Материалы заданий Олимпиады школьников «Надежда энергетики» по предмету «информатика» в 2014/2015 учебном году

Характер и уровень сложности олимпиадных задач направлены на достижение целей проведения олимпиады: выявить способных участников, твердо владеющих школьной программой и наиболее подготовленных к освоению образовательных программ технических ВУЗов, обладающих логикой и творческим характером мышления, умеющих алгоритмически описать реальные ситуации из различных предметных областей и применить к ним наиболее подходящие методы информатики. Необходимы знания способов описания алгоритмов (язык блок-схем, псевдокод) и умение работать с базовыми конструкциями.

Задания Олимпиады дифференцированы по сложности и требуют различных временных затрат на полное и безупречное решение. Они охватывают все разделы школьной программы, но носят, в большинстве, комплексный характер, позволяющий варьировать оценки в зависимости от проявленных в решении творческих подходов и продемонстрированных технических навыков. Участники должны самостоятельно определить разделы и теоретические факты программы, применимые в каждой задаче, разбить задачу на подзадачи, грамотно выполнить решение каждой подзадачи, синтезировать решение всей задачи из решений отдельных подзадач.

Успешное выполнение олимпиадной работы не требует знаний, выходящих за пределы школьной программы, но, как видно из результатов Олимпиады, доступно не каждому школьнику, поскольку требует творческого подхода, логического мышления, умения увидеть и составить правильный и оптимальный план решения, четкого и технически грамотного выполнения каждой части решения.

Умение справляться с заданиями Олимпиады по информатике приходит к участникам с опытом, который вырабатывается на тренировочном и отборочном этапах Олимпиады.

Решения вариантов заключительного этапа

Решение вариантов 7111 и 7112

Задание 1.

Разработайте алгоритм для перевода целых чисел, представленных в десятичной/восьмеричной системе счисления, в римскую систему счисления. В качестве исходных данных выступает число в десятичной/восьмеричной системе счисления. В качестве результата необходимо вывести представление в виде римских цифр.

Римские цифры появились около 500 лет до нашей эры у этрусков и представляют собой комбинацию буквенных обозначений, каждая буква обозначает число (цифру) из списка: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000. Натуральные числа записываются при помощи повторения этих цифр. При этом некоторые из цифр (I, X, C, M) могут повторяться, но не более трёх раз, таким образом, с их помощью можно записать любое целое число не более 3999 (MMMCMXCIX). Числа составляются путём составления «вычитающих» или «добавляющих» комбинаций.

- Если большая цифра стоит перед меньшей, то они складываются (принцип сложения). Правило применяется до трёх подряд, потом – смотри следующее правило.
- Если же меньшая цифра стоит перед большей, то меньшая вычитается из большей (принцип вычитания). Правило применяется только во избежание четырёхкратного повторения одной и той же цифры. Существует шесть вариантов использования правила вычитания: IV = 4, IX = 9, XL = 40, XC = 90, CD = 400, CM = 900.

Число / цифра 0 (ноль) не входит в ряд римских цифр.

```
алг РимскоеЧисло()
нач
  строка arab_num, roman_num
  цел num, i
  цел    values[13] = {1000, 900,  500, 400,  100, 90,   50,  40,   10,  9,    5,   4,    1}
  строка digits[13] = {"M",  "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"}

  ввод arab_num
  num = atoi(arab_num, 10)         // 10 – основание используемой системы счисления. В другом варианте будет 8
  если num <= 0 то
    вывод "Число введено некорректно"
  иначе
    если num > 3999 то
      вывод "Число слишком большое"
    иначе
      ronam_num = ""
      для i от 1 до 13
      нц
        пока num – values[i] >= 0
        нц
          num = num – values[i]
          roman_num = roman_num + digits[i]     // «+» – конкатенация строк
        кц
      кц
      вывод roman_num
    всё
  всё
кон

алг atoi(арг строка str, арг цел radix)
нач
  цел i, d, num

  num = 0
  для i от 1 до Length(str)
  нц
    d = ord(str[i]) – ord('0')           // ord – функция, определяющая код символа
    если d < 0 или d >= radix то
      вернуть -1                         // Поскольку в римской системе нет 0 и отрицательных чисел,
    иначе                                // значение -1 будет признаком ошибки при вводе
      num = num * radix + d
    всё
```

```
  кц
  вернуть num
кон
```

Задание 2. (схема решения)

Двоично-десятичная система счисления - это форма записи рациональных чисел, при которой каждый десятичный разряд числа записывается в виде его четырёхбитного двоичного кода. Например, десятичное число 21_{10} будет записано в двоичной системе счисления как 10101_2, а в двоично-десятичном коде как $0010\ 0001_{2\text{-}10}$.В каких случаях её применение целесообразно?

Преимущества 2-10 с.с.

- упрощён вывод чисел на индикацию — вместо последовательного деления на 10 требуется просто вывести на индикацию каждый полубайт.
- Для дробных чисел (как с фиксированной, так и с плавающей запятой) при переводе в «обычную» десятичную с.с. и наоборот не теряется точность.
- Упрощены умножение и деление на 10, а также округление.

Недостатки:

- Требует больше памяти.
- Усложнены арифметические операции. Так как в 8421-BCD используются только 10 возможных комбинаций 4-х битового поля вместо 16, существуют запрещённые комбинации битов: 1010(1010), 1011(1110), 1100(1210), 1101(1310), 1110(1410) и 1111(1510).

Таким образом, применение 2-10 с.с. целесообразно для операций умножения/деления на 10 и перевода дробных чисел из/в десятичную с.с.

Задание 3.

Мультипликативно обратным к целому числу a по модулю n является целое число b ($0 \le a < n$ и $0 \le b < n$), для которого справедливо сравнение $a \cdot b \equiv 1 \{\mathrm{mod}\ n\}$ (два целых числа сравнимы по модулю n, если они дают одинаковые остатки при делении на n). Мультипликативно обратное число существует тогда и только тогда, когда a и n взаимно простые числа (их наибольший общий делитель равен 1).

Разработайте алгоритм для нахождения мультипликативно обратного числа по задаваемому значению модуля для задаваемого целого числа.

Примеры

Входные данные		**Результат работы** (b)
Модуль (n)	**Целое число** (a)	
5	3	2
10	3	7
10	5	Не существует

Заметим, что если $a \ge 2$, то b не может быть равно 1, т.к. $a \cdot b = a$, и поскольку $a < n$, $a \cdot b \bmod n = a$, и будет больше и равно 2, что нас не устраивает. Соответственно, если $a = 1$, то b не может быть больше или равно 2 по тем же причинам, и может быть равно только 1. Получается, что число 1 мультипликативно обратно самому себе по любому модулю.

Далее, $a \cdot b \equiv 1 \{\mathrm{mod}\ n\}$ можно записать как $a \cdot b = m \cdot n + 1$, где m – некий коэффициент (в приведённых примерах $m = 1$ ($3 \cdot 2 = 1 \cdot 5 + 1$) и $m = 2$ ($3 \cdot 7 = 2 \cdot 10 + 1$)). В этом уравнении два неизвестных – b и m, но путём рассуждений можно прийти к выводу, что $2 \le b < n$ и $0 \le m \le a$.

Поэтому самым примитивным алгоритмом решения этой задачи будет подбор значения *b* или *m* с помощью проверки всех возможных значений.

```
алг МультипликативноОбратноеЧисло()
нач
  цел n, m, a, b
  лог found

  ввод n, a

  если n <= 0 или a >= n то
    вывод "Некорректные исходные данные"
  иначе
    если a = 0 то
      вывод "Решения нет, т.к. число 0 не имеет мультипликативно обратного числа"
    иначе
      если НОД(n, a) <> 1 то
        вывод "Решения нет, т.к. НОД <> 1"
      иначе
        если a = 1 то
          b = 1
          m = 0
        иначе

          b = 2                                  // Подбор значения b
          found = false
          пока не found
          нц
            если (a * b) mod n = 1 то
              m = (a * b - 1) div n
              found = true
            всё
          кц

          m = 0                                  // Подбор значения m
          found = false
          пока не found
          нц
            если (m * n + 1) mod a = 0 то
              b = (m * n + 1) div a
              found = true
            всё
          кц

        всё
        вывод b, m
      всё
    всё
  всё
кон

алг НОД(арг цел x, y)
нач
  пока x <> y
  нц
    если x > y то
      x = x - y
    иначе
      y = y - x
    всё
  кц
  вернуть x
кон
```

Приведённые способы решения сопоставимы по времени работы, но оптимальными не являются. Попробуем найти другой вариант.

Всем известен алгоритм Евклида, который находит наибольший общий делитель двух чисел (только что он использовался при проверке существования решения данной задачи). Но существует также и расширенный алгоритм Евклида, который для двух целых чисел x и y находит коэффициенты u и v, такие, что $x \cdot u + y \cdot v = d$, где d – наибольший общий делитель чисел x и y ($x \geq y$).

Алгоритм заключается в следующем. Сначала надо составить два исходных уравнения.

$x \cdot u_1 + y \cdot v_1 = x$

$x \cdot u_2 + y \cdot v_2 = y$

Для того чтобы уравнения выполнялись, коэффициенты должны иметь следующие значения: $u_1 = 1$, $v_1 = 0$, $u_2 = 0$, $v_2 = 1$. После этого из первого уравнения вычитается второе, умноженное на результат целочисленного деления x на y (обозначим как q).

$x \cdot (u_1 - q \cdot u_2) + y \cdot (v_1 - q \cdot v_2) = x - q \cdot y$

В правой части уравнения получается остаток от деления x на y. На следующем шаге те же действия выполняются над вторым и третьим уравнениями. Алгоритм прекращает работу, когда правая часть уравнения становится равной 0. При этом в качестве результата берутся значения коэффициентов, полученные на предпоследнем шаге. Значение в правой части уравнения на этом же предпоследнем шаге даёт наибольший общий делитель чисел x и y.

Рассмотрим примеры.

u	***v***	***r***	***q***	***u***	***V***	***r***	***q***	***u***	***v***	***r***	***q***
1	0	5		1	0	10		1	0	55	
0	1	3		0	1	3		0	1	15	
1	–1	2	1	**1**	**–3**	**1**	3	1	–3	10	3
–1	**2**	**1**	1	–3	10	0	3	**–1**	**4**	**5**	1
3	-5	0	2					3	–11	0	2
$-1 \cdot 5 + 2 \cdot 3 = 1$				$1 \cdot 10 + -3 \cdot 3 = 1$				$-1 \cdot 55 + 4 \cdot 15 = 5$			

Перепишем уравнение $a \cdot b = m \cdot n + 1$ как $-m \cdot n + a \cdot b = 1$. Нам известны значения n и a, которые будут соответствовать числам x и y в расширенном алгоритме Евклида. После получения результата надо проверить, что $d = 1$. Кроме того, в данной задаче b должно быть положительно, а это возможно только тогда, когда u (равное $-m$), будет отрицательно. Но расширенный алгоритм Евклида не всегда даёт такой результат. Например, при $n = 10$ и $a = 3$ мы получаем $u = 1$ и $v = -3$. Чтобы решение удовлетворяло требованиям задачи, произведём следующие преобразования: $-m \cdot n + a \cdot b = u \cdot n + a \cdot v = (u \cdot n - n \cdot a) + (a \cdot v + n \cdot a) = (u - a) \cdot n + (v + n) \cdot a$, т.е. $m = a - u$, $b = v + n$.

```
алг МультипликативноОбратноеЧисло()
нач
  цел n, m, a, b, d

  ввод n, a

  если n <= 0 или a >= n то
    вывод "Некорректные исходные данные"
  иначе
    если a = 0 то
      вывод "Решения нет, т.к. число 0 не имеет мультипликативно обратного числа"
    иначе
      РасширенныйАлгоритмЕвклида(n, a, m, b, d)
      если  d <> 1 то
        вывод "Решения нет, т.к. НОД <> 1"
      иначе
        если b < 0 то
          b = b + n
          m = m – a
        всё
        вывод b, m
      всё
    всё
  всё
кон

алг РасширенныйАлгоритмЕвклида(арг цел x, y, рез цел u, v, d)
```

```
нач
  цел q, r, u1, u2, v1, v2

  u1 = 1
  u2 = 0
  v1 = 0
  v2 = 1

  пока y > 0
  нц
    q = x div y
    r = x mod y

    u = u1 - q * u2
    v = v1 - q * v2

    x = y
    y = r

    u1 = u2
    u2 = u
    v1 = v2
    v2 = v
  кц

  d = x
  u = u1
  v = v1
кон
```

Задание 4.

Робот выкладывает по спирали (например, по часовой стрелке) квадраты из реек одинаковой длины. Вершинами квадратов являются точки, в которых сходятся концы реек, а сторонами квадратов – сами рейки. Какое минимальное количество реек понадобится роботу, чтобы выложить *n* квадратов? Ограничение $1 \le n \le 15000$.

