



**Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

**ПРЕДПРОФЕССИОНАЛЬНЫЙ ЭКЗАМЕН  
для учащихся инженерных классов (11 класс) города Москвы**

***Семинар «Решение ситуационных задач практической части  
предпрофессионального экзамена»  
(Направление программирование)***

**Авторы:** *Калмыков Ю.В., старший преподаватель кафедры «Основы математики и информатики» СУНЦ МГТУ им. Н.Э. Баумана*

**Москва – 2019**

# Об экзамене

«Предпрофессиональный экзамен – это форма независимой итоговой оценки с участием представителей вузов, которая проводится по результатам освоения учащимися предпрофессиональных профильных программ в инженерных и медицинских классах образовательных организаций города. Экзамен состоит из двух частей – теоретической и практической. Первую, теоретическую, часть экзамена участники сдают в Московском центре качества образования, где для участников созданы все условия, аналогичные условиям при сдаче ЕГЭ. Вторая часть экзамена пройдет на базе вузов. Участников обеспечат всем необходимым оборудованием и материалами для выполнения практических заданий»

# Основные направления задач практической части

- **Программирование + физика.**

Сложное программирование, элементарная физика.

- **Физика + программирование.**

Задача по физике, для решения которой надо программировать.

# Направления подготовки по программированию

- Алгоритмы на графах. Как хранить граф в памяти, поиск пути.
- Численное интегрирование.
- Метод половинного деления.
- Рекурсивный перебор для массива.
- Динамическое программирование.
- Задача коммивояжёра  
([https://ru.wikipedia.org/wiki/Задача\\_коммивояжёра](https://ru.wikipedia.org/wiki/Задача_коммивояжёра)).
- Задача о рюкзаке ([https://ru.wikipedia.org/wiki/Задача\\_о\\_рюкзаке](https://ru.wikipedia.org/wiki/Задача_о_рюкзаке)).
- Задача о трубе (Проверить возможность составить из кусков труб трубу нужной длины с нужной пропускной способностью).

# Направления подготовки по физике

- Конденсаторы
- Колебания
- Резисторы
- Кинематика
- Задачи, в которых меняется ускорение или есть сопротивление среды.

# Перестановки в строке

Вывести все возможные перестановки символов в строке.

```
procedure perest(head, tail: string);
```

```
...
```

```
var
```

```
    tail: string;
```

```
begin
```

```
    readln(tail);
```

```
    perest('', tail);
```

```
end.
```

# Перестановки в строке

```
procedure perest(head, tail: string);
var newhead, newtail: string; i: integer;
begin
  if length(tail) = 0 then writeln(head)
  else begin
    for i := 1 to length(tail) do
      begin
        newhead := head + copy(tail, i, 1);
        newtail := tail;
        delete(newtail, i, 1);
        perest(newhead, newtail);
      end;
    end;
  end;
end;
```

# Перестановки в массиве

Вывести все возможные перестановки элементов в массиве.

```
const N = 5;
type tmas = array [1 .. N] of integer;
procedure init(var mas: tmas);
...
procedure perest(mas: tmas; cur: integer);
...
var mas: tmas;
begin
  init(mas);
  perest(mas, 1);
end.
```



# Перестановки в массиве

```
procedure init(var mas: tmas);  
var i: integer;  
begin  
    for i := 1 to N do  
        mas[i] := i;  
end;  
procedure vivod(mas: tmas);  
var i: integer;  
begin  
    for i := 1 to N do  
        write (mas[i], ' ');  
    writeln;  
end;
```

# Перестановки в массиве

```
procedure perest(mas: tmas; cur: integer);
var i: integer;
begin
  if cur > N then
    vivod(mas)
  else
    for i := cur to N do begin
      swap(mas[i], mas[cur]);
      perest(mas, cur + 1);
      swap(mas[i], mas[cur]);
    end;
end;
```

# Перестановки в массиве

```
procedure swap(var x, y: integer);  
var z: integer;  
begin  
    z:=x;  
    x:=y;  
    y:=z;  
end;
```

# Задача: Работа силы

Тело движется вдоль оси  $Ox$  под воздействием силы  $F(x)$ , сонаправленной с осью  $Ox$ . В общем виде  $F(x)$  определена как  $F=a(x+1)^{1/2}+b(x+2)+cx^2+d*\ln(x+1)$ , где  $a, b, c, d$  – коэффициенты, которые могут меняться на разных отрезках. Гарантируется, что каждое уравнение имеет математический смысл.

Зная количество отрезков, коэффициенты уравнения кривых и границы отрезков, на которых эти кривые применяются, найдите совершенную над телом работу силы  $f(x)$  с точностью  $\varepsilon = 10^{-5}$ .

Точность  $\varepsilon$  считать достигнутой, когда при вычислении интегральной суммы уменьшение отрезка  $x$  вдвое приводит к изменению суммы меньше, чем на  $\varepsilon$ .

