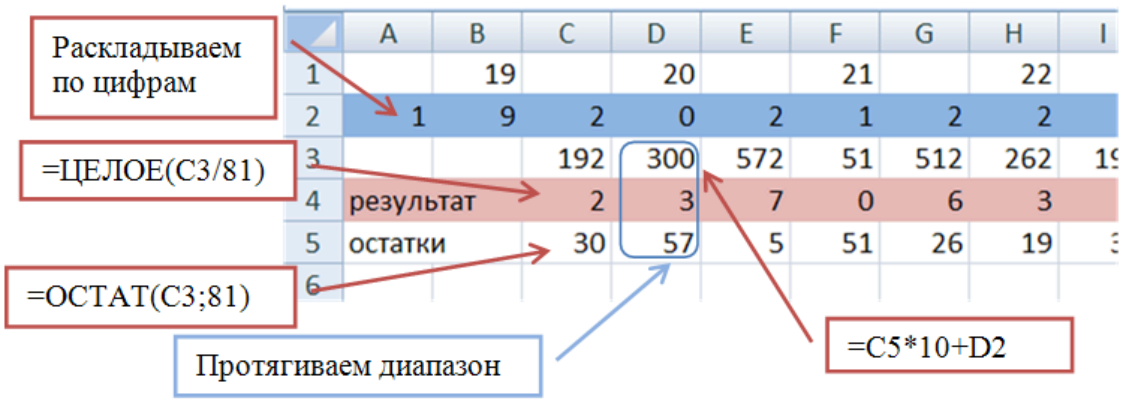

Рис. 41. Решение задачи "Очень длинное число" в табличном процессоре

Если последний остаток в итоге окажется нулевым, значит Сергей Иванович прав.

```
#include <iostream>
#include <sstream>
using namespace std;
int main(){
    string s;
    string streamst(s);          // связываем поток st со строкой s
    for(int i = 19; i <= 99; i++) st << i;//записываем число в поток
    char t;
    int temp = 0;
    for (int i = 1; i <= 3; i++){
           st >> t;
```

```
            (temp *= 10) += t - '0';
    }
    while (st >> t){
      ((temp %= 81) *= 10) += t - '0';
    cout << (temp % 3 ? "NO" : "YES");
}
```

В итоге Сергей Иванович окажется прав ☺

1.2 Модель 10^n

Представление числа напоминает представление многочлена, только вместо x в соответствующей степени имеем основание β в нужной степени. Как известно, многочлен $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ удобно представлять в виде массива, элементы которого представляют коэффициенты a_i, а индекс i — определяет соответствующую степень x. Длинное число хранится аналогично, осталось определиться с выбором основания β.

Например, то же самое число 123456789 можно представить в десятитысячной ($\beta = 10^4$) системе счисления следующим образом:

$123456789_{10} = 1*(10^4)^2 + 2345*(10^4)^1 + 6789*(10^4)^0$

Представляя число 123456789 в десятитысячной системе счисления, мы получаем сразу два преимущества:

1. сокращаем количество потребляемой памяти, так как вместо массива из 9 чисел нам достаточно хранить массив из 3 чисел (1, 2345 и 6789);
2. значительно уменьшаем время выполнения стандартных операций над длинными числами, поскольку за раз обрабатываем 4 разряда числа. В общем, компьютер одинаково быстро складывает одноразрядные и 32-разрядные числа, поэтому этим следует воспользоваться.

Задача №6. **Сумма произведений (acmp.ru)**

Требуется вычислить сумму произведений цифр каждого N-значного числа. При этом следует учесть, что если в числе встречается цифра 0, то произведение его цифр равно нулю. Для N=3 искомая сумма представлена следующим рядом:

S = 1*0*0 + 1*0*1 + 1*0*2 + … + 9*9*8 + 9*9*9 = 91125

Входные данные

В единственной строке записано натуральное число N (N < 1000).

Выходные данные

В единственную строку нужно вывести одно целое число — сумму произведений всех N-значных чисел.