Для первого квадрата надо 4 рейки, для второго – 3 рейки, а затем прослеживается закономерность – идут повторяющиеся пары, которые начинаются с 3, а количество 2-ек увеличивается на одну при каждом новом появлении пары. Т.е. при первом появлении пары имеем: 3 2 3 2, при втором появлении: 3 2 2 3 2 2, при третьем: 3 2 2 2 3 2 2 2 … и т.д.

Первый вариант – без массива, но с досрочным выходом из алгоритма.

```
алг Робот(арг цел n)
нач
  цел s, i, j, c2

  если n = 1 то
    вернуть 4
  всё
  если n = 2 то
    вернуть 7
  всё

  c2 = 1
  s  = 7
  i  = 3
  пока true
  нц
    s = s + 3
    i = i + 1
    если  i = n то
      вернуть s
    всё
    для j от 1 до c2
    нц
      s = s + 2
      i = i + 1
      если  i = n то
        вернуть s
      всё
```

```
    кц
    s = s + 3
    i = i + 1
    если  i = n то
      вернуть s
    всё
    для j от 1 до c2
    нц
      s = s + 2
      i = i + 1
      если  i = n то
        вернуть s
      всё
    кц
    c2 = c2 + 1
  кц
кон
```

Второй вариант – с массивом, но структурно.

```
алг Робот(арг цел n)
нач
  цел s, i, j, c2, a[15000]

  a[1] = 4
  a[2] = 3
  c2 = 1
  i  = 3
  пока i <= n
  нц
    a[i] = 3
    i = i + 1
    для j от 1 до c2
    нц
      a[i] = 2
      i = i + 1
    кц
    a[i] = 3
    i = i + 1
    для j от 1 до c2
    нц
      a[i] = 2
      i = i + 1
    кц
    c2 = c2 + 1
  кц

  s = 0
  для i от 1 до n
  нц
    s = s + a[i]
  кц

  вернуть s
кон
```

Задание 5.

Результаты научных экспериментов записаны в таблицу вида (№ п/п, значение параметра, результат). Перед Вами стоит задача – упорядочить данные по убыванию значений результата. Разработать схему хранения данных и наиболее быстрый алгоритм для решения задачи. Поле «Значение параметра» – название, «результат» – целое число.

В исходной таблице содержатся данные разных типов (числа и строки). Для хранения таких данных можно было бы, в принципе, использовать три отдельных массива. Однако это будет не очень удобно при перестановках, которые потребуются для сортировки. Было бы хорошо хранить каждую запись (№ п/п, значение параметра, результат) в одной переменной. Для объединения значений разных типов в одной переменной существует специальных тип данных, который так и называется *запись*. В нашем случае нужна запись из трёх полей, и массив будет состоять из таких записей.

Одним из самых быстрых методов сортировки является *сортировка Хоара*, которую также называют *быстрой сортировкой*. Основная идея алгоритма состоит в том, что случайным образом выбирается некоторый элемент массива y, после чего массив просматривается слева, пока не встретится элемент x[i] такой, что x[i] ≥ y, а затем массив просматривается справа, пока не встретится элемент x[j] такой, что x[j] ≤ y. Эти два элемента меняются местами, и процесс просмотра, сравнения и обмена продолжается, пока i не окажется больше j. В результате массив окажется разбитым на две части – левую, в которой значения элементов будут меньше или равны y, и правую, в которой значения элементов будут больше или равны y. Далее процесс рекурсивно продолжается для левой и правой частей массива до тех пор, пока каждая часть не будет содержать один элемент – массив из одного элемента по определению считается упорядоченным. Можно также отдельно рассмотреть случай для массива из двух элементов.

```
алг Эксперименты()
нач
  тип result = запись
                 цел n
                 строка parameter
                 цел value
               конец

  цел n, i
  result r[1000]

  ввод n
  для i от 1 до n
  нц
    ввод r[i]
  кц

  QuickSort(r, 1, n)

  для i от 1 до n
  нц
    вывод r[i]
  кц
кон

алг QuickSort(арг result x[1000], арг цел n1, n2)
нач
  цел i, j
  result y, k

  если n2 - n1 = 1 то
    если x[n1].value < x[n2].value то
      y = x[n1]
      x[n1] = x[n2]
      x[n2] = y
    всё
  иначе
    если n2 - n1 > 1 то
      k = x[(n1 + n2) div 2]
      i = n1
      j = n2
      повторять
        пока (x[i].value > k.value)
        нц
          i = i + 1
        кц
        пока (x[j].value < k.value)
        нц
          j = j - 1
        кц
        если i <= j то
          y = x[i]
          x[i] = x[j]
          x[j] = y
          i = i + 1
          j = j - 1
        всё
      до i > j
```

```
      QuickSort(x, n1, j)
      QuickSort(x, i, n2)
    всё
  всё
кон
```

Однако для небольших массивов время работы мало, и для человека разница между временем работы самого быстрого и самого медленного метода сортировки может быть незаметна. Например, массив из 100 целых чисел сортируется за время, примерно равное 1 мс. Человек не в состоянии чувствовать подобные временные интервалы. А сложность реализации других методов сортировки может быть гораздо ниже. Рассмотрим для сравнения метод сортировки обменами. Это один из самых простых и широко известных методов сортировки, называемый также «методом пузырька». Метод называют пузырьковой сортировкой, потому что на каждом шаге наибольший элемент неотсортированной части подобно пузырьку газа в воде «всплывает» вверх.

Суть метода сортировки обменами состоит в следующем. На первом шаге сравниваются пары соседних элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-1]$ и $x[n]$. Если $x[i] < x[i+1]$ (для сортировки по убыванию), то элементы $x[i]$ и $x[i+1]$ меняются местами. После просмотра всего массива максимальный элемент оказывается на последнем месте. На втором шаге сравниваются пары элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-2]$ и $x[n-1]$, т.е. все элементы, кроме последнего, который уже оказался на своём месте после выполнения первого шага. На последнем шаге сравниваются элементы $x[1]$ и $x[2]$.

```
алг Sort(арг result x[1000], арг цел n)
нач
  цел i, k
  result y
  лог f

  k = 1
  повторять
    f = true
    для i от 1 до n – k
    нц
      если x[i].value < x[i + 1].value то
        y = x[i]
        x[i] = x[i + 1]
        x[i + 1] = y
        f = false
      всё
    кц
    k = k + 1
  до f
кон
```

Задание 6.

Школьники Пётр и Данила играли в игру. Результаты (количество очков) записывали в таблицу на странице блокнота (№ партии, рез-т1, рез-т2). Размер страницы M * N клеток. Перед Вами – завершённая партия. Кто набрал больше очков? В качестве результата вывести имя школьника.

```
алг Игра()
нач
  цел m, n, r[100, 3], r1, r2, i, j

  ввод m, n
  если n <> 3 или m < 1 или m > 100 то
    вывод "Неверные исходные данные"
  иначе
    для i от 1 до n
    нц
      для j от 1 до 3
      нц
        ввод  r[i, j]
      кц
    кц
```

```
    r1 = 0
    r2 = 0
    для i от 1 до n
    нц
      r1 = r1 + r[i, 2]
      r2 = r2 + r[i, 3]
    кц

    если r1 > r2 то
      вывод "Меньше очков набрал Пётр"
    иначе
      если r2 > r1 то
        вывод "Меньше очков набрал Данила"
      иначе
        вывод "Школьники набрали одинаковое количество очков"
      всё
    всё
  всё

кон
```

Решение варианта 7113

Задание 1.

Рассмотрим систему счисления, в которой основанием с.с. являются числа Фибоначчи. Алфавитом этой системы счисления являются цифры 0 и 1. В записи числа в фибоначчиевой системе не могут стоять две единицы подряд. Пример. Покажем, как записывать числа в фибоначчиевой системе счисления:

$37_{10} = 34 + 3 = 1*34+0*21+0*13+0*8+0*5+1*3+0*2+0*1= 10000100_{Fib}$;

$25_{10} = 21 + 3 + 1 = 1*21+0*13+0*8+0*5+1*3+0*2+1*1=1000101_{Fib}$.

Разработайте алгоритм перевода чисел из десятичной с.с. в фибоначчиеву.

Построим массив чисел Фибоначчи до заданного числа включительно. Для этого первые два элемента массива нужно положить равными 1 и 2 соответственно, а каждый последующий элемент будет равен сумме двух предыдущих. Положим *n* равным заданному числу. Далее проверяем все элементы массива, начиная с наибольшего. Если текущий элемент массива меньше или равен *n*, то очередная цифра числа в фибоначчиевой системе счисления будет равна 1, в противном случае очередная цифра будет равна 0. Если цифра равна 1, то *n* надо уменьшить на значение текущего элемента массива.

```
алг СистемаСчисления()
нач
  цел fib[1000], n, k, i
  лог sd

  ввод n

  fib[1] = 1
  fib[2] = 2
  k = 2
  пока fib[k] < n
  нц
    k = k + 1
    fib[k] = fib[k – 1] + fib[k – 2]
  кц

  sd = false
  для i от k до 1 шаг -1
  нц
    если fib[i] < n то
      вывод "1"
      sd = true
      n = n – fib[i]
    иначе
      если sd то
        вывод "0"
      всё
    всё
  кц
кон
```

Задание 2.

Найти все числа *N* из диапазона $N1 \le N \le N1 + 50$, которые представляются суммой четырёх квадратов натуральных чисел не единственным образом и которые имеют на заданном диапазоне наименьшее количество таких представлений.

Для каждого числа *N* из указанного диапазона необходимо проверить возможность представления в виде суммы четырёх квадратов натуральных чисел и найти количество таких представлений, а также найти количество представлений, возможных, когда натуральные числа попадают в заданный диапазон. Поэтому для каждого числа *N* будем формировать два массива. Для этого внутри цикла, перебирающего возможные значения *N*, нужны ещё четыре вложенных

цикла. В первом счётчик цикла *i* меняется от 1 до *N*, во втором счётчик цикла *j* меняется от *i* до *N*, в третьем счётчик цикла меняется от *j* до *N*, в четвёртом счётчик цикла *m* меняется от *k* до *N*. Таким образом, мы переберём все возможные комбинации натуральных чисел, при этом в комбинации можно использовать одинаковые числа, но такие комбинации как (1, 2) и (2, 1) будут считаться одинаковыми и обе использоваться не будут. Для каждой комбинации чисел проверяем, возможно ли представление нужного числа, и попадают ли исходные числа в заданный диапазон. После построения массивов положим переменную *min* равной $(N1 + 50)^4$ – значение, которое заведомо больше, чем возможное количество представлений какого-либо числа в виде суммы четырёх квадратов натуральных чисел, – и будем просматривать полученные массивы. Если текущий элемент первого массива больше 1, а текущий элемент второго массива меньше значения переменной *min*, но при этом больше 0, то переменной *min* надо присвоить новое значение – значение текущего элемента второго массива. После нахождения минимума надо снова пройти по массивам и найти те числа, для которых значение в первом массиве больше 1, а значение во втором массиве равно найденному минимуму.

```
алг СуммаКвадратов()
нач
  цел N1, A, B n, i, j, k, m, min
  цел c1[0..50], c2[0..50]

  ввод N1, A, B

  для i от 0 до 50
  нц
    c1[i] = 0
    c2[i] = 0
  кц

  для n от N1 до N1 + 50
  нц
    для i от 1 до N
    нц
      для j от i до N
      нц
        для k от j до N
        нц
          для m от k до N
          нц
            если i * i + j * j + k * k + m * m = n то
              c1[n – N1] = c1[n – N1] + 1
              если A <= i и i <= B и A <= j и j <= B и A <= k и k <= B и A <= m и m <= B то
                c2[n – N1] = c2[n – N1] + 1
              всё
            всё
          кц
        кц
      кц
    кц
  кц

  min = (N1 + 50) * (N1 + 50) * (N1 + 50) * (N1 + 50)
  для n от N1 до N1 + 50
  нц
    если c1[n – N1] > 1 и c2[n – N1] < min и c2[n – N1] > 0 то
      min = c2[n – N1]
    всё
  кц

  для n от N1 до N1 + 50
  нц
    если c2[n – N1] = min то
      вывод n
    всё
  кц
кон
```

Задание 3.

Для проверки, является ли большое целое простым, может использоваться вероятностный тест Миллера-Рабина. Пусть $p > 2$ – простое число. Представим число $p - 1$ в виде $p - 1 = 2^s \cdot d$, где d – нечётное число. Тогда для любого a из $\{1, 2, \ldots, p - 1\}$ выполняется одно из условий:

1. $a^d = 1 \{\mathrm{mod}\ p\}$

2. Существует r, $0 \le r \le s - 1$, для которого $a^{k \cdot d} = -1 \{\mathrm{mod}\ p\}$, где $k = 2^r$.

В тесте Миллера-Рабина эти проверки выполняются для t случайно выбираемых a ($t = \log_2(p)$). Разработайте алгоритм проверки вводимого числа на простоту по тесту Миллера-Рабина.

