11 Класс

Максимальная продолжительность – 240 мин.
Максимально возможное количество баллов – 500

Задача 1 (Время – 1 сек., память – 16 Мб, 100 баллов)

Требуется написать программу, определяющую наименьшее общее кратное (НОК) чисел a и b.

Входные данные: два натуральных числа A и B через пробел, не превышающих 46340.

Выходные данные: целое число — НОК чисел A и B.

Примеры работы программы:

№	*ввод*	*вывод*
1	36 27	108
2	39 65	195

Решение: Для нахождения НОК удобно использовать следующее свойство: для любых натуральных чисел a и b верно равенство НОД(a,b)*НОК(a,b)=a*b , откуда получаем, что НОК(a,b)=a*b/НОД(a,b).

В условиях данной задачи можно НОД найти перебором, но более универсально использовать алгоритм Евклида:

Примеры решения:

```
//Pascal
read(a,b);
a1:=a; b1:=b;
while a*b > 0 do
   if a >= b then a = a mod b else b = b mod a;
write(a1*b2/(a+b));
```

//C++:

```
#include <stdio.h>
#include <iostream.h>

int a,b;

int main(){
  cin >> a >> b;
  while(b) b^=a^=b^=a%=b;
  cout << a;
  return 0;
```

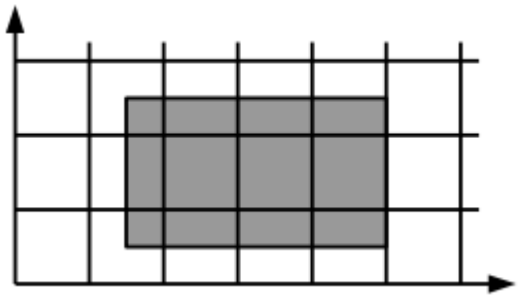
Критерии оценивания: эффективный алгоритм, программа работает без ошибок – 100 баллов, программа неэффективная, но выдает верный результат – 50 баллов.

Задача 2 (Время – 1 сек., память – 16 Мб, 100 баллов)

Стена покрыта квадратной плиткой со стороной *M* см. На стену повесили картину, известны координаты (*X* и *Y*) левого нижнего угла картины, её ширина и высота (*W* и *H)*. Определите количество плиток, которые оказались частично или полностью закрыты картиной.

Входные данные: *M* — сторону плитки,. *X* и *Y* — координаты левого нижнего угла картины, *W* и *H* — ширина и высота картины. Ось *OX* направлена вправо, ось *OY* направлена вверх. Все числа целые, не превосходящие 2×10^9, числа *M, W, H* — положительные, числа *X* и *Y* — положительные или равны 0.

Выходные данные: количество плиток, полностью или частично закрытых картиной. Плитка считается закрытой картиной, если пересечение картины и плитки имеет ненулевую площадь, то есть касание картины и плитки не считается перекрытием.



Решение, правильно работающее только для случаев, когда все входные числа не превосходят 100, будет оцениваться в 40 баллов. Решение, правильно работающее только для случаев, когда все входные числа не превосходят 10^5, будет оцениваться в 70 баллов.

Решение:

Разобьём плоскость на плитки (квадраты со стороной *M*) и пронумеруем столбцы и строки квадратов начиная с 0. Сначала определим, в какой столбец плиток попадёт левая сторона картины, в какой столбец попадёт правая сторона картины, в какой ряд плиток попадёт нижняя сторона картины, в какой ряд плиток попадёт верхняя сторона картины. Для этого нужно координаты сторон плиток поделить на *M*. Левая сторона картины имеет координату *X*, правая — *X* + *W*, нижняя — *Y*, верхняя — *Y* + *H*. При этом при определении номера столбца левой стороны и номера строки нижней стороны нужно делить на *M* с округлением вниз, а при определении номера столбца правой стороны и номера строки верхней стороны нужно делить на *M* с округлением вверх. Затем определяем, количество столбцов и строк, которое занимает картина и перемножаем два полученных числа. Отметим, что в языках Pascal, C++, Java для получения полного балла необходимо проводить вычисления с использованием 64-битных целых чисел, т. к. при использовании обычных целых чисел происходит переполнение целочисленной переменной при вычислении.