Примеры

№	Ввод	Вывод
1	1	45

2	3	91125
3	5	184528125

Решение

Легко заметить, что требуемые числа S_n образуют геометрическую прогрессию с шагом 45. В самом деле, если S_1=45, то

$S_2 = 0\times0 + 0\times1 + 0 * 2 + .. + 0\times9 + 1\times0 + 1\times1 + 1\times2 + .. + 1\times9 + .. 9\times9 == 0\times(0 + 1 + 2 + .. 9) +$

аналогично для S_3 и т.д.

```
#include <cstdio>
#define SIZE 400                          //ячеек в массиве
#define BASE 1000000                    //основание системы счисления
intn, i, a[SIZE]={45,};                 //a[0]=45,остальные нули
int main(){
    scanf("%d",&n);
    for(;--n;){                         //вычисление 45^n
        for(i = 0; i < SIZE; i++) a[i] *= 45;
        for(i = 0; i < SIZE - 1; i++){
            a[i+1] += a[i] / BASE;
            a[i] %= BASE;
        }
    }
    for(i = SIZE; !a[--i];);            //вывод ячеек длинного числа
    printf("%d",a[i]);                  //первая ненулевая как есть
    for(;i--;)
        printf("%.6d",a[i]);            //остальные с нулями спереди
}
```

1.3 Улучшенная модель с массивом

Задача №7. **Опупенные числа (Симферополь, IIэтап, 2015)**

Вася Пупкин на уроке информатики недавно познакомился с числами Фибоначчи:

1, 1, 2, 3, 5, 8, 13, 21, 34, …

где каждое последующее число равно сумме двух предыдущих. С тех пор Васе не дают покоя лавры Леонардо Пизанского, известного под прозвищем «Фибоначчи». И Вася решил создать числовой ряд имени Пупкина «опупенные числа».

1,1,1,3,5,9,17,31,57,105,193,…

где каждое последующее число равно сумме ТРЁХ предыдущих.

Помогите Васе написать программу, определяющую такое число, по его номеру в последовательности.

Входные данные

Единственная строка содержит число N – номер члена числового ряда ($1 \le N \le 10000$).

Выходные данные

Одно число – значение P_N члена числового ряда с номером N.

Примеры

№	Ввод	Вывод
1	3	1
2	13	653
3	43	5680425 0945

Решение

Это типичная задача на длинную арифметику. Нам она интересна лишь в контексте применения данной модели работы с длинными числами

```
#include <cstdio>
#define SIZE 1000
#define BASE 1000000
int N, f[SIZE], tmp[SIZE];
int f1[SIZE]={1,1,}, f2[SIZE]={1,1,}, f3[SIZE]={1,1,};
//суммирование длинных c = a + b
void SumLong(int *a, int *b, int *c){
      for (int i = 0; i < SIZE; i++) c[i] = 0;
      int len = (a[0] > b[0] ? a[0] : b[0]);
      for (int i=1; i<=len; i++) {
            c[i+1] = (a[i] + b[i] + c[i]) / BASE;
            c[i]   = (a[i] + b[i] + c[i]) % BASE;
      }
      c[0] = len;
      if (c[len+1]) c[0]++;
}
//присваивание для длинных чисел to = from
void CopyLong(int * from, int *to){
      for (int i = 0; i < SIZE; i++) to[i] = from[i];
}
//вывод длинного числа
void WriteLong(int *a){
      printf("%d", a[a[0]]);
```

```
     for (int i = a[0]-1; i; i--)
          printf("%06d", a[i]);
}

int main() {
     scanf("%d", &N);
     if (N < 4) {
          printf("1");
          return 0;
     }
     for (int i = 4; i <= N; i++) {
          SumLong(f3, f2, tmp);
          SumLong(tmp, f1, f);
          CopyLong(f2, f3);
          CopyLong(f1, f2);
          CopyLong(f,  f1);
     }
     WriteLong(f);
}
```

1.4 Класс длинной арифметики через вектор

1.4.1 Базовое описание класса сверхдлинных знаковых целых

В этой части книги рассмотрим поэтапно создание класса **huge** для выполнения операций длинной арифметики. Наш класс будет выполнять стандартные арифметические операции над знаковыми целыми числами большой длины.

Реализуем его на базе STL-контейнера **vector**– массива переменной длины. В минимальном варианте нам потребуется описать величину **base** – максимальное значение ячейки массива, содержащего элементы **huge.** Само описание вектора **value** в **protected**– оно должно быть доступно только функциям-членам класса и дружественным функциям. Также в **protected** поместим величину **digits** – максимальное количество цифр «длинного» числа, содержащихся в одной ячейке массива. Чтобы уменьшить количество операций, а значит, ускорить вычисления, возьмем наибольшее возможное для int значение **digit = 9** и **base =1e9**.