Прежде всего, определим числа s и d. Для этого необходимо число $p - 1$ поделить на 2 несколько раз, пока возможно деление без остатка. Тогда количество возможных делений даст число s, а полученный результат – число d. После это считаем число t и t раз выбираем число a из множества возможных значений. Для каждого значения a считаем значение $a^d \bmod p$. Для того чтобы не возводить большое число в большую степень, рискуя превысить максимальное число, представимое в компьютере, можно использовать следующее свойство: $(x \cdot y) \bmod n = ((x \bmod n) \cdot (y \bmod n)) \bmod n$. Если $x = a^d \bmod p$ равно 1, то можно перейти к следующему значению a. Если же $x \ne 1$, то необходимо проверить второе условие. Для этого $s - 1$ раз вычисляем $x = x^2 \bmod p$. Если ни на одном шаге x не равно $p - 1$, то возвращаем «составное». В противном случае можно продолжить проверку. Если для всех a будет выполнено хотя бы одно из двух условий, возвращаем «вероятно, простое».

```
алг ТестМиллераРабина()
нач
  цел p, s, d, a, t, r, i, j, ap, x
  лог prime

  ввод p

  d = p - 1
  s = 0
  пока d mod 2 = 0
  нц
    d = d div 2
    s = s + 1
  кц

  prime = true
  t = log2(p)
  i = 1
  пока i <= t и prime
  нц
    a = rand(p – 1)             // Случайным образом выбираем число от 1 до p – 1
    ap = a mod p
    x = 1
    для j от 1 до d
    нц
      x = (x * ap) mod p
    кц

    если x <> 1 то
      prime = false
      j = 0
      пока j <= s – 1 и не prime
      нц
        если x mod p = p – 1 то
          prime = true
        всё
        x = (x * x) mod p
        j = j + 1
      кц
    всё
    i = i + 1
  кц

  если prime то
```

```
    вывод "Вероятно, простое"
  иначе
    вывод "Составное"
  всё

кон
```

Задание 4.

Даны целые неотрицательные числа *M* и *N*, количество разрядов которых может быть велико. Разработайте алгоритм для нахождения величины N^M. Ограничение: длина чисел $0 \le N, M \le 1000$ цифр.

Столь длинные числа не могут быть представлены в современных компьютерах, тем более, что для результата понадобится уже 1000 * 1000 цифр. Поэтому для решения задачи необходимо разработать способ хранения чисел нужной длины и алгоритм умножения чисел в новом представлении. Для представления чисел можно использовать массив, каждый элемент которого хранит одну цифру числа. Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первый сомножитель надо умножить на каждую цифру второго числа, начиная с последней, и сложить результаты, учитывая, что результат умножения на очередную цифру надо сдвигать влево на соответствующее количество позиций. Для этого берётся первый сомножитель, помещается в промежуточную переменную, и каждая цифра первого сомножителя умножается на текущую цифру второго сомножителя. После умножения очередной цифры надо вычислить собственно новую цифру как остаток от деления на 10 и число для переноса в следующий разряд как результат от деления на 10. Для ускорения работы алгоритма желательно отдельно рассмотреть случаи умножения числа на цифру 0 и цифру 1. После умножения выполняем сдвиг на нужное количество позиций и прибавляем результат в общую сумму. Для возведения в степень нужно умножить число *N* на себя *M* раз.

```
алг Сдвиг(арг рез большое_число n, арг цел p)
  цел i

  для i от 1000 * 1000 до p + 1 шаг -1
  нц
    n[i] = n[i - p]
  кц
  для i от p до 1 шаг -1
  нц
    n[i] = 0
  кц
кон

алг Сумма(арг большое_число n1, n2)
  большое_число res
  цел i, c                                    // c – переменная для переноса в следующий разряд

  c = 0
  для i от 1 до  1000 * 1000                  // Просматриваем все элементы массива, т.к., во-первых,
  нц                                          // мы не знаем, сколько цифр содержат числа, а, во-вторых,
    res[i] = n1[i] + n2[i] + c                // старшие разряды результата в любом случае надо обнулить
    если res[i] > 9 то
      c = 1
      res[i] = res[i] – 10
    иначе
      c = 0
    всё
  кц
  вернуть res
кон

алг Произведение(арг большое_число n1, n2)
  большое_число res, m
  цел i, j, c                                 // c – переменная для переноса в следующий разряд

  res = 0                                     // Будем считать, что такая операция тоже уже реализована
  для i от 1 до  1000 * 1000
```

```
  нц
    если n2[i] = 1 то
      m = n1
    иначе
      если 2 <= n2[i] и n2[i] <= 9 то
        c = 0
        для j от 1 до  1000 * 1000
        нц
          m[j] = n1[j] * n2[i]
          c = m[j] div 10
          m[j] = m[j] mod 10
        кц
      всё
    всё
    Сдвиг(m, i – 1)
    res = res + m
  кц
  вернуть res
кон

алг МеньшеИлиРавно(арг большое_число n1, n2)
  цел i

  для i от 1000 * 1000 до 1 шаг -1
  нц
    если n1[i] < n2[i] то
      вернуть true
    иначе
      если n1[i] > n2[i] то
        вернуть false
      всё
    всё
  кц
  вернуть true
кон

алг Степень()
  большое_число n, m, i, s, res

  ввод n, m

  res = 1
  i = 1
  s = 1
  пока МеньшеИлиРавно(i, m)
  нц
    res = Произведение(res, n)
    i = Сумма(i, s)
  кц

  вывод res
кон
```

Задание 5.

На листе блокнота школьник Матвей записал несколько строчек из чисел. Размер страницы $M \cdot N$ клеток. Лист вырвал из блокнота и забрал его одноклассник Николай. Хобби Николая – упорядочение чисел. Разработайте алгоритм, который упорядочит все числа на листе в убывающем порядке.

Алгоритмы сортировки обычно рассчитаны на одномерные массивы. Можно, конечно, попробовать модифицировать какой-нибудь алгоритм для обработки матрицы, но придётся переходить от последнего элемента одной строки к первому элементу следующей строки, но это потребует много проверок и дополнительных действий. Поэтому проще переписать матрицу в одномерный массив, упорядочить его, а затем переписать новый массив обратно в матрицу. Рассмотрим возможные методы сортировки.

Сортировка обменами. Одним из самых простых и широко известных методов сортировки является метод сортировки обменами, называемый также «методом пузырька». Однако скорость работы этого метода оставляет желать лучшего. Этот метод, однако, чувствителен к отсортированным данным, поэтому алгоритм имеет смысл использовать в тех случаях, когда предполагается сортировка практически упорядоченных данных. Метод называют пузырьковой сортировкой, потому что на каждом шаге наибольший элемент неотсортированной части подобно пузырьку газа в воде «всплывает» вверх.

Суть метода сортировки обменами состоит в следующем. На первом шаге сравниваются пары соседних элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-1]$ и $x[n]$. Если $x[i] < x[i+1]$ (для сортировки по убыванию), то элементы $x[i]$ и $x[i+1]$ меняются местами. После просмотра всего массива максимальный элемент оказывается на последнем месте. На втором шаге сравниваются пары элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-2]$ и $x[n-1]$, т.е. все элементы, кроме последнего, который уже оказался на своём месте после выполнения первого шага. На последнем шаге сравниваются элементы $x[1]$ и $x[2]$.

Сортировка выбором. Суть метода сортировки выбором состоит в следующем. На первом шаге в массиве находится минимальный (для сортировки по возрастанию) или максимальный (для сортировки по убыванию) элемент. Этот элемент меняется местами с первым элементом массива. На втором шаге мы рассматриваем часть массива, начинающуюся со второго элемента. Находим минимальный или максимальный элемент этой части и меняем его местами со вторым элементом. Процесс повторяет $n-1$ раз – на последнем шаге рассматривается часть массива, состоящая из двух последних элементов.

Сортировка включениями. Одним из наиболее простых и естественных методов внутренней сортировки является сортировка простым включением. Идея алгоритма очень проста. Для каждого элемента массива, начиная со второго, производится сравнение с элементами с меньшим индексом (значение элемента $x[i]$ последовательно сравнивается с элементами $x[i-1]$, $x[i-2]$ ...), и до тех пор, пока для очередного элемента $x[j]$ выполняется соотношение $x[j] < x[i]$ (для сортировки по убыванию), элемент $x[j]$ сдвигает на одну позицию вправо. Если удаётся встретить такой элемент $x[j]$, что $x[j] \le x[i]$, или если достигнута нижняя граница массива, в $(j+1)$-ую позицию записывается значение элемента $x[i]$ и производится переход к обработке элемента $x[i+1]$. Поскольку при сдвиге элементов элемент $x[i]$ будет перезаписан, предварительно необходимо запомнить значение этого элемента. Этот метод называется также «карточной сортировкой», поскольку часто используется игроками для сортировки выпавших карт.

Сортировка разделением. Этот метод называется также *сортировкой Хоара* по имени разработчика или *быстрой сортировкой*, поскольку он действительно оказывается одним из самых быстрых методов сортировки. Основная идея алгоритма состоит в том, что случайным образом выбирается некоторый элемент массива y, после чего массив просматривается слева, пока не встретится элемент $x[i]$ такой, что $x[i] \le y$, а затем массив просматривается справа, пока не встретится элемент $x[j]$ такой, что $x[j] \ge y$. Эти два элемента меняются местами, и процесс просмотра, сравнения и обмена продолжается, пока i не окажется больше j. В результате массив окажется разбитым на две части – левую, в которой значения элементов будут больше или равны y, и правую, в которой значения элементов будут меньше или равны y. Далее процесс рекурсивно продолжается для левой и правой частей массива до тех пор, пока каждая часть не будет содержать один элемент – массив из одного элемента по определению считается упорядоченным. Можно также отдельно рассмотреть случай для массива из двух элементов. Алгоритм действительно работает очень быстро – время его работы оказывается на 2-3 порядка (!) меньше, чем время работы предыдущих рассмотренных алгоритмов.

```
алг СортировкаМатрицы()
  цел m, n, i, j, k
  вещ matrix[100, 100], mas[10000]

  ввод m, n
  для i от 1 до m
```

```
  нц
    для j от 1 до n
    нц
      ввод matrix[i, j]
    кц
  кц

  k = 0
  для i от 1 до m
  нц
    для j от 1 до n
    нц
      k = k + 1
      mas[k] = matrix[i, j]
    кц
  кц

  QuickSort(mas, 1, k)

  k = 0
  для i от 1 до m
  нц
    для j от 1 до n
    нц
      k = k + 1
      matrix[i, j] = mas[k]
    кц
  кц

  для i от 1 до m
  нц
    для j от 1 до n
    нц
      вывод matrix[i, j]
    кц
  кц
кон

алг QuickSort(арг рез вещ x[10000], арг цел n1, n2)
нач
  цел i, j,
  вещ y, k

  если n2 - n1 = 1 то
    если x[n1] < x[n2] то
      y = x[n1]
      x[n1] = x[n2]
      x[n2] = y
    всё
  иначе
    если n2 - n1 > 1 то
      k = x[(n1 + n2) div 2]
      i = n1
      j = n2
      повторять
        пока (x[i] > k)
        нц
          i = i + 1
        кц
        пока (x[j] < k)
        нц
          j = j - 1
        кц
        если i <= j то
          y = x[i]
          x[i] = x[j]
          x[j] = y
          i = i + 1
          j = j - 1
        всё
      до i > j
      QuickSort(x, n1, j)
      QuickSort(x, i, n2)
    всё
```

```
  всё
кон
```

Задание 6 (Схема решения).

В лесничестве провели эксперимент по выращиванию елей разных видов. Виды отличаются друг от друга по числу ярусов веток (от N до M). К сожалению, печатная версия плана рассадки сгорела в пожаре, а электронной не было. Перед Вами стоит задача посчитать количество елей каждого вида в настоящий момент. Участок эксперимента изначально имел прямоугольную форму ($L \cdot K$ км) и было высажено Q саженцев, не все из которых есть сейчас. Каждая ель занимает квадрат не более 2 м^2 площади. Разработайте алгоритм и схему хранения данных.

При решении данной задачи можно использовать различные способы хранения данных. Поскольку участок имеет прямоугольную форму, первое, что приходит в голову – использовать матрицу. Однако поскольку площадь, занимаемая каждой елью может быть разной, ели могут быть рассажены не рядами, и в этом случае форма рассадки не будет совпадать с матричной. Поэтому можно использовать массив (в задаче сказано, что было высажено P саженцев) или список. **Список** – это динамически создаваемая структура из однотипных элементов, в которой каждый элемент хранит адрес следующего, а иногда также и предыдущего, элемента. Элементы списка, в отличие от элементов массива, не должны располагаться в памяти последовательно единым целым.

Достоинства списков:

- размер списка ограничивается только доступным объёмом машинной памяти (массивы имеют ограничения на максимальный размер);
- при изменении последовательности элементов списка требуется не перемещение данных в памяти, а только коррекция адресов.