## Задача: Работа силы (ввод/вывод)

- В первой строке вводится натуральное число  $N$  – число отрезков.  $N$  не превышает 5.
- Далее следует  $N$  строк, в каждой из которых через пробел записаны шесть вещественных положительных чисел  $x_i$ ,  $x_{i+1}$ ,  $a$ ,  $b$ ,  $c$ ,  $d$  – соответственно, границы отрезка, и коэффициенты при кривой на этом отрезке.
- Гарантируется, что разрывов нет и каждый следующий отрезок начинается там, где закончился предыдущий.
- Никакие числа не превышают 1000.
- На выходе программа должна выдать вещественное число – совершенную над телом работу силы  $f(x)$ . Все величины задаются в системе СИ, ответ привести в джоулях.

## Задача: Работа силы (решение)

```
function work(x0, x1, a, b, c, d: real): real;  
...  
var w, x0, x1, a, b, c, d: real; i, n: integer;  
begin  
    w := 0;  
    readln(n);  
    for i:=1 to n do begin  
        readln(x0, x1, a, b, c, d);  
        w := w + work(x0, x1, a, b, c, d);  
    end;  
    writeln(w);  
end.
```

# Задача: Работа силы (решение)

```
const
  eps = 0.00001;

function f(x, a, b, c, d: real): real;
begin
  f := a * sqrt(x + 1) + b * (x + 2) +
       c * sqr(x) + d * ln(x + 1);
end;
```

## Задача: Работа силы (решение)

```
function work(x0, x1, a, b, c, d: real): real;
var vt, pv, x, h: real;
begin
  vt := sqr(f((x1 + x0) / 2, a, b, c, d)) * (x1 - x0);
  h := (x1 - x0) / 2;
  repeat
    pv := vt; x := x0; vt := 0;
    while x < x1 do begin
      vt := vt + f(x, a, b, c, d) * h; x := x + h;
    end;
    h := h / 2;
  until abs(vt - pv) < eps;
  work := vt;
end;
```



# Задача: Уравновешивание груза

Чтобы уравновесить тяжелый груз, висящий на одном конце легкого стержня, требуется уравновесить его с помощью гирь. Зная массу груза, массу одинаковых гирь из набора и расстояния от точек приложения сил до точки подвеса, подобрать сочетание точек подвеса гирь, которое позволит уравновесить груз.

Груз считается уравновешенным, если моменты сил, действующих на него и на набор гирь, различаются не более чем на 5%.

## Задача: Уравновешивание груза (ввод/вывод)

- В первой строке вводится натуральное число  $N$  – число гирь и точек подвеса.  $N$  не превышает 10.
- Далее через пробел в одной строке вводятся вещественные положительные числа  $M$ ,  $L$ ,  $m_w$  – соответственно, масса груза, длина стержня от точки подвеса до груза, масса гирь.
- Далее следует  $N$  строк, в каждой из которых содержится одно вещественное положительное число  $l_i$  – расстояние от точки, на которой можно закрепить гирю до точки подвеса.
- Никакие числа не превышают 1000.
- На выходе программа должна выдать через запятую номера точек подвеса, или вывести 0, если уравновесить грузы невозможно.

# Задача: Уравновешивание груза (решение)

```
var mas: tmas1; nums: tmasn; j, n, kolvo: integer;
    flag: boolean; m, l, m_w: real;
begin
    readln(n); readln(m, l, m_w);
    for j := 1 to n do begin
        read(mas[j]); nums[j] := j;
    end;
    kolvo := 1; flag := false;
    while (kolvo <= n) and not flag do begin
        perest(nums, mas, l, kolvo, n, m, l, m_w, flag);
        kolvo := kolvo+1;
    end;
    if not flag then writeln(0);
end.
```

# Задача: Уравновешивание груза (решение)

```
const
  MaxN = 10;
  eps = 0.05;
type
  tmasl = array[1..MaxN] of real;
  tmasn = array[1..MaxN] of integer;
procedure output(num: tmasn; kolvo: integer);
var i: integer;
begin
  for i := 1 to kolvo - 1 do
    write(num[i], ', ');
  writeln(num[kolvo]);
end;
```

## Задача: Уравновешивание груза (решение)

```
function check(nums: tmasn; mas: tmasl; kolvo: integer;
               m, l, m_w: real): boolean;
var
  w2: real; i: integer;
begin
  w2:=0;
  for i := 1 to kolvo do
    w2 := w2 + mas[nums[i]];
  check := abs(m * l - m_w * w2) / (m * l) <= eps;
end;
procedure swap(var x, y: integer);
...
```

```

procedure perest(nums: tmasn; mas: tmasl;
  cur, kolvo, n: integer; m, l, m_w: real; var flag: boolean);
var i: integer;
begin
  if cur > kolvo then begin
    flag := check(nums, mas, kolvo, m, l, m_w);
    if flag then output(nums, kolvo)
  end
else begin
  i := cur;
  while (i <= n) and not flag do begin
    swap(nums[i], nums[cur]);
    perest(nums, mas, cur+1, kolvo, n, m, l, m_w, flag);
    swap(nums[i], nums[cur]);
    i := i + 1;
  end;
end;
end;
end;