С модификатором доступа **public** объявим конструкторы класса, операторы присваивания и потокового ввода-вывода.

```
class huge
{
protected:
    vector<int> value;
    static const int base = 1e9;
    static const int digits = 9;
public:
```

```
    huge();
    huge(const string& );
    huge(long long);

    huge& operator =(const huge&);
    huge& operator +=(const huge&);
    huge& operator *=(const huge&);

    friend huge operator +(const huge&, const huge&);
    friend huge operator *(const huge&, const huge&);

    friend istream& operator >>(istream&, huge&);
    friend ostream& operator <<(ostream&, const huge&);
};
```

Для реализации класса нам потребуются библиотеки ввода-вывода

```
#include <iostream>  //операции ввода-вывода + работа со строками
#include <iomanip>   //флаги для форматированного вывода
#include <vector>    //описание vector-ов
#include <sstream>   //строковые потоки для преобразования чисел
#include <cstdlib>   //функция atoi

using namespace std; //стандартное пространство имён
```

1.4.2 Конструкторы класса

Для создания объектов класса huge нам потребуется три конструктора: конструктор по умолчанию, конструктор по длинному целому и конструктор по строке.

```
//конструктор по умолчанию
huge::huge() {
    value.push_back(0);
}
//конструктор по строке
huge::huge(const string& str){
    for (int i = str.length(); i > 0; i -= digits)
        if (i < digits)
            value.push_back(atoi(str.substr(0, i).c_str()));
        else
            value.push_back(atoi(str.substr(i-digits,
                                             digits).c_str()));
}
//конструктор по длинному целому
```

```
huge::huge(long long x) {
    ostringstream ost;
    ost << x;
    string str = ost.str();
    for (int i = str.length() ; i > 0 ; i -= digits)
        if (i < digits)
            value.push_back(atoi(str.substr(0, i).c_str()));
        else
          value.push_back(atoi(str.substr(i-digits,
                                           digits).c_str()));
}
```

1.4.3 Сложение длинных чисел

```
//оператор добавления длинного целого
huge& huge::operator +=(const huge& b){
    for (int carry = 0, i = 0;
         i < max(this->value.size(), b.value.size()) || temp; i++){

        if (i ==  value.size())       //создаём новые разряды
        value.push_back(0);           // и заполняем их нулями

        value[i] += carry + (i < b.value.size() ? b.value[i] : 0 );
        carry = value[i] >= base;     // находим значение переноса
        if (carry) value[i] -= base;
    }
    return *this;
}
//универсальный оператор сложения
huge operator +(const huge& x, const huge& y){
    huge z = x;
    z += y;
    return z;
}
```

1.4.4 Длинное умножение

Поскольку элементы нашего вектора имеют тип **int**, произведение двух таких элементов может вызвать переполнение. Поэтому будем результат поразрядного умножения сохранять в **long long**. Для преобразования произведения двух целых к типу **long long** используем в качестве одного из множителей литерал **1LL** (единица в представлении **long long**).

```
// универсальный оператор умножения
huge operator * (consthuge&a, const huge& b){
```

```
    huge c;
    c.value.resize(a.value.size() + b.value.size());

    for (int i = 0; i<a.value.size(); i++)
      for (int j = 0, carry = 0; j < b.value.size() || carry; ++j){
            long long cur = c.value[i+j] + a.value[i] * 1LL *
                          (j<b.value.size()  ?  b.value[j]  :  0)  +
carry;
            c.value[i+j] = int(cur % huge::base);
            carry = int (cur / huge::base);
        }

    while (c.value.size() > 1 && !c.value.back())
        c.value.pop_back();

    return c;
}

huge& huge::operator *= (const huge& x){
    huge y = *this;
    *this = y + x;
    return *this;
}
```

1.4.5 Ввод, вывод и присваивание

```
//реализация потокового ввода
istream& operator >>(istream& is, huge& x){
    string str;
    is >> str;
    x = huge(str);
    return is;
}
//реализация потокового вывода
ostream& operator <<(ostream&os, const huge& x){
    os<< (x.value.empty() ? 0 : x.value.back());

    for (int i =  x.value.size() - 2; i >= 0; i--)
        os << setfill('0')
           << setw(huge::digits)
```

```
            << x.value[i];
    return os;
}
// оператор присваивания - реализация
huge& huge::operator =(const huge& x){
    this->value = x.value;
    return *this;
}
```

1.5 Длинная арифметика в факторизованном виде

1.5.1 Общее представление

Здесь идея заключается в том, чтобы хранить не само число, а его факторизацию, т.е. степени каждого входящего в него простого.

Этот метод также весьма прост для реализации, и в нём очень легко производить операции умножения и деления, однако невозможно произвести сложение или вычитание. С другой стороны, этот метод значительно экономит память в сравнении с "классическим" подходом, и позволяет производить умножение и деление значительно (асимптотически) быстрее.