Пример решения:

```
#Python
M = int(input())
X = int(input())
Y = int(input())
W = int(input())
H = int(input())
left = X // M
right = (X + W - 1) // M
bottom = Y // M
```

```
top = (Y + H - 1) // M
print((right - left + 1) * (top - bottom + 1))
```

Критерии оценивания: решение, правильно работающее только для случаев, когда все входные числа не превосходят 100, будет оцениваться в 40 баллов. Решение, правильно работающее только для случаев, когда все входные числа не превосходят 10^5, будет оцениваться в 70 баллов. Правильное решение - 100 баллов.

Задача 3 (Время – 1 сек., память – 16 Мб, 100 баллов)

Напишите программу-архиватор, работающую по следующему принципу: пусть есть некоторая последовательность символов латиницы, например «bbaaaaaaaaaaccc». Если буква встречается более 2 раз подряд, то она нуждается в «архивации». Последовательность до 9 символов включительно заменяется цифрой (количество повторений символа) и самим символом. В ситуации, когда одинаковых символов следующих подряд более 9, первые 9 заменяются цифрой «9» и символом, оставшаяся часть такой последовательности далее рассматривается по вышеуказанным правилам.

Заархивированная строка будет выглядеть следующим образом «bb9aaa3c».

Входные данные: s - строка, которую требуется заархивировать заданному алгоритму.

Выходные данные: строка – результат архивации.

Пример работы программы:

ввод	**вывод**
bbaaaaaaaaaaccc	bb9aaa3c

Решение.

Предлагаемая задача на обработку строки текста, согласно описанному алгоритму.

Возможные решения:

//Pascal.

```
program solve03;
uses SysUtils;
//Рекурсивная функция кодирования последовательности одинаковых символов
function code(a: string): string;
  begin
    if length(a) < 3 then result := a else
    if length(a) < 10 then result := IntToStr(length(a)) + a[1]
      else begin
            a := copy(a,10,length(a)-9);
            result := '9' + a[1] + code(a);
       end;
  end;
//Основная программа
var i: integer;
    s,t,res: string;
begin
readln(s);
s := s + '!'; t := ''; res := '';
//Будем разбивать строку на подстроки - последовательности одинаковых символов
for i := 1 to length(s)-1 do begin
    t := t + s[i];
    if s[i] <> s[i+1] then begin
//Если в череде повторяющихся символов встречен отличный от
```

```
//предыдущих, то подстроку отправляем на кодирование
        res := res + code(t);
        t := '';
    end;
end;
//Учтём, что последний символ может быть пустым и тогда его кодировать не надо
if t <> '' then res := res + code(t);
//Вывод результата
writeln(res);
end.
```

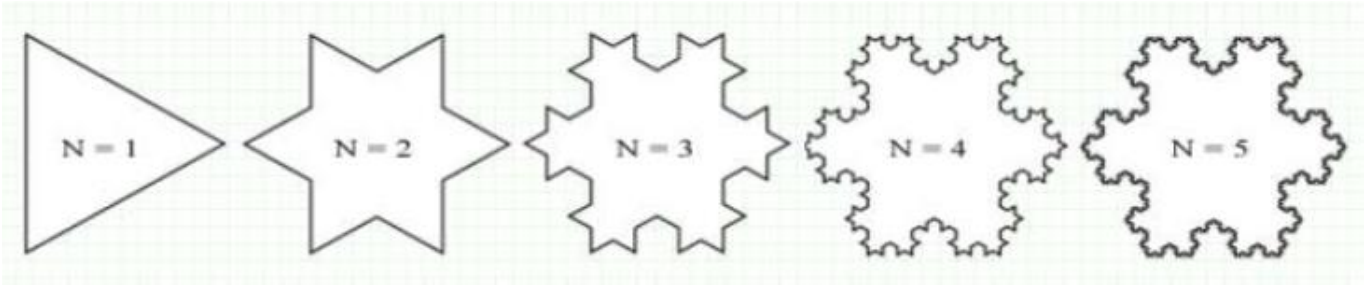
 # Python.

```
input(s);
s =s.strip()+'*'
buf = s[0]
s = s[1:]
count = 1
outs = ''
t = True
while t:
    #print(buf)
    if buf[-1] == s[0]:
        buf +=s[0]
        s = s[1:]
        if len(buf)==9:
            outs+='9'+buf[0]
            buf = s[0]
            s = s[1:]
        else:
            if len(buf)<3:
                outs+=buf
            else:
                outs += str(len(buf))+buf[0]
            buf = s[0]
            s = s[1:]
        if len (s) == 0: t = False
write(outs)
```

Критерии оценивания: эффективный алгоритм, программа работает без ошибок – 100 баллов; программа неэффективная, но выдает верный результат – 50 баллов, программа выдает неверный результат, но в целом решение верное – 20 баллов.