Недостатки списков:

- на адреса расходуется дополнительная память;
- доступ к элементам связной структуры может быть менее эффективным по времени, поскольку невозможно обратиться непосредственно к i-ому элементу списка – для того чтобы добраться до нужного элемента, надо пройти до него от начала списка.

Таким образом, пройдя по участку, мы составим массив или список с данными: в каждом элементе будет храниться количество ярусов веток у данного саженца. В список элементы добавляются в процессе работы по одному. Потом проходим по массиву или списку и строим массив, в котором каждому возможному количеству ярусов веток будет соответствовать количество саженцев с данным количеством ярусов веток. Если использовался список, при завершении работы его надо удалить.

В приведенном ниже алгоритме не рассматриваются ограничения на исходные данные (L*K*10^6 ≤ 2*Q).

Рассмотрим вариант со списком. Количество елей каждого вида будем хранить в массиве, при этом будем использовать количество ярусов веток как индекс массива. Если N окажется больше 1, то часть элементов массива использоваться не будет, но такой подход удобнее и проще. Будем также считать, что количество ярусов веток не превышает 10.

```
алг Лесничество()
  цел n, m, levels[10], i

  ввод n, m

  создать список
  для всех елей
  нц
```

```
    добавить в список новый элемент и сделать его текущим
    записать в текущий элемент количество ярусов очередной ели
  кц

  для i от n до m
  нц
    levels[i] = 0
  кц

  установить текущий элемент на начало списка
  пока список не закончился
  нц
    для i от n до m
    нц
      если текущее значение в списке = i то
        levels[i] = levels[i] + 1
      всё
    кц
    перейти к следующему элементу списка
  кц

  для i от n до m
  нц
    вывод i, levels[i]
  кц

  удалить список
кон
```

Решение вариантов 7101 и 7102

Задание 1.

Разработайте алгоритм для перевода целых чисел, представленных в десятичной/восьмеричной системе счисления, в римскую систему счисления. В качестве исходных данных выступает число в десятичной/восьмеричной системе счисления. В качестве результата необходимо вывести представление в виде римских цифр.

Римские цифры появились около 500 лет до нашей эры у этрусков и представляют собой комбинацию буквенных обозначений, каждая буква обозначает число (цифру) из списка: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000. Натуральные числа записываются при помощи повторения этих цифр. При этом некоторые из цифр (I, X, C, M) могут повторяться, но не более трёх раз, таким образом, с их помощью можно записать любое целое число не более 3999 (MMMCMXCIX). Числа составляются путём составления «вычитающих» или «добавляющих» комбинаций.

- Если большая цифра стоит перед меньшей, то они складываются (принцип сложения). Правило применяется до трёх подряд, потом – смотри следующее правило.
- Если же меньшая цифра стоит перед большей, то меньшая вычитается из большей (принцип вычитания). Правило применяется только во избежание четырёхкратного повторения одной и той же цифры. Существует шесть вариантов использования правила вычитания: IV = 4, IX = 9, XL = 40, XC = 90, CD = 400, CM = 900.

Число / цифра 0 (ноль) не входит в ряд римских цифр.

```
алг РимскоеЧисло()
нач
  строка arab_num, roman_num
  цел num, i
  цел    values[13] = {1000, 900,  500, 400,  100, 90,   50,  40,   10,  9,    5,   4,    1}
  строка digits[13] = {"M",  "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"}

  ввод arab_num
  num = atoi(arab_num, 10)         // 10 – основание используемой системы счисления. В другом варианте будет 8
  если num <= 0 то
    вывод "Число введено некорректно"
  иначе
    если num > 3999 то
      вывод "Число слишком большое"
    иначе
      ronam_num = ""
      для i от 1 до 13
      нц
        пока num – values[i] >= 0
        нц
          num = num – values[i]
          roman_num = roman_num + digits[i]     // «+» – конкатенация строк
        кц
      кц
      вывод roman_num
    всё
  всё
кон

алг atoi(арг строка str, арг цел radix)
нач
  цел i, d, num

  num = 0
  для i от 1 до Length(str)
  нц
    d = ord(str[i]) – ord('0')           // ord – функция, определяющая код символа
    если d < 0 или d >= radix то
      вернуть -1                         // Поскольку в римской системе нет 0 и отрицательных чисел,
    иначе                                // значение -1 будет признаком ошибки при вводе
      num = num * radix + d
    всё
```

```
  кц
  вернуть num
кон
```

Задание 2. (схема решения)

Двоично-десятичная система счисления - это форма записи рациональных чисел, при которой каждый десятичный разряд числа записывается в виде его четырёхбитного двоичного кода. Например, десятичное число 21_{10} будет записано в двоичной системе счисления как 10101_2, а в двоично-десятичном коде как $0010\ 0001_{2\text{-}10}$.В каких случаях её применение целесообразно?

Преимущества 2-10 с.с.

- упрощён вывод чисел на индикацию — вместо последовательного деления на 10 требуется просто вывести на индикацию каждый полубайт.
- Для дробных чисел (как с фиксированной, так и с плавающей запятой) при переводе в «обычную» десятичную с.с. и наоборот не теряется точность.
- Упрощены умножение и деление на 10, а также округление.

Недостатки:

- Требует больше памяти.
- Усложнены арифметические операции. Так как в 8421-BCD используются только 10 возможных комбинаций 4-х битового поля вместо 16, существуют запрещённые комбинации битов: 1010(1010), 1011(1110), 1100(1210), 1101(1310), 1110(1410) и 1111(1510).

Таким образом, применение 2-10 с.с. целесообразно для операций умножения/деления на 10 и перевода дробных чисел из/в десятичную с.с.

Задание 3.

Мультипликативно обратным к целому числу a по модулю n является целое число b ($0 \le a < n$ и $0 \le b < n$), для которого справедливо сравнение $a \cdot b \equiv 1 \{\mathrm{mod}\ n\}$ (два целых числа сравнимы по модулю n, если они дают одинаковые остатки при делении на n). Мультипликативно обратное число существует тогда и только тогда, когда a и n взаимно простые числа (их наибольший общий делитель равен 1).

Разработайте алгоритм для нахождения мультипликативно обратного числа по задаваемому значению модуля для задаваемого целого числа.

Примеры

Входные данные		**Результат работы** (b)
Модуль (n)	**Целое число** (a)	
5	3	2
10	3	7
10	5	Не существует

Заметим, что если $a \ge 2$, то b не может быть равно 1, т.к. $a \cdot b = a$, и поскольку $a < n$, $a \cdot b \bmod n = a$, и будет больше и равно 2, что нас не устраивает. Соответственно, если $a = 1$, то b не может быть больше или равно 2 по тем же причинам, и может быть равно только 1. Получается, что число 1 мультипликативно обратно самому себе по любому модулю.

Далее, $a \cdot b \equiv 1 \{\mathrm{mod}\ n\}$ можно записать как $a \cdot b = m \cdot n + 1$, где m – некий коэффициент (в приведённых примерах $m = 1$ ($3 \cdot 2 = 1 \cdot 5 + 1$) и $m = 2$ ($3 \cdot 7 = 2 \cdot 10 + 1$)). В этом уравнении два неизвестных – b и m, но путём рассуждений можно прийти к выводу, что $2 \le b < n$ и $0 \le m \le a$.

Поэтому самым примитивным алгоритмом решения этой задачи будет подбор значения *b* или *m* с помощью проверки всех возможных значений.

```
алг МультипликативноОбратноеЧисло()
нач
  цел n, m, a, b
  лог found

  ввод n, a

  если n <= 0 или a >= n то
    вывод "Некорректные исходные данные"
  иначе
    если a = 0 то
      вывод "Решения нет, т.к. число 0 не имеет мультипликативно обратного числа"
    иначе
      если НОД(n, a) <> 1 то
        вывод "Решения нет, т.к. НОД <> 1"
      иначе
        если a = 1 то
          b = 1
          m = 0
        иначе

          b = 2                                  // Подбор значения b
          found = false
          пока не found
          нц
            если (a * b) mod n = 1 то
              m = (a * b - 1) div n
              found = true
            всё
          кц

          m = 0                                  // Подбор значения m
          found = false
          пока не found
          нц
            если (m * n + 1) mod a = 0 то
              b = (m * n + 1) div a
              found = true
            всё
          кц

        всё
        вывод b, m
      всё
    всё
  всё
кон

алг НОД(арг цел x, y)
нач
  пока x <> y
  нц
    если x > y то
      x = x - y
    иначе
      y = y - x
    всё
  кц
  вернуть x
кон
```

Приведённые способы решения сопоставимы по времени работы, но оптимальными не являются. Попробуем найти другой вариант.

Всем известен алгоритм Евклида, который находит наибольший общий делитель двух чисел (только что он использовался при проверке существования решения данной задачи). Но существует также и расширенный алгоритм Евклида, который для двух целых чисел x и y находит коэффициенты u и v, такие, что $x \cdot u + y \cdot v = d$, где d – наибольший общий делитель чисел x и y ($x \geq y$).

Алгоритм заключается в следующем. Сначала надо составить два исходных уравнения.

$x \cdot u_1 + y \cdot v_1 = x$

$x \cdot u_2 + y \cdot v_2 = y$

Для того чтобы уравнения выполнялись, коэффициенты должны иметь следующие значения: $u_1 = 1$, $v_1 = 0$, $u_2 = 0$, $v_2 = 1$. После этого из первого уравнения вычитается второе, умноженное на результат целочисленного деления x на y (обозначим как q).

$x \cdot (u_1 - q \cdot u_2) + y \cdot (v_1 - q \cdot v_2) = x - q \cdot y$

В правой части уравнения получается остаток от деления x на y. На следующем шаге те же действия выполняются над вторым и третьим уравнениями. Алгоритм прекращает работу, когда правая часть уравнения становится равной 0. При этом в качестве результата берутся значения коэффициентов, полученные на предпоследнем шаге. Значение в правой части уравнения на этом же предпоследнем шаге даёт наибольший общий делитель чисел x и y.

Рассмотрим примеры.

u	***v***	***r***	***q***	***u***	***V***	***r***	***q***	***u***	***v***	***r***	***q***
1	0	5		1	0	10		1	0	55	
0	1	3		0	1	3		0	1	15	
1	–1	2	1	**1**	**–3**	**1**	3	1	–3	10	3
–1	**2**	**1**	1	–3	10	0	3	**–1**	**4**	**5**	1
3	-5	0	2					3	–11	0	2
$-1 \cdot 5 + 2 \cdot 3 = 1$				$1 \cdot 10 + -3 \cdot 3 = 1$				$-1 \cdot 55 + 4 \cdot 15 = 5$			

Перепишем уравнение $a \cdot b = m \cdot n + 1$ как $-m \cdot n + a \cdot b = 1$. Нам известны значения n и a, которые будут соответствовать числам x и y в расширенном алгоритме Евклида. После получения результата надо проверить, что $d = 1$. Кроме того, в данной задаче b должно быть положительно, а это возможно только тогда, когда u (равное $-m$), будет отрицательно. Но расширенный алгоритм Евклида не всегда даёт такой результат. Например, при $n = 10$ и $a = 3$ мы получаем $u = 1$ и $v = -3$. Чтобы решение удовлетворяло требованиям задачи, произведём следующие преобразования: $-m \cdot n + a \cdot b = u \cdot n + a \cdot v = (u \cdot n - n \cdot a) + (a \cdot v + n \cdot a) = (u - a) \cdot n + (v + n) \cdot a$, т.е. $m = a - u$, $b = v + n$.

```
алг МультипликативноОбратноеЧисло()
нач
  цел n, m, a, b, d

  ввод n, a

  если n <= 0 или a >= n то
    вывод "Некорректные исходные данные"
  иначе
    если a = 0 то
      вывод "Решения нет, т.к. число 0 не имеет мультипликативно обратного числа"
    иначе
      РасширенныйАлгоритмЕвклида(n, a, m, b, d)
      если  d <> 1 то
        вывод "Решения нет, т.к. НОД <> 1"
      иначе
        если b < 0 то
          b = b + n
          m = m - a
        всё
        вывод b, m
      всё
    всё
  всё
кон

алг РасширенныйАлгоритмЕвклида(арг цел x, y, рез цел u, v, d)
```

```
нач
  цел q, r, u1, u2, v1, v2

  u1 = 1
  u2 = 0
  v1 = 0
  v2 = 1

  пока y > 0
  нц
    q = x div y
    r = x mod y

    u = u1 - q * u2
    v = v1 - q * v2

    x = y
    y = r

    u1 = u2
    u2 = u
    v1 = v2
    v2 = v
  кц

  d = x
  u = u1
  v = v1
кон
```

Задание 4.

Робот выкладывает по спирали (например, по часовой стрелке) квадраты из реек одинаковой длины. Вершинами квадратов являются точки, в которых сходятся концы реек, а сторонами квадратов – сами рейки. Какое минимальное количество реек понадобится роботу, чтобы выложить *n* квадратов? Ограничение $1 \le n \le 15000$.