Этот метод часто применяется, когда необходимо производить деление по непростому модулю: тогда достаточно хранить число в виде степеней по простым делителям этого модуля, и ещё одного числа — остатка по тому же модулю.

Рассмотрим задачу, в которой не используется длинная арифметика, но применяется «наивная» факторизация.

Задача №8. Преобразование моноклеточных (acmp.ru)

(Время: 1 сек. Память: 16 Мб)

Как великолепна страна Байтландия! В ней есть цветущие леса, прозрачные реки, кисельные берега… Но речь пойдет не о них. Уже много лет в Байтландии функционирует НИИ “Цитологии и генетики”. В нем выводятся новые формы жизни. Недавно ученым этого НИИ удалось разработать принципиально новый вид организмов. Особенностью этих организмов является то, что они состоят из большого количества однотипных клеток, то есть являются моноклеточными.

Правительство Байтландии заинтересовалось новой разработкой и сделало заказ на производство двух моноклеточных организмов, в каждом из которых должно быть по M клеток. За несколько дней до сдачи проекта было обнаружено, что в одном из организмов получается не M клеток, а N. На какой из стадий разработки была допущена ошибка неизвестно, но положение надо исправлять!

Сотрудниками НИИ было принято решение о преобразовании моноклеточного с N клетками в моноклеточное с M клетками. Для этого в экстренном режиме было разработано два типа вещества:

1. Вещество, которое делит клетки моноклеточного организма, т.е. каждая клетка делится на P частей. В результате количество клеток умножается на P, где P – простое число.
2. Вещество, объединяющее клетки. Клетки организма объединяются в группы по T штук, где T также простое число. Далее каждая группа клеток объединяется в одну клетку. В результате общее количество клеток делится на T. При этом T выбирается таким, чтобы деление происходило без остатка.

Отметим, что натуральное число называется простым, если оно имеет только два натуральных делителя – это единица и само число.

Серьезным недостатком этих веществ является их высокая стоимость. В соответствии с этим требуется преобразовать моноклеточное с N клетками в моноклеточное с M клетками за минимальное количество операций. За одну операцию к моноклеточному можно применить одно вещество из двух заданных типов. Помогите НИИ “Цитологии и генетики” разрешить эту непростую задачу!

Входные данные

В первой строке заданы два натуральных числа N и M ($1 \le N, M \le 10^9$) разделенные одиночным пробелом.

Выходные данные

Одно целое число – минимальное количество операций, необходимое для преобразования моноклеточного организма с N клетками в моноклеточный организм с M клетками.

Примеры

№	Ввод	Вывод
1	2 36	3
2	32768 3	16
3	1434 1434	0

Решение

Вначале найдём НОД(N, M) при помощи функции **__gcd** по алгоритму Евклида, определённой в библиотеке **algorithm**. Разделим каждое из чисел на эту величину. У нас останутся *взаимно простые числа* (не имеющие общих делителей, кроме 1). Для них останется произвести факторизацию – представление в виде произведения простых множителей, а затем найти суммы степеней простых множителей. Реализуем её в виде функции f(t).

```
#include <bits/stdc++.h>
using namespace std;

int n, m, i, p, g;

void f(int t){                    // делим каждое число на НОД
    t /= g;                       // и производим факторизацию
    for(i = 2; i * i <= t; i++)
        while(t % i == 0){
            t /= i;
            p++;
        }
    p += t > 1;
}
```

```
int main(){
    cin >> n >> m;
    g = __gcd(n, m);
    f(n);
    f(m);
    cout << p;
}
```

1.5.2 Реализация факторизованного представления структурой

Настало время обзавестись в нашем инструментарии реализацией факторизованного представления чисел. Воспользуемся представлением в виде структуры.