Задача 4 (Время – 1 сек., память – 16 Мб, 100 баллов)

Снежинка Коха – фрактальная кривая, которая строится на основе равностороннего треугольника, который представляет собой снежинку Коха порядка (N=1). Снежинка Коха К-го порядка строится из подобной кривой (К-1)-го порядка (К>1) путем замены каждой стороны данной фигуры четырьмя отрезками, каждый из которых представляет 1/3 от длины исходного отрезка (см. рисунок).



По заданному значению N требуется определить площадь фигуры, ограниченной снежинкой Коха N-го порядка, полагая, что при N=1 площадь равна единице.

Входные данные: натуральное число N ($N<=10^{18}$).

Выходные данные: площадь фигуры, ограниченной снежинкой Коха N-го порядка не менее, чем с шестью знаками после десятичной точки.

Пример работы программы:

№	Ввод	Вывод
1	1	1.000000
2	2	1.333333

Решения, работающие только для N<=5 будут оцениваться в 30 баллов.

Решения, работающие только для $N<=10^6$ будут оцениваться в 60 баллов.

Решение: Заметим, что изначально мы имеем равносторонний треугольник с площадью 1. При построении снежинки Коха 2-го порядка к нему добавляются еще 3 треугольника, площадь каждого из которых в 9 раз меньше площади исходного (т.к. линейные размеры уменьшены втрое). Так мы получили 12-сторонний многоугольник с равными длинами сторон.

Далее, при переходе от фигуры (i-1)-го порядка к i-му порядку очередное количество добавляемых треугольников увеличивается в 4 раза, а их площади уменьшаются в 9 раз.

Используем следующие обозначения:

P - площадь добавляемых треугольников,

K - количество добавляемых треугольников,

S - площадь всех треугольников, включая предыдущие шаги.

Таким образом, мы можем на каждом шаге вычислять количество добавляемых треугольников K, их площади P и добавлять к конечному ответу **S** их общую площадь **K·P**. В результате за N-1 шаг таких действий мы получим ответ. При этом следует учесть, что ответ нужно выводить с заданной точностью и что при N>50 в качестве ответа можно вывести площадь снежинки Коха 50-го порядка, т.к. последовательность площадей снежинки Коха сходится к числу 1.6 и при больших значениях N в качестве ответа можно выводить значение 1.6

Есть и другой способ решения этой задачи. Площадь снежинки Коха можно вычислить математически и представить решение одной формулой:

$$S_1 = 1$$

$$S_2 = S_1 + 3 \cdot \frac{1}{9} \cdot S_1 = 1 + \frac{1}{3}$$

$$S_3 = S_2 + 3 \cdot 4 \cdot \frac{1}{9^2} \cdot S_1 = 1 + \frac{1}{3} + \frac{1}{3} \cdot \frac{4}{9}$$

$$S_4 = S_3 + 3 \cdot 4^2 \cdot \frac{1}{9^3} \cdot S_1 = 1 + \frac{1}{3} + \frac{1}{3} \cdot \frac{4}{9} + \frac{1}{3} \cdot \frac{4^2}{9^2}$$

$$S_N = 1 + \frac{1}{3} + \frac{1}{3} \cdot \frac{4}{9} + \frac{1}{3} \cdot \frac{4^2}{9^2} + \ldots + \frac{1}{3} \cdot \frac{4^{N-2}}{9^{N-2}} = 1 + \frac{1}{3} \cdot \sum_{i=0}^{N-2} \left(\frac{4}{9}\right)^i = 1 + \frac{1}{3} \cdot \left(\frac{1 - \left(\frac{4}{9}\right)^{N-1}}{1 - \left(\frac{4}{9}\right)} \right) = 1 + \frac{1}{3} \cdot \left(1 - \left(\frac{4}{9}\right)^{N-1}\right)$$

Откуда легко видеть, что площадь бесконечной снежинки Коха составляет ровно 1.6, так как верно, что

$$\lim_{n\to\infty}\left(1+\frac{3}{5}\cdot\left(1-\left(\frac{4}{9}\right)^{N-1}\right)\right)=1\frac{3}{5}=1.6$$