Для первого квадрата надо 4 рейки, для второго – 3 рейки, а затем прослеживается закономерность – идут повторяющиеся пары, которые начинаются с 3, а количество 2-ек увеличивается на одну при каждом новом появлении пары. Т.е. при первом появлении пары имеем: 3 2 3 2, при втором появлении: 3 2 2 3 2 2, при третьем: 3 2 2 2 3 2 2 2 … и т.д.

Первый вариант – без массива, но с досрочным выходом из алгоритма.

```
алг Робот(арг цел n)
нач
  цел s, i, j, c2

  если n = 1 то
    вернуть 4
  всё
  если n = 2 то
    вернуть 7
  всё

  c2 = 1
  s  = 7
  i  = 3
  пока true
  нц
    s = s + 3
    i = i + 1
    если  i = n то
      вернуть s
    всё
    для j от 1 до c2
    нц
      s = s + 2
      i = i + 1
      если  i = n то
        вернуть s
      всё
```

```
    кц
    s = s + 3
    i = i + 1
    если  i = n то
      вернуть s
    всё
    для j от 1 до c2
    нц
      s = s + 2
      i = i + 1
      если  i = n то
        вернуть s
      всё
    кц
    c2 = c2 + 1
  кц
кон
```

Второй вариант – с массивом, но структурно.

```
алг Робот(арг цел n)
нач
  цел s, i, j, c2, a[15000]

  a[1] = 4
  a[2] = 3
  c2 = 1
  i  = 3
  пока i <= n
  нц
    a[i] = 3
    i = i + 1
    для j от 1 до c2
    нц
      a[i] = 2
      i = i + 1
    кц
    a[i] = 3
    i = i + 1
    для j от 1 до c2
    нц
      a[i] = 2
      i = i + 1
    кц
    c2 = c2 + 1
  кц

  s = 0
  для i от 1 до n
  нц
    s = s + a[i]
  кц

  вернуть s
кон
```

Задание 5.

Результаты научных экспериментов записаны в таблицу вида (№ п/п, значение параметра, результат). Перед Вами стоит задача – упорядочить данные по возрастанию значений результата. Разработать схему хранения данных и наиболее быстрый алгоритм для решения задачи. Поле «Значение параметра» – название, «результат» – целое число.

В исходной таблице содержатся данные разных типов (числа и строки). Для хранения таких данных можно было бы, в принципе, использовать три отдельных массива. Однако это будет не очень удобно при перестановках, которые потребуются для сортировки. Было бы хорошо хранить каждую запись (№ п/п, значение параметра, результат) в одной переменной. Для объединения значений разных типов в одной переменной существует специальных тип данных, который так и называется *запис*ь. В нашем случае нужна запись из трёх полей, и массив будет состоять из таких записей.

Одним из самых быстрых методов сортировки является *сортировка Хоара*, которую также называют *быстрой сортировкой*. Основная идея алгоритма состоит в том, что случайным образом выбирается некоторый элемент массива y, после чего массив просматривается слева, пока не встретится элемент x[i] такой, что x[i] ≥ y, а затем массив просматривается справа, пока не встретится элемент x[j] такой, что x[j] ≤ y. Эти два элемента меняются местами, и процесс просмотра, сравнения и обмена продолжается, пока i не окажется больше j. В результате массив окажется разбитым на две части – левую, в которой значения элементов будут меньше или равны y, и правую, в которой значения элементов будут больше или равны y. Далее процесс рекурсивно продолжается для левой и правой частей массива до тех пор, пока каждая часть не будет содержать один элемент – массив из одного элемента по определению считается упорядоченным. Можно также отдельно рассмотреть случай для массива из двух элементов.

```
алг Эксперименты()
нач
  тип result = запись
                 цел n
                 строка parameter
                 цел value
               конец

  цел n, i
  result r[1000]

  ввод n
  для i от 1 до n
  нц
    ввод r[i]
  кц

  QuickSort(r, 1, n)

  для i от 1 до n
  нц
    вывод r[i]
  кц
кон

алг QuickSort(арг result x[1000], арг цел n1, n2)
нач
  цел i, j
  result y, k

  если n2 - n1 = 1 то
    если x[n1].value > x[n2].value то
      y = x[n1]
      x[n1] = x[n2]
      x[n2] = y
    всё
  иначе
    если n2 - n1 > 1 то
      k = x[(n1 + n2) div 2]
      i = n1
      j = n2
      повторять
        пока (x[i].value < k.value)
        нц
          i = i + 1
        кц
        пока (x[j].value > k.value)
        нц
          j = j - 1
        кц
        если i <= j то
          y = x[i]
          x[i] = x[j]
          x[j] = y
          i = i + 1
          j = j - 1
        всё
      до i > j
```

```
      QuickSort(x, n1, j)
      QuickSort(x, i, n2)
    всё
  всё
кон
```

Однако для небольших массивов время работы мало, и для человека разница между временем работы самого быстрого и самого медленного метода сортировки может быть незаметна. Например, массив из 100 целых чисел сортируется за время, примерно равное 1 мс. Человек не в состоянии чувствовать подобные временные интервалы. А сложность реализации других методов сортировки может быть гораздо ниже. Рассмотрим для сравнения метод сортировки обменами. Это один из самых простых и широко известных методов сортировки, называемый также «методом пузырька». Метод называют пузырьковой сортировкой, потому что на каждом шаге наибольший элемент неотсортированной части подобно пузырьку газа в воде «всплывает» вверх.

Суть метода сортировки обменами состоит в следующем. На первом шаге сравниваются пары соседних элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-1]$ и $x[n]$. Если $x[i] > x[i+1]$ (для сортировки по возрастанию), то элементы $x[i]$ и $x[i+1]$ меняются местами. После просмотра всего массива максимальный элемент оказывается на последнем месте. На втором шаге сравниваются пары элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-2]$ и $x[n-1]$, т.е. все элементы, кроме последнего, который уже оказался на своём месте после выполнения первого шага. На последнем шаге сравниваются элементы $x[1]$ и $x[2]$.

```
алг Sort(арг result x[1000], арг цел n)
нач
  цел i, k
  result y
  лог f

  k = 1
  повторять
    f = true
    для i от 1 до n - k
    нц
      если x[i].value > x[i + 1].value то
        y = x[i]
        x[i] = x[i + 1]
        x[i + 1] = y
        f = false
      всё
    кц
    k = k + 1
  до f
кон
```

Задание 6.

Школьники Пётр и Данила играли в игру. Результаты (количество очков) записывали в таблицу на странице блокнота (№ партии, рез-т1, рез-т2). Размер страницы M * N клеток. Перед Вами – завершённая партия. Кто набрал меньше очков? В качестве результата вывести имя школьника.

```
алг Игра()
нач
  цел m, n, r[100, 3], r1, r2, i, j

  ввод m, n
  если n <> 3 или m < 1 или m > 100 то
    вывод "Неверные исходные данные"
  иначе
    для i от 1 до n
    нц
      для j от 1 до 3
      нц
        ввод  r[i, j]
      кц
    кц
```

```
    r1 = 0
    r2 = 0
    для i от 1 до n
    нц
      r1 = r1 + r[i, 2]
      r2 = r2 + r[i, 3]
    кц

    если r1 < r2 то
      вывод "Меньше очков набрал Пётр"
    иначе
      если r2 < r1 то
        вывод "Меньше очков набрал Данила"
      иначе
        вывод "Школьники набрали одинаковое количество очков"
      всё
    всё
  всё

кон
```

Решение варианта 7103

Задание 1.

Рассмотрим систему счисления, в которой основанием с.с. являются числа Фибоначчи. Алфавитом этой системы счисления являются цифры 0 и 1. В записи числа в фибоначчиевой системе не могут стоять две единицы подряд. Пример. Покажем, как записывать числа в фибоначчиевой системе счисления:

$37_{10} = 34 + 3 = 1*34+0*21+0*13+0*8+0*5+1*3+0*2+0*1= 10000100_{Fib}$;

$25_{10} = 21 + 3 + 1 = 1*21+0*13+0*8+0*5+1*3+0*2+1*1=1000101_{Fib}$.

Разработайте алгоритм перевода чисел из десятичной с.с. в фибоначчиеву.

Построим массив чисел Фибоначчи до заданного числа включительно. Для этого первые два элемента массива нужно положить равными 1 и 2 соответственно, а каждый последующий элемент будет равен сумме двух предыдущих. Положим *n* равным заданному числу. Далее проверяем все элементы массива, начиная с наибольшего. Если текущий элемент массива меньше или равен *n*, то очередная цифра числа в фибоначчиевой системе счисления будет равна 1, в противном случае очередная цифра будет равна 0. Если цифра равна 1, то *n* надо уменьшить на значение текущего элемента массива.

```
алг СистемаСчисления()
нач
  цел fib[1000], n, k, i
  лог sd

  ввод n

  fib[1] = 1
  fib[2] = 2
  k = 2
  пока fib[k] < n
  нц
    k = k + 1
    fib[k] = fib[k – 1] + fib[k – 2]
  кц

  sd = false
  для i от k до 1 шаг -1
  нц
    если fib[i] < n то
      вывод "1"
      sd = true
      n = n – fib[i]
    иначе
      если sd то
        вывод "0"
      всё
    всё
  кц
кон
```

Задание 2.

Найти все числа *N* из диапазона $N1 \le N \le N1 + 50$, которые представляются суммой четырёх квадратов натуральных чисел не единственным образом и которые имеют на заданном диапазоне наименьшее количество таких представлений.

Для каждого числа *N* из указанного диапазона необходимо проверить возможность представления в виде суммы четырёх квадратов натуральных чисел и найти количество таких представлений, а также найти количество представлений, возможных, когда натуральные числа попадают в заданный диапазон. Поэтому для каждого числа *N* будем формировать два массива. Для этого внутри цикла, перебирающего возможные значения *N*, нужны ещё четыре вложенных

цикла. В первом счётчик цикла *i* меняется от 1 до *N*, во втором счётчик цикла *j* меняется от *i* до *N*, в третьем счётчик цикла меняется от *j* до *N*, в четвёртом счётчик цикла *m* меняется от *k* до *N*. Таким образом, мы переберём все возможные комбинации натуральных чисел, при этом в комбинации можно использовать одинаковые числа, но такие комбинации как (1, 2) и (2, 1) будут считаться одинаковыми и обе использоваться не будут. Для каждой комбинации чисел проверяем, возможно ли представление нужного числа, и попадают ли исходные числа в заданный диапазон. После построения массивов положим переменную *min* равной $(N1 + 50)^4$ – значение, которое заведомо больше, чем возможное количество представлений какого-либо числа в виде суммы четырёх квадратов натуральных чисел, – и будем просматривать полученные массивы. Если текущий элемент первого массива больше 1, а текущий элемент второго массива меньше значения переменной *min*, но при этом больше 0, то переменной *min* надо присвоить новое значение – значение текущего элемента второго массива. После нахождения минимума надо снова пройти по массивам и найти те числа, для которых значение в первом массиве больше 1, а значение во втором массиве равно найденному минимуму.

```
алг СуммаКвадратов()
нач
  цел N1, A, B n, i, j, k, m, min
  цел c1[0..50], c2[0..50]

  ввод N1, A, B

  для i от 0 до 50
  нц
    c1[i] = 0
    c2[i] = 0
  кц

  для n от N1 до N1 + 50
  нц
    для i от 1 до N
    нц
      для j от i до N
      нц
        для k от j до N
        нц
          для m от k до N
          нц
            если i * i + j * j + k * k + m * m = n то
              c1[n – N1] = c1[n – N1] + 1
              если A <= i и i <= B и A <= j и j <= B и A <= k и k <= B и A <= m и m <= B то
                c2[n – N1] = c2[n – N1] + 1
              всё
            всё
          кц
        кц
      кц
    кц
  кц

  min = (N1 + 50) * (N1 + 50) * (N1 + 50) * (N1 + 50)
  для n от N1 до N1 + 50
  нц
    если c1[n – N1] > 1 и c2[n – N1] < min и c2[n – N1] > 0 то
      min = c2[n – N1]
    всё
  кц

  для n от N1 до N1 + 50
  нц
    если c2[n – N1] = min то
      вывод n
    всё
  кц
кон
```

Задание 3.

Для проверки, является ли большое целое простым, может использоваться вероятностный тест Миллера-Рабина. Пусть $p > 2$ – простое число. Представим число $p - 1$ в виде $p - 1 = 2^s \cdot d$, где d – нечётное число. Тогда для любого a из $\{1, 2, \ldots, p - 1\}$ выполняется одно из условий:

1. $a^d = 1 \{\text{mod } p\}$

2. Существует r, $0 \le r \le s - 1$, для которого $a^{k \cdot d} = -1 \{\text{mod } p\}$, где $k = 2^r$.

В тесте Миллера-Рабина эти проверки выполняются для t случайно выбираемых a ($t = \log_2(p)$). Разработайте алгоритм проверки вводимого числа на простоту по тесту Миллера-Рабина.