```

# Задача: Пушка Гаусса

Для работы пушки Гаусса требуется подключить конденсаторы. Из-за ограничений, заданных при разработке оружия, объем набора конденсаторов не должен превышать заданное значение. Зная параметры типовых конденсаторов, которые подключаются параллельно в одну схему, требуется определить несколько вариантов подключения конденсаторов к пушке Гаусса.

# Задача: Пушка Гаусса

Заранее известно следующее.

1. Ограничение по объему  $V$  составляет  $1000 \text{ см}^3$ .
2. Всего существует не более пяти видов типовых конденсаторов. Объем каждого из них кратен  $0.5 \text{ см}^3$ , а емкость равна целому числу условных единиц (примечание автора: поскольку мы не знаем, какими будут конденсаторы во времена, когда будут создавать пушку Гаусса, мы остереглись давать конкретные единицы измерения). Запас конденсаторов считать бесконечным.



# Задача: Пушка Гаусса (ввод/вывод)

В первой строке на ввод подается целое число  $N$  – количество видов конденсаторов.

Далее в  $N$  строках задаются одно вещественное и два целых числа:  $vol$ ,  $cap$  и  $cost$  – объем, емкость и цена конденсатора. Объем вводится с точностью до одного знака после запятой. (примечание автора: даже во времена, когда будут создавать пушку Гаусса, объем, цена и емкость конденсатора положительны.).

## Вопросы:

Задание 1. Найти цену набора конденсаторов, который, помещаясь в заданном объёме, обеспечит наибольшую ёмкость.

Задание 2. Найти емкость наиболее дешевого набора конденсаторов, который целиком займёт заданный объем.

## Задача: Пушка Гаусса (решение)

```
const V = 2000; M = 5;
type  tmatr = array [0 .. V, 1 .. 2] of integer;
      tcapa = array [1 .. M, 1 .. 3] of integer;
var  capa: tcapa; n, i: integer; vol: real;
begin
  readln(n);
  for i := 1 to n do begin
    readln(vol, capa[i, 2], capa[i, 3]);
    capa[i, 1] := round(2 * vol);
  end;
  writeln(task_cost(capa, n));
  writeln(task_capacity(capa, n));
end.
```

# Задача: Пушка Гаусса (решение)

```
procedure init(var matr: tmatr);  
var i: integer;  
begin  
    matr[0, 1] := 0;  
    matr[0, 2] := 0;  
    for i := 1 to V do begin  
        matr[i, 1] := -1;  
        matr[i, 2] := 0;  
    end;  
end;
```

## Задача: Пушка Гаусса (решение)

```
function task_cost(capac: tcapac; n: integer): integer;
var i, j, max: integer; matr: tmatr;
begin
  init(matr);
  for i:=1 to V do
    for j:=1 to n do
      if (i - capac[j, 1] >= 0) and
          (matr[i - capac[j, 1], 1] <> -1) then
        if (matr[i, 1] = -1) or
            (matr[i, 1] < matr[i - capac[j, 1], 1] + capac[j, 2]) then
          begin
            matr[i, 1] := matr[i - capac[j, 1], 1] + capac[j, 2];
            matr[i, 2] := matr[i - capac[j, 1], 2] + capac[j, 3];
          end;
        ...
      end;
    end;
  end;
```

# Задача: Пушка Гаусса (решение)

```
function task_cost(сара: tсара; n: integer): integer;  
var i, j, max: integer; matr: tmatr;  
begin  
    ...  
    max := 0;  
    for i := 1 to V do  
        if matr[i, 1] > matr[max, 1] then  
            max := i;  
    task_cost := matr[max, 2];  
end;
```

## Задача: Пушка Гаусса (решение)

```
function task_capacity(cap: tcap; n: integer): integer;
var i, j: integer; matr: tmatr;
begin init(matr);
  for i := 1 to V do
    for j := 1 to n do
      if (i - cap[j, 1] >= 0) and
          (matr[i - cap[j, 1], 1] <> -1) then
        if (matr[i, 1] = -1) or
            (matr[i, 2] > matr[i - cap[j, 1], 2] + cap[j, 3]) then
          begin
            matr[i, 1] := matr[i - cap[j, 1], 1] + cap[j, 2];
            matr[i, 2] := matr[i - cap[j, 1], 2] + cap[j, 3];
          end;
        task_capacity := matr[V, 1];
      end;
    end;
  end;
```