Примеры программ:

```pascal
//Pascal
var n : int64;
s,p,k : real;
begin
read(n);
if n>50 then n := 50;
s := 1;
p := 1;
k := 3;
while n>1 do begin
    p := p/9;
    s := s + k*p;
    k := k*4;
    dec(n)
end;
write(s)
end.
```

```pascal
begin
write(1+3/5*(1-power(4/9, ReadReal-1)))
end
```

```cpp
// C++
#include <bits/stdc++.h>
using namespace std;
int main(){
long long n;
double s=1,p=1,k=3;
cin >> n;
n = min(50LL, n);
while(--n)
   p /= 9,
   s += k*p,
   k *= 4;
cout << fixed << s;
return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
double n;
cin >> n;
cout << fixed << 1+0.6*(1-pow(4./9, n-1));
return 0;
}
```

```python
#Python
print(1+0.6*(1-(4/9)**(int(input())-1)))
```

Решения, работающие только для N<=5, оцениваются в 30 баллов.
Решения, работающие только для $N<=10^6$, оцениваются в 60 баллов.
Решения, работающие при $N<=10^{18}$, оцениваются в 100 баллов.

Задача 5 (Время – 1 сек., память – 16 Мб, 100 баллов)

В выборах участвовало несколько кандидатов. В результате проведения тайного голосования был получен список из N фамилий. Каждая фамилия - это голос, отданный участником голосования в пользу это кандидата.

На основании этих данных требуется построить гистограмму результатов проведенного голосования.

Входные данные: N – натуральное число, количество отданных голосов (N<=1000). В последующих N строках идут фамилии кандидатов по одному в каждой строке. Каждая фамилия содержит от1 до 10 букв английского алфавита, при этом первая буква прописная, а остальные – строчные.

Выходные данные: гистограмма голосования в форме прямоугольника, состоящего из символов «.» (ASCII 46) и «X» (ASCII 88). Число строк должно соответствовать максимальному количеству голосов, отданных за какого-либо кандидата. Число столбцов должно соответствовать количеству кандидатов, участвовавших в выборах. Гарантируется, что за каждого кандидата был отдан как минимум один голос. Высота i-го столбца (число символов «X») таблицы должна соответствовать числу отданных голосов за i-го кандидата. При этом кандидаты перечисляются в алфавитном порядке слева направо. Столбцы обозначаются символами «X» снизу вверх, пустые места – символами «.».

Пример работы программы:

№	ввод	вывод
1	7 Ivanov Petrov Sidorov Petrov Zubov Petrov Zubov	. X . . . X . X XXXX

Решение:

Сначала посчитаем для каждого из кандидатов количество отданных за него голосов. Для этого удобно использовать структуру данных, именуемую ассоциативным массивом или словарем, желательно упорядоченным по ключу. Многие языки поддерживают такую структуру. Например, в C++ это map, в PascalABC - SortedDictionary, а в Python - dict:

```
// пример описания ассоциативного массива на C++
map <string, int> m;
```

Поскольку ограничения в задаче невелики, то можно также обойтись обычным массивом структур из двух полей:

```
//структура "Кандидат", хранящая информацию о голосах кандидата
struct candidate{
string surname;                 // фамилия кандидата
int votes;                      // количество голосов
}
candidate m[1..N]; //массив кандидатов с информацией о голосах
```

Последовательно читая фамилии кандидатов surname, будем увеличивать значение m[surname] на единицу в случае наличия такого элемента, либо создавать элемент m[surname]=1 в противном случае. При этом несложно вычислить максимальное значение mx голосов у победившего кандидата. Зная ширину (m.size - количество уникальных фамилий кандидатов) и высоту (mx) таблицы, несложно реализовать вывод решения.

Примеры программ:

```
//PascalABC.NET 3.2
```

```
var
    i, mx, n: integer;
    s : string;
    a := new SortedDictionary <string, integer>;
begin
readln(n);
for i:=1 to n do begin
     readln(s);
     if s in a then a[s] += 1
               else a[s] := 1;
    mx := max(mx, a[s])
end;
for i:=mx downto 1 do begin
   foreach var p in a do
      if p.value<i then write('.')
                   else write('X');
writeln
end
end.
```

```
//GNU C++ 5.1
#include <bits/stdc++.h>
using namespace std;
int n,i,mx;
string s;
map<string, int> m;
int main(){
cin >> n;
while(cin >> s)
    mx = max(mx, ++m[s]);
for(i=0; i<mx; i++){
    for(auto x : m)
        cout << (mx-i > x.second?'.':'X');
     cout << endl;
}
return 0;
}
```

Критерии оценивания: эффективный алгоритм, программа работает без ошибок – 100 баллов; программа неэффективная, но выдает верный результат – 50 баллов, программа выдает неверный результат, но в целом решение верное – 20 баллов.