Прежде всего, определим числа s и d. Для этого необходимо число $p - 1$ поделить на 2 несколько раз, пока возможно деление без остатка. Тогда количество возможных делений даст число s, а полученный результат – число d. После это считаем число t и t раз выбираем число a из множества возможных значений. Для каждого значения a считаем значение $a^d \bmod p$. Для того чтобы не возводить большое число в большую степень, рискуя превысить максимальное число, представимое в компьютере, можно использовать следующее свойство: $(x \cdot y) \bmod n = ((x \bmod n) \cdot (y \bmod n)) \bmod n$. Если $x = a^d \bmod p$ равно 1, то можно перейти к следующему значению a. Если же $x \ne 1$, то необходимо проверить второе условие. Для этого $s - 1$ раз вычисляем $x = x^2 \bmod p$. Если ни на одном шаге x не равно $p - 1$, то возвращаем «составное». В противном случае можно продолжить проверку. Если для всех a будет выполнено хотя бы одно из двух условий, возвращаем «вероятно, простое».

```
алг ТестМиллераРабина()
нач
  цел p, s, d, a, t, r, i, j, ap, x
  лог prime

  ввод p

  d = p - 1
  s = 0
  пока d mod 2 = 0
  нц
    d = d div 2
    s = s + 1
  кц

  prime = true
  t = log2(p)
  i = 1
  пока i <= t и prime
  нц
    a = rand(p – 1)             // Случайным образом выбираем число от 1 до p – 1
    ap = a mod p
    x = 1
    для j от 1 до d
    нц
      x = (x * ap) mod p
    кц

    если x <> 1 то
      prime = false
      j = 0
      пока j <= s – 1 и не prime
      нц
        если x mod p = p – 1 то
          prime = true
        всё
        x = (x * x) mod p
        j = j + 1
      кц
    всё
    i = i + 1
  кц

  если prime то
```

```
    вывод "Вероятно, простое"
  иначе
    вывод "Составное"
  всё

кон
```

Задание 4.

Даны целые неотрицательные числа *M* и *N*, количество разрядов которых может быть велико. Разработайте алгоритм для нахождения величины N^M. Ограничение: длина чисел $0 \le N, M \le 1000$ цифр.

Столь длинные числа не могут быть представлены в современных компьютерах, тем более, что для результата понадобится уже 1000 * 1000 цифр. Поэтому для решения задачи необходимо разработать способ хранения чисел нужной длины и алгоритм умножения чисел в новом представлении. Для представления чисел можно использовать массив, каждый элемент которого хранит одну цифру числа. Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первый сомножитель надо умножить на каждую цифру второго числа, начиная с последней, и сложить результаты, учитывая, что результат умножения на очередную цифру надо сдвигать влево на соответствующее количество позиций. Для этого берётся первый сомножитель, помещается в промежуточную переменную, и каждая цифра первого сомножителя умножается на текущую цифру второго сомножителя. После умножения очередной цифры надо вычислить собственно новую цифру как остаток от деления на 10 и число для переноса в следующий разряд как результат от деления на 10. Для ускорения работы алгоритма желательно отдельно рассмотреть случаи умножения числа на цифру 0 и цифру 1. После умножения выполняем сдвиг на нужное количество позиций и прибавляем результат в общую сумму. Для возведения в степень нужно умножить число *N* на себя *M* раз.

```
алг Сдвиг(арг рез большое_число n, арг цел p)
  цел i

  для i от 1000 * 1000 до p + 1 шаг -1
  нц
    n[i] = n[i - p]
  кц
  для i от p до 1 шаг -1
  нц
    n[i] = 0
  кц
кон

алг Сумма(арг большое_число n1, n2)
  большое_число res
  цел i, c                                    // c – переменная для переноса в следующий разряд

  c = 0
  для i от 1 до  1000 * 1000                  // Просматриваем все элементы массива, т.к., во-первых,
  нц                                          // мы не знаем, сколько цифр содержат числа, а, во-вторых,
    res[i] = n1[i] + n2[i] + c                // старшие разряды результата в любом случае надо обнулить
    если res[i] > 9 то
      c = 1
      res[i] = res[i] – 10
    иначе
      c = 0
    всё
  кц
  вернуть res
кон

алг Произведение(арг большое_число n1, n2)
  большое_число res, m
  цел i, j, c                                 // c – переменная для переноса в следующий разряд

  res = 0                                     // Будем считать, что такая операция тоже уже реализована
  для i от 1 до  1000 * 1000
```

```
  нц
    если n2[i] = 1 то
      m = n1
    иначе
      если 2 <= n2[i] и n2[i] <= 9 то
        c = 0
        для j от 1 до  1000 * 1000
        нц
          m[j] = n1[j] * n2[i]
          c = m[j] div 10
          m[j] = m[j] mod 10
        кц
      всё
    всё
    Сдвиг(m, i – 1)
    res = res + m
  кц
  вернуть res
кон

алг МеньшеИлиРавно(арг большое_число n1, n2)
  цел i

  для i от 1000 * 1000 до 1 шаг -1
  нц
    если n1[i] < n2[i] то
      вернуть true
    иначе
      если n1[i] > n2[i] то
        вернуть false
      всё
    всё
  кц
  вернуть true
кон

алг Степень()
  большое_число n, m, i, s, res

  ввод n, m

  res = 1
  i = 1
  s = 1
  пока МеньшеИлиРавно(i, m)
  нц
    res = Произведение(res, n)
    i = Сумма(i, s)
  кц

  вывод res
кон
```

Задание 5.

На листе блокнота школьник Матвей записал несколько строчек из чисел. Размер страницы $M \cdot N$ клеток. Лист вырвал из блокнота и забрал его одноклассник Николай. Хобби Николая – упорядочение чисел. Разработайте алгоритм, который упорядочит все числа на листе в возрастающем порядке.

Алгоритмы сортировки обычно рассчитаны на одномерные массивы. Можно, конечно, попробовать модифицировать какой-нибудь алгоритм для обработки матрицы, но придётся переходить от последнего элемента одной строки к первому элементу следующей строки, но это потребует много проверок и дополнительных действий. Поэтому проще переписать матрицу в одномерный массив, упорядочить его, а затем переписать новый массив обратно в матрицу. Рассмотрим возможные методы сортировки.

Сортировка обменами. Одним из самых простых и широко известных методов сортировки является метод сортировки обменами, называемый также «методом пузырька». Однако скорость работы этого метода оставляет желать лучшего. Этот метод, однако, чувствителен к отсортированным данным, поэтому алгоритм имеет смысл использовать в тех случаях, когда предполагается сортировка практически упорядоченных данных. Метод называют пузырьковой сортировкой, потому что на каждом шаге наибольший элемент неотсортированной части подобно пузырьку газа в воде «всплывает» вверх.

Суть метода сортировки обменами состоит в следующем. На первом шаге сравниваются пары соседних элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-1]$ и $x[n]$. Если $x[i] > x[i+1]$ (для сортировки по возрастанию), то элементы $x[i]$ и $x[i+1]$ меняются местами. После просмотра всего массива максимальный элемент оказывается на последнем месте. На втором шаге сравниваются пары элементов $x[1]$ и $x[2]$, $x[2]$ и $x[3]$, …, $x[n-2]$ и $x[n-1]$, т.е. все элементы, кроме последнего, который уже оказался на своём месте после выполнения первого шага. На последнем шаге сравниваются элементы $x[1]$ и $x[2]$.

Сортировка выбором. Суть метода сортировки выбором состоит в следующем. На первом шаге в массиве находится минимальный (для сортировки по возрастанию) или максимальный (для сортировки по убыванию) элемент. Этот элемент меняется местами с первым элементом массива. На втором шаге мы рассматриваем часть массива, начинающуюся со второго элемента. Находим минимальный или максимальный элемент этой части и меняем его местами со вторым элементом. Процесс повторяет $n-1$ раз – на последнем шаге рассматривается часть массива, состоящая из двух последних элементов.

Сортировка включениями. Одним из наиболее простых и естественных методов внутренней сортировки является сортировка простым включением. Идея алгоритма очень проста. Для каждого элемента массива, начиная со второго, производится сравнение с элементами с меньшим индексом (значение элемента $x[i]$ последовательно сравнивается с элементами $x[i-1]$, $x[i-2]$ ...), и до тех пор, пока для очередного элемента $x[j]$ выполняется соотношение $x[j] > x[i]$ (для сортировки по возрастанию), элемент $x[j]$ сдвигает на одну позицию вправо. Если удаётся встретить такой элемент $x[j]$, что $x[j] \le x[i]$, или если достигнута нижняя граница массива, в $(j+1)$-ую позицию записывается значение элемента $x[i]$ и производится переход к обработке элемента $x[i+1]$. Поскольку при сдвиге элементов элемент $x[i]$ будет перезаписан, предварительно необходимо запомнить значение этого элемента. Этот метод называется также «карточной сортировкой», поскольку часто используется игроками для сортировки выпавших карт.

Сортировка разделением. Этот метод называется также *сортировкой Хоара* по имени разработчика или *быстрой сортировкой*, поскольку он действительно оказывается одним из самых быстрых методов сортировки. Основная идея алгоритма состоит в том, что случайным образом выбирается некоторый элемент массива y, после чего массив просматривается слева, пока не встретится элемент $x[i]$ такой, что $x[i] \ge y$, а затем массив просматривается справа, пока не встретится элемент $x[j]$ такой, что $x[j] \le y$. Эти два элемента меняются местами, и процесс просмотра, сравнения и обмена продолжается, пока i не окажется больше j. В результате массив окажется разбитым на две части – левую, в которой значения элементов будут меньше или равны y, и правую, в которой значения элементов будут больше или равны y. Далее процесс рекурсивно продолжается для левой и правой частей массива до тех пор, пока каждая часть не будет содержать один элемент – массив из одного элемента по определению считается упорядоченным. Можно также отдельно рассмотреть случай для массива из двух элементов. Алгоритм действительно работает очень быстро – время его работы оказывается на 2-3 порядка (!) меньше, чем время работы предыдущих рассмотренных алгоритмов.

```
алг СортировкаМатрицы()
  цел m, n, i, j, k
  вещ matrix[100, 100], mas[10000]

  ввод m, n
  для i от 1 до m
```

```
  нц
    для j от 1 до n
    нц
      ввод matrix[i, j]
    кц
  кц

  k = 0
  для i от 1 до m
  нц
    для j от 1 до n
    нц
      k = k + 1
      mas[k] = matrix[i, j]
    кц
  кц

  QuickSort(mas, 1, k)

  k = 0
  для i от 1 до m
  нц
    для j от 1 до n
    нц
      k = k + 1
      matrix[i, j] = mas[k]
    кц
  кц

  для i от 1 до m
  нц
    для j от 1 до n
    нц
      вывод matrix[i, j]
    кц
  кц
кон

алг QuickSort(арг рез вещ x[10000], арг цел n1, n2)
нач
  цел i, j,
  вещ y, k

  если n2 - n1 = 1 то
    если x[n1] > x[n2] то
      y = x[n1]
      x[n1] = x[n2]
      x[n2] = y
    всё
  иначе
    если n2 - n1 > 1 то
      k = x[(n1 + n2) div 2]
      i = n1
      j = n2
      повторять
        пока (x[i] < k)
        нц
          i = i + 1
        кц
        пока (x[j] > k)
        нц
          j = j - 1
        кц
        если i <= j то
          y = x[i]
          x[i] = x[j]
          x[j] = y
          i = i + 1
          j = j - 1
        всё
      до i > j
      QuickSort(x, n1, j)
      QuickSort(x, i, n2)
    всё
```

```
  всё
кон
```

Задание 6 (Схема решения).

В лесничестве провели эксперимент по выращиванию пихт разных видов. Виды отличаются друг от друга по числу ярусов веток (от N до M). К сожалению, печатная версия плана рассадки сгорела в пожаре, а электронной не было. Перед Вами стоит задача посчитать количество пихт каждого вида в настоящий момент. Участок эксперимента изначально имел прямоугольную форму ($L \cdot K$ км) и было высажено Q саженцев, не все из которых есть сейчас. Каждая пихта занимает квадрат не более 3 м^2 площади. Разработайте алгоритм и схему хранения данных.

При решении данной задачи можно использовать различные способы хранения данных. Поскольку участок имеет прямоугольную форму, первое, что приходит в голову – использовать матрицу. Однако поскольку площадь, занимаемая каждой пихтой может быть разной, пихты могут быть рассажены не рядами, и в этом случае форма рассадки не будет совпадать с матричной. Поэтому можно использовать массив (в задаче сказано, что было высажено Q саженцев) или список. **Список** – это динамически создаваемая структура из однотипных элементов, в которой каждый элемент хранит адрес следующего, а иногда также и предыдущего, элемента. Элементы списка, в отличие от элементов массива, не должны располагаться в памяти последовательно единым целым.

Достоинства списков:

- размер списка ограничивается только доступным объёмом машинной памяти (массивы имеют ограничения на максимальный размер);
- при изменении последовательности элементов списка требуется не перемещение данных в памяти, а только коррекция адресов.