Сами числа будут описаны в форме

$$C = p_0^{q_0} \times p_1^{q_1} \times \ldots \times p_n^{q_n}$$

```
#include <iostream>
#include <vector>
using namespace std;

struct factorized{                      //структура факторизации
    vector<int>p, q;                    // множители и степени
    factorized(){};                     // конструктор по умолчанию
    factorized(long long);              //   конструктор   по   длинному
целому
    factorized& operator =(const factorized&);        //присваивание
    const factorized operator *(const factorized&);   //умножение
    friend ostream& operator <<(ostream&, const factorized&); //вывод
    long long value();                  // значение числа
};

// конструктор по длинному целому - реализация
factorized::factorized(longlongx){
    for(int d = 2, xx = x; x>1 && d*d <= xx; d++){ // xx - старое x
        if(x % d == 0) {                // поиск простых множителей
            p.push_back(d);             // производим от 2 до sqrt(x)
            q.push_back(0);
        }
        while(x % d == 0){              // считаем степени для
            x /= d;                     // множителей в разложении
            q[q.size()-1]++;
        }
```

```
    }
    if(x > 1) {                         // если остался ещё простой
        p.push_back(x);                 // множитель больше sqrt(xx)
        q.push_back(1);
    }
}

// потоковый вывод - реализация
ostream& operator <<(ostream& os, const factorized& x){
    if(x.p.empty()) os << 1;
    else
    for(int i = 0; i < x.p.size(); i++){   // выводим в виде
        os << x.p[i] << '^' << x.q[i];     // 2^3*3^1*5^2 (для600)
        if(i < x.p.size()-1) os << '*';
}
    return os;
}

//присваивание - реализация
factorized& factorized::operator =(const factorized& op){
    this->p = op.p;
    this->q = op.q;
    return *this;
}

// произведение факторизованных чисел - реализация
const factorized factorized::operator *(const factorized& op){
    factorized res;
    long long i = 0, j = 0;
    bool ii, jj;                                  //    ii==true    если
берем
    while(i<p.size() || j<op.p.size()){    // множитель от первого
        ii = jj = false;                   // числа, jj- от второго
        if(i == p.size()) {ii = false; jj = true;}
        else if(j==op.p.size()) {ii = true; jj = false;}
                else {ii = p[i] <= op.p[j]; jj = op.p[j] <= p[i];}
        res.p.push_back(ii ? p[i] : op.p[j]);
        res.q.push_back(ii*q[i] + jj*op.q[j]);
        i += ii; j += jj;
    }
```

```
    return res;
}

// определить значение факторизованного числа - реализация
long long factorized::value(){
    long long res = 1;
    for(int i = 0; i < p.size(); i++)
        for(int j = 0; j < q[i]; j++)
            res *= p[i];
    return res;
}
```

Следует отметить, что множители в такой модели выстраиваются в порядке возрастания и при произведении чисел в факторизованной модели, возрастающий порядок сохраняется.

Задача №9. **Степень (acmp.ru)**

(Время: 1 сек. Память: 16 Мб)

Источник задачи: Московская олимпиада по информатике 2003/2003 г.

Для того чтобы проверить, как её ученики умеют считать, Мария Ивановна каждый год задаёт им на дом одну и ту же задачу – для заданного натурального A найти минимальное натуральное N такое, что N в степени N (N, умноженное на себя N раз) делится на A. От года к году меняется только число A.

Вы решили помочь будущим поколениям. Для этого вам необходимо написать программу, решающую эту задачу.

Входные данные

Единственное число A ($1 \le A \le 10^9$ – на всякий случай, вдруг Мария Ивановна задаст большое число, чтобы «завалить» кого-нибудь…).

Выходные данные

Единственное число N.

Примеры

№	Ввод	Вывод
1	8	4
2	13	13

Решение

Воспользуемся факторизованной моделью представления чисел. Если A представить в виде:

$$A = p_0^{q_0} \times p_1^{q_1} \times \ldots \times p_n^{q_n}$$

Где все множители – простые числа, то, очевидно, минимальное возможное значение N равно:

$$N_{min} = p_0 \times p_1 \times \ldots \times p_n$$

Также, очевидно, N≤A, поскольку число A^A будет делиться на A без остатка. Соответственно, число N кратно N_{min} и не превышает A.

Возводить числа, представленные в факторизованной модели в любую степень, в том числе равную самому числу, очень легко. Например, если

$$C = p_0^{q_0} \times p_1^{q_1} \times \ldots \times p_n^{q_n}$$

то

$$C^C = p_0^{C \times q_0} \times p_1^{C \times q_1} \times \ldots \times p_n^{C \times q_n}$$

Далее можно воспользоваться описанной ранее структурой **factorized**, из которой, впрочем, нам потребуются лишь конструктор по длинному целому и оператор присваивания.

```
struct factorized{
     ...
};
int main(){
    int a, n = 1, i, j, m;
    cin >> a;
    factorized b(a), c;                 // b – это факторизованное a
    for(i = 0; i < b.p.size(); i++)
        n *= b.p[i];                    // находим n_min
    for(m = n; n <= a; n += m){         // перебираем n от n_min до a
        c = n;                          // c – это факторизованное n
        for(i = j = 0; i<b.p.size(); i++){
            while(b.p[i] > c.p[j]) j++;
            if(b.q[i] > c.q[j]*n) break;
        }
        if(i == b.p.size()) {cout << n; return 0;}
    }
}
```