Недостатки списков:

- на адреса расходуется дополнительная память;
- доступ к элементам связной структуры может быть менее эффективным по времени, поскольку невозможно обратиться непосредственно к i-ому элементу списка – для того чтобы добраться до нужного элемента, надо пройти до него от начала списка.

Таким образом, пройдя по участку, мы составим массив или список с данными: в каждом элементе будет храниться количество ярусов веток у данного саженца. В список элементы добавляются в процессе работы по одному. Потом проходим по массиву или списку и строим массив, в котором каждому возможному количеству ярусов веток будет соответствовать количество саженцев с данным количеством ярусов веток. Если использовался список, при завершении работы его надо удалить.

В приведенном ниже алгоритме не рассматриваются ограничения на исходные данные (L*K*10^6 ≤ 3*Q).

Рассмотрим вариант со списком. Количество пихт каждого вида будем хранить в массиве, при этом будем использовать количество ярусов веток как индекс массива. Если N окажется больше 1, то часть элементов массива использоваться не будет, но такой подход удобнее и проще. Будем также считать, что количество ярусов веток не превышает 10.

```
алг Лесничество()
  цел n, m, levels[10], i

  ввод n, m

  создать список
```

```
  для всех пихт
  нц
    добавить в список новый элемент и сделать его текущим
    записать в текущий элемент количество ярусов очередной пихты
  кц

  для i от n до m
  нц
    levels[i] = 0
  кц

  установить текущий элемент на начало списка
  пока список не закончился
  нц
    для i от n до m
    нц
      если текущее значение в списке = i то
        levels[i] = levels[i] + 1
      всё
    кц
    перейти к следующему элементу списка
  кц

  для i от n до m
  нц
    вывод i, levels[i]
  кц

  удалить список
кон
```

Решение варианта 7092

Задание 1.

Рассмотрим систему счисления, в которой основанием с.с. являются факториалы. N-разрядное число является суммой факториалов N первых натуральных чисел. Пример. 4321_f = 4*4! + 3*3! + 2*2! + 1*1! = 119_{10}. Является ли она позиционной? Ответ аргументировать.

Для ответа на вопрос следует вспомнить определение позиционной с.с.

В позиционных системах счисления один и тот же числовой знак (цифра) в записи числа имеет различные значения в зависимости от того места (разряда), где он расположен. Таким образом, данную с.с. можно считать позиционной.

Задание 2. (схема решения)

Как Вы знаете, этой осенью в нашей стране опять произошел перевод часов. Чтобы избежать ошибок при смене часовых поясов, все часы в компьютере могут работать на GMT (универсальном времени по Гринвичу). Часовых поясов много (от GMT-12 до GMT+14), стран с переводом часов тоже много. Какие-то страны вообще не переводят время. Ставится задача: определить местное время в заданной стране в заданном часовом поясе. Разработайте схему хранения данных, с использованием которой можно решить поставленную задачу наиболее оптимально.

Решение задачи весьма актуально в современной жизни. Общепринятым в настоящее время является следующий способ: существует таблица данных по часовым поясам, «привязанная» к крупным городам (странам) по всему миру. Также для каждого часового пояса есть отдельные данные о дате перевода на летнее/зимнее время. Поскольку в разные года перевод происходит по-разному (пример – наша страна), то также храним информацию о годе, с которого наступает текущий «режим перехода. Также учитываем, что информация время от времени меняется, и её надо обновлять. Для этого в современных операционных системах реализован механизм TimeZones.

Задание 3. (схема решения)

В криптографической системе с открытым ключом RSA используется функция Эйлера $\omega(p)$, значением которой является количество остатков от деления на p, взаимно простых с p. Взаимно простыми являются целые числа, наибольший общий делитель которых равен 1. Разработайте алгоритм для вычисления значения функции Эйлера от задаваемого целого числа.

Алгоритм простой, но не оптимальный. Проверим все числа от 1 до p – 1. Для каждого числа надо найти наибольший общий делитель этого числа и числа p. (Для нахождения наибольшего делителя двух чисел можно использовать алгоритм Евклида, который заключается в вычитании меньшего числа из большего до тех пор, пока числа не станут равными – это число и будет наибольшим общим делителем.) Если наибольший общий делитель равен 1, значит, мы нашли число, взаимно простое с p, и надо прибавить 1 к общему результату.

Алгоритм оптимальный. Для простого числа p функция Эйлера $\omega(p) = p - 1$. Для непростого числа p функция Эйлера $\omega(p) = p \cdot \prod_{i|p}\left(1-\frac{1}{i}\right)$, где i – простое число и пробегает все значения, участвующие в разложении числа p на простые сомножители. Поэтому можно использовать следующий алгоритм. В первую очередь проверим, не является ли заданное число p простым. (Для этого необходимо проверить, не делится ли заданное число p на какое-либо число от 2 до

$\sqrt{p}$.) Если число действительно простое, то результат будет равен $p - 1$. Если же число p не является простым, построим массив простых чисел от 2 до $p / 2$ (при проверке на простоту достаточно взять значения до $\sqrt{p}$, но в данном случае нам понадобятся все возможные простые сомножители, а поскольку наименьший возможный сомножитель равен 2, то наибольший возможный сомножитель будет равен $p / 2$). Далее положим результат равным p и переберём все элементы из массива простых чисел. Если число p делится на какой-либо элемент массива, необходимо умножить результат на выражение $\left(1-\frac{1}{x[i]}\right)$, где $x[i]$ – текущий элемент массива простых чисел. В данном случае каждый сомножитель рассматривается только один раз, даже если он несколько раз входит в разложение числа p на простые сомножители. Например, для числа $25 = 5 \cdot 5$ функция Эйлера $\omega(25) = 25 \cdot \left(1-\frac{1}{5}\right) = 20$.

Задание 4. (схема решения)

Даны целые неотрицательные числа M и N, количество разрядов которых может быть велико. Разработайте алгоритм для нахождения величины N^M. Ограничение: длина чисел $0 \le N, M \le 1000$ цифр.

Столь длинные числа не могут быть представлены в современных компьютерах, тем более, что для результата понадобится уже 1000 * 1000 цифр. Поэтому для решения задачи необходимо разработать способ хранения чисел нужной длины и алгоритм умножения чисел в новом представлении. Для представления чисел можно использовать массив, каждый элемент которого хранит одну цифру числа. Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первый сомножитель надо умножить на каждую цифру второго числа, начиная с последней, и сложить результаты, учитывая, что результат умножения на очередную цифру надо сдвигать влево на соответствующее количество позиций. Для этого берётся первый сомножитель, помещается в промежуточную переменную, и каждая цифра первого сомножителя умножается на текущую цифру второго сомножителя. После умножения очередной цифры надо вычислить собственно новую цифру как остаток от деления на 10 и число для переноса в следующий разряд как результат от деления на 10. Для ускорения работы алгоритма желательно отдельно рассмотреть случаи умножения числа на цифру 0 и цифру 1. После умножения выполняем сдвиг на нужное количество позиций и прибавляем результат в общую сумму. Для возведения в степень нужно умножить число N на себя M раз.

Задание 5. (схема решения)

В одном доме с Игорем живёт пенсионер Сергей Петрович, который в силу возраста страдает склерозом. В частности, он забывает почтовый индекс своего дома. Однако при этом он помнит, что он не менее чем пятизначный и содержит не более 8 цифр, он делится на 7 и 11. Разработайте алгоритм, который находит возможные варианты индекса.

Для решения задачи необходимо перебрать все числа от 10000 (наименьшее пятизначное число) до 99999999 (наибольшее восьмизначное число) и проверить, делится ли текущее число на 7 и на 11. При этом можно использовать признаки делимости чисел. Число делится на 7 тогда и только тогда, когда утроенное число десятков, сложенное с числом единиц, делится на 7. Т.е. число надо поделить на 10, результат умножить на 3 и прибавить остаток от деления на 10 исходного числа. Поскольку полученное число может оказаться достаточно большим, эти действия нужно повторить несколько раз до тех пор, пока результат не станет меньше 10. Если он при этом равен 7, то исходное число делится на 7. Например, возьмём число 1001. 100 * 3 + 1 = 301, 30 * 3 + 1 = 91, 9 * 3 + 1 = 28, 2 * 3 + 8 = 14, 1 * 3 + 4 = 7. Значит, 1001 делится на 7. Число делится на 11 тогда и только тогда, когда модуль разности между суммой цифр, занимающих нечётные позиции, и суммой цифр, занимающих чётные места, делится на 11. Например, 9163627 делится на 11, т.к. |(9 + 6 + 6 + 7) – (1 + 3 + 2)| = 22, а 22 делится на 11, т.к. |2 – 2| = 0.

Задание 6.

Машинным эпсилон ε_M называется наименьшее положительное число ε такое, что при расчёте на компьютере $1 + \varepsilon_M \neq 1$. Объясните, как возможно при сложении числа и единицы получить в результате 1 и предложите алгоритм вычисления ε_M.

Вещественное число в компьютере представляется с помощью знака, мантиссы и порядка, например, $-3.507 \cdot 10^{-3}$, причём мантисса всегда нормализована, т.е. содержит только одну значащую цифру до десятичной точки. Кроме того, мантисса ограничена по размеру. Положим, к примеру, что мантисса может содержать 4 цифры (конечно, их больше, но собственно количество цифр не влияет на наши рассуждения). Тогда мы можем представить в компьютере число $1 \cdot 10^{-4}$ и сложить его с числом $1 \cdot 10^{-5}$, получив в результате $1.1 \cdot 10^{-4}$. Но если мы попробуем сложить число $1 \cdot 10^{-4}$ с единицей ($1 \cdot 10^{0}$), то, по идее, мы должны получить число $1.0001 \cdot 10^{0}$, но поскольку мантисса ограничена четырьмя цифрами, пятая цифра отбрасывается, и в результате сложения мы получаем единицу. Вспомним теперь, что в компьютере числа представляются в двоичной системе счисления. Для нахождения числа ε_M необходимо единицу делить на 2 до тех пор, пока результат сложения получаемого числа с 1 не будет равен 1.

Задание 7. (схема решения)

Разработайте алгоритм, который определяет (в порядке убывания) номера разрядов, содержащих цифру 4 в десятичной записи числа 44!.

Число 44! слишком большое для того, чтобы его можно было точно представить в современном компьютере. Поэтому для решения задачи необходимо разработать способ хранения чисел нужной длины и алгоритм умножения чисел в новом представлении. Для представления чисел можно использовать массив, каждый элемент которого хранит одну цифру числа. После нахождения значения 44! можно будет просмотреть этот массив и найти номера разрядов, содержащих цифру 4.

Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первый сомножитель надо умножить на каждую цифру второго числа, начиная с последней, и сложить результаты, учитывая, что результат умножения на очередную цифру надо сдвигать влево на соответствующее количество позиций. Для этого берётся первый сомножитель, помещается в промежуточную переменную, и каждая цифра первого сомножителя умножается на текущую цифру второго сомножителя. После умножения очередной цифры надо вычислить собственно новую цифру как остаток от деления на 10 и число для переноса в следующий разряд как результат от деления на 10. Для ускорения работы алгоритма желательно отдельно рассмотреть случаи умножения числа на цифру 0 и цифру 1. После умножения выполняем сдвиг на нужное количество позиций и прибавляем результат в общую сумму.

Решение варианта 7093

Задание 1.

Рассмотрим систему счисления, в которой основанием с.с. являются числа Фибоначчи.

Алфавитом этой системы счисления являются цифры 0 и 1. В записи числа в фибоначчиевой системе не могут стоять две единицы подряд. Пример. Покажем, как записывать числа в фибоначчиевой системе счисления:

37_{10} = 34 + 3 = 1*34+0*21+0*13+0*8+0*5+1*3+0*2+0*1= 10000100Fib;

25_{10} = 21 + 3 + 1 = 1*21+0*13+0*8+0*5+1*3+0*2+1*1=1000101Fib.

Является ли с.с. позиционной? Ответ аргументировать.

Для ответа на вопрос следует вспомнить определение позиционной с.с.

В позиционных системах счисления один и тот же числовой знак (цифра) в записи числа имеет различные значения в зависимости от того места (разряда), где он расположен. Таким образом, указанная с.с. относится к позиционным.

Задание 2.

Как Вы знаете, для представления чисел в ЭВМ используют двоичную систему счисления.

В чем ее преимущества и недостатки? Почему не пользуются «классической» десятичной с.с.?

Есть ли еще какие-нибудь с.с., которые целесообразно использовать при работе на компьютере? Если есть – то зачем их использовать?

Недостатки – неудобство для обычного пользователя. Не пользуются 10-й с.с, поскольку она не приспособлена к существующим логическим схемам, которые основаны на булевой алгебре. Ещё используют 8-ую с.с, 16-ю с.с, 2-10 с.с. Смешанную с.с. 2-10 для выполнения операций умножения. Восьмеричная с.с., например, используется для указания прав доступа в операционных системах семейства UNIX. Шестнадцатеричные с.с. используются обычно при программировании на низкоуровневом языке Ассемблера.

Задание 3.

Для проверки, является ли большое целое простым, может использоваться вероятностный тест Миллера-Рабина. Пусть $p > 2$ – простое число. Представим число $p - 1$ в виде $p - 1 = 2^s \cdot d$, где d – нечётное число. Тогда для любого a из $\{1, 2, \ldots, p - 1\}$ выполняется одно из условий:

1. $a^d = 1 \{\mathrm{mod}\ p\}$

2. Существует r, $0 \le r \le s - 1$, для которого $a^{k \cdot d} = -1 \{\mathrm{mod}\ p\}$, где $k = 2^r$.

В тесте Миллера-Рабина эти проверки выполняются для t случайно выбираемых a (t –исходные данные). Разработайте алгоритм проверки вводимого числа на простоту по тесту Миллера-Рабина.

Прежде всего, определим числа s и d. Для этого необходимо число $p - 1$ поделить на 2 несколько раз, пока возможно деление без остатка. Тогда количество возможных делений даст число s, а полученный результат – число d. После это считаем число t и t раз выбираем число a из множества возможных значений. Для каждого значения a считаем значение $a^d \bmod p$. Для того

чтобы не возводить большое число в большую степень, рискуя превысить максимальное число, представимое в компьютере, можно использовать следующее свойство: $(x \cdot y) \bmod n = ((x \bmod n) \cdot (y \bmod n)) \bmod n$. Если $x = a^d \bmod p$ равно 1, то можно перейти к следующему значению a. Если же $x \neq 1$, то необходимо проверить второе условие. Для этого $s - 1$ раз вычисляем $x = x^2 \bmod p$. Если ни на одном шаге x не равно $p - 1$, то возвращаем «составное». В противном случае можно продолжить проверку. Если для всех a будет выполнено хотя бы одно из двух условий, возвращаем «вероятно, простое».

```
алг ТестМиллераРабина()
нач
  цел p, s, d, a, t, r, i, j, ap, x
  лог prime

  ввод p,t

  d = p - 1
  s = 0
  пока d mod 2 = 0
  нц
    d = d div 2
    s = s + 1
  кц

  prime = true
  i = 1
  пока i <= t и prime
  нц
    a = rand(p – 1)             // Случайным образом выбираем число от 1 до p – 1
    ap = a mod p
    x = 1
    для j от 1 до d
    нц
      x = (x * ap) mod p
    кц

    если x <> 1 то
      prime = false
      j = 0
      пока j <= s – 1 и не prime
      нц
        если x mod p = p – 1 то
          prime = true
        всё
        x = (x * x) mod p
        j = j + 1
      кц
    всё
    i = i + 1
  кц

  если prime то
    вывод "Вероятно, простое"
  иначе
    вывод "Составное"
  всё

кон
```

Задание 4.

Найти все числа N из диапазона $N1 \leq N \leq N1 + 50$, которые представляются суммой четырёх квадратов натуральных чисел не единственным образом и которые имеют на заданном диапазоне наименьшее количество таких представлений.

Для каждого числа N из указанного диапазона необходимо проверить возможность представления в виде суммы четырёх квадратов натуральных чисел и найти количество таких представлений, а также найти количество представлений, возможных, когда натуральные числа

попадают в заданный диапазон. Поэтому для каждого числа *N* будем формировать два массива. Для этого внутри цикла, перебирающего возможные значения *N*, нужны ещё четыре вложенных цикла. В первом счётчик цикла *i* меняется от 1 до *N*, во втором счётчик цикла *j* меняется от *i* до *N*, в третьем счётчик цикла меняется от *j* до *N*, в четвёртом счётчик цикла *m* меняется от *k* до *N*. Таким образом, мы переберём все возможные комбинации натуральных чисел, при этом в комбинации можно использовать одинаковые числа, но такие комбинации как (1, 2) и (2, 1) будут считаться одинаковыми и обе использоваться не будут. Для каждой комбинации чисел проверяем, возможно ли представление нужного числа, и попадают ли исходные числа в заданный диапазон. После построения массивов положим переменную *min* равной $(N1 + 50)^4$ – значение, которое заведомо больше, чем возможное количество представлений какого-либо числа в виде суммы четырёх квадратов натуральных чисел, – и будем просматривать полученные массивы. Если текущий элемент первого массива больше 1, а текущий элемент второго массива меньше значения переменной *min*, но при этом больше 0, то переменной *min* надо присвоить новое значение – значение текущего элемента второго массива. После нахождения минимума надо снова пройти по массивам и найти те числа, для которых значение в первом массиве больше 1, а значение во втором массиве равно найденному минимуму.

```
алг СуммаКвадратов()
нач
  цел N1, A, B n, i, j, k, m, min
  цел c1[0..50], c2[0..50]

  ввод N1, A, B

  для i от 0 до 50
  нц
    c1[i] = 0
    c2[i] = 0
  кц

  для n от N1 до N1 + 50
  нц
    для i от 1 до N
    нц
      для j от i до N
      нц
        для k от j до N
        нц
          для m от k до N
          нц
            если i * i + j * j + k * k + m * m = n то
              c1[n – N1] = c1[n – N1] + 1
              если A <= i и i <= B и A <= j и j <= B и A <= k и k <= B и A <= m и m <= B то
                c2[n – N1] = c2[n – N1] + 1
              всё
            всё
          кц
        кц
      кц
    кц
  кц

  min = (N1 + 50) * (N1 + 50) * (N1 + 50) * (N1 + 50)
  для n от N1 до N1 + 50
  нц
    если c1[n – N1] > 1 и c2[n – N1] < min и c2[n – N1] > 0 то
      min = c2[n – N1]
    всё
  кц

  для n от N1 до N1 + 50
  нц
    если c2[n – N1] = min то
      вывод n
    всё
  кц
кон
```

Задание 5.

В стране Котории король Самодур посадил в тюрьму графа Умноедова. Тюрьма была построена при короле Знаетеле, и из каждой камеры можно было выйти, если собрать головоломку на двери. Разработав алгоритм, помогите графу выйти из камеры! (цифры заменены звёздочками).

```
    * 1 *
  x
    3 * 2
------------
    * 3 *
  3 * 2 *
* 2 * 5
------------
1 * 8 * 3 *
```

Три неизвестные цифры в перемножаемых числах однозначно определяют все остальные. Поэтому достаточно определить только их. На компьютере это можно сделать методом подбора – пишем три вложенных цикла, каждый из которых перебирает десять возможных цифр. Зная цифры чисел, мы можем определить сами числа, а также промежуточные и окончательный результаты. Затем с помощью операций «деление» и «остаток от деления» можно определить, какие цифры стоят на той или иной позиции, и сравнить их с известными. Впрочем, даже если у графа нет компьютера, то за него можно не беспокоиться, поскольку задача решается логическим путём даже быстрее, чем на компьютере.

Приведённый ниже алгоритм предназначен для решения именно этой головоломки, однако он обобщается для решения других головоломок такого же типа.

```
алг Головоломка()
нач
  цел d1, d2, d3, n1, n2, r, r1, r2, r3

  для d1 от 0 до 9
    для d2 от 0 до 9
      для d3 от 0 до 9
        n1 = d1 * 100 + 10 + d2
        n2 = 300 + d3 * 10 + 2
        r1 = n1 * 2
        r2 = n1 * d3
        r3 = n1 * 3
        r = n1 * n2
        если (r1 div 10) mod 10 = 3 и (r2 div 10) mod 10 = 2 и r2 div 1000 = 3 и
              r3 mod 10 = 5 и (r3 div 100) mod 10 = 2 и (r div 10) mod 10 = 3 и
             (r div 1000) mod 10 = 8 и r div 100000 = 1 то
          вывод n1, n2, r1, r2, r3, r
        всё
      кц
    кц
  кц
кон
```

Задание 6.

Разработайте алгоритм для перевода целого числа *kx* из системы счисления с основанием *q* ($2 \le q \le 16$) в десятичную систему счисления. Входные и выходные данные являются строкой текста.

Для решения данной задачи переведём исходную строку в число во внутреннем компьютерном представлении, а затем полученное число – в строку, содержащую число в другой системе счисления. Прежде всего, проверим, не является ли первый символ строки знаком «плюс» или «минус». Знак «плюс» можно пропустить или при желании повторить в результирующей строке, а наличие знака «минус» обязательно надо запомнить. Результат для начала положим равным 0. Далее просматриваем все элементы строки. Каждый элемент прежде всего проверяем на корректность и, если это так, переводим в соответствующую цифру. Для этого для цифр надо из кода текущего символа вычесть код символа «0», а для букв, которые используются в качестве цифр в системах счисления с основанием больше 10, из кода текущего символа вычесть код символа «а» и прибавить 10. Далее результат умножаем на основание системы счисления q и прибавляем к результату цифру, полученную из текущего символа. После просмотра всей исходной строки надо сформировать новую строку из полученного числа. Для этого в цикле получаем очередную цифру, взяв остаток от деления на 10, и делим число на 10. Из цифры получаем код очередного символа, прибавив к цифре код символа «0». Цикл завершается, когда число станет равным 0. После этого при необходимости добавляем в строку символ «плюс» или «минус». Символы выходной строки образуются в обратном порядке, поэтому строку надо инвертировать.

```
алг Перевод()
нач
  строка inum, onum
  цел q, n, i, k, d, sign, f
  символ c

  ввод inum, q

  если inum[1] = '+' то
    sign = 1
    f = 2
  иначе
    если inum[1] = '-' то
      sign = -1
      f = 2
    иначе
      sign = 0
      f = 1
    всё
  всё

  n = 0
  для i от f до length(inum)
  нц
    если '0' <= inum[i] и inum[i] <= '9' то
      d = ord(inum[i]) – ord('0')
    иначе
      если 'a' <= inum[i] и inum[i] <= 'f' то
        d = ord(inum[i]) – ord('a') + 10
      иначе
        если 'A' <= inum[i] и inum[i] <= 'F' то
          d = ord(inum[i]) – ord('A') + 10
        иначе
          вывод "Некорректные исходные данные"
          прервать выполнение алгоритма
        всё
      всё
    всё
    если d > q – 1 то
      вывод "Некорректные исходные данные"
      прервать выполнение алгоритма
    всё
    n = n * q + d
  кц

  k = 0
  пока n > 0
  нц
    d = n mod 10
    n = n div 10
```

```
    k = k + 1
    onum[k] = chr(d + ord('0'))
  кц

  если sign = 1 то
    k = k + 1
    onum[k] = '+'
  иначе
    если sign = -1 то
      k = k + 1
      onum[k] = '-'
    всё
  всё

  для i от 1 до k div 2
  нц
    c = onum[i]
    onum[i] = onum[k – i + 1]
    onum[k – i + 1] = c
  кц

  вывод onum

кон
```

Задание 7.

Школьник Валера увлекается арифметикой. Ему попалась задача: найти все четвёрки простых чисел до 500000, принадлежащие одному десятку. Разработайте наиболее эффективный алгоритм, который решает поставленную задачу.

Для начала построим массив простых чисел. Запишем в этот массив числа 2, 3, 5 и 7. Положим $n = 11$. Переменная n будет пробегать нечётные значения до 500000, причём на каждом шаге цикла будем проверять два значения n, один раз прибавив к n число 2, а затем – число 4. Таким образом для ускорения поиска мы выкинем из рассмотрения не только чётные числа, но и нечётные числа, делящиеся на 3. Для проверки очередного значения n на простоту можно использовать уже найденные простые числа, сохранённые в массиве, от 2 до $\sqrt{n}$. При формировании массива надо подсчитать значение k – количество элементов массива простых чисел. После этого в цикле для i от 1 до $k - 3$ проверяем разность между i-ым и $(i + 3)$-им элементами массива. Если эта разность меньше 10, выводим четыре числа, начиная с i-го.

```
алг ПростыеЧисла()
нач
  цел nums[250000], n, k, i
  лог f

  nums[1] = 2
  nums[2] = 3
  nums[3] = 5
  nums[4] = 7
  k = 4

  n = 11
  пока n <= 500000
  нц
    f = true
    i = 1
    пока nums[i] <= целая_часть(sqrt(n)) и f
    нц
      если n mod nums[i] = 0 то
        f = false
      всё
      i = i + 1
    кц
    если f то
      k = k + 1
      nums[k] = n
    всё
    n = n + 2
    f = true
```

```
    i = 1
    пока nums[i] <= целая_часть(sqrt(n)) и f
    нц
      если n mod nums[i] = 0 то
        f = false
      всё
      i = i + 1
    кц
    если f то
      k = k + 1
      nums[k] = n
    всё
    n = n + 4
  кц

  для i от 1 до k - 3
  нц
    если nums[i + 3] - nums[i] < 10 то
      вывод nums[i], nums[i + 1], nums[i + 2], nums[i + 3]
    всё
  кц
```

КОН