Материалы заданий Олимпиады школьников «Надежда энергетики» по предмету «информатика» в 2015/2016 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 Олимпиады школьников
«Надежда энергетики» по предмету «информатика»
в 2015/2016 учебном году

Решение варианта 7991 для 9 классов

Задание 1

Древние Майя использовали 20-ичную систему счисления за одним исключением: во втором разряде было не 20, а 18 ступеней, то есть за числом (17)(19) сразу следовало число (1)(0)(0). Это было сделано для облегчения расчётов календарного цикла, поскольку (1)(0)(0) = 360 – примерно равно числу дней в солнечном году. Разработайте алгоритм перевода натуральных чисел из десятичной с.с. в систему Майя.

Решение. Для хранения чисел будем использовать массив, каждый элемент которого будет хранить одну цифру числа в системе счисления Майя. Для перевода будем использовать обычный алгоритм – делим число в десятичной системе счисления на 20 и записываем полученные цифры. После получения всех цифр надо проверить цифру второго разряда – если она больше или равна 18, то вычитаем 18 из этой цифры и осуществляем перенос 1 в следующий разряд. Но поскольку в третьем разряде может оказаться цифра (19), то перенос может перейти в четвёртый разряд и т.д.

```
число = запись
  цел digits[100]                              // Массив цифр
  цел count                                    // Количество цифр в числе
кон

алг ПереводЧислаВСистемуМайя()
нач
  цел n, tr, i
  число m

  ввод n

  count = 0
  выполнить
    m.digits[count] = n mod 20
    count = count + 1
    n = n div 20
  до n = 0 или count > 100

  если count > 100 то
    вывод 'Слишком большое число'
  иначе
    если m.digits[2] >= 18 то
      m.digits[2] = m.digits[2] – 18
      tr = 1
      i = 3
      пока tr = 1 и i <= 100
      нц
        m.digits[i] = m.digits[i] + tr
        если m.digits[i] >= 20 то
          m.digits[i] = m.digits[i] - 20
        иначе
          tr = 0
        всё
        i = i + 1
      кц
    всё
    если i > 100 то
      вывод 'Слишком большое число'
    иначе
      если i > count то
        count = i
      всё
      для i от count + 1 до 100
      нц
        m.digits[i] = 0
      кц
      для i от count до 1 шаг -1
      нц
        вывод '(', m.digits[i], ')'
      кц
    всё
```

```
  всё
кон
```

Задание 2

На листе бумаги в строке записано N натуральных чисел. Разработайте алгоритм, который переупорядочивает их (в новой строке) так, чтобы начало строки составляли числа, имеющие чётные значения, расположенные в порядке возрастания. Оставшуюся часть строки должны составлять нечётные значения, расположенные в порядке убывания.

Решение. Переставим элементы массива так, чтобы сначала шли чётные элементы, а потом нечётные. Для этого найдём элемент с номером i – первый нечётные элемент и элемент с номером j – последний чётный элемент, и поменяем их местами. Затем найдём номер i следующего нечётного элемента и номер j предыдущего чётного элемента, и снова поменяем их местами. Продолжать эти действия надо до тех пор, пока i не станет больше j. На тот случай, если в массиве нет чётных или нечётных элементов, проверяем, чтобы i не стало больше общего количества элементов массива, а j не стало меньше 1. После переупорядочения i будет хранить номер первого нечётного элемента, а j – номер последнего чётного элемента, и теперь нужно отсортировать обе части массива.

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 $x[i] \geq y$, а затем массив просматривается справа, пока не встретится элемент $x[j]$ такой, что $x[j] \leq y$. 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 Переупорядочение()
нач
  цел n, a[n], i, j, c

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

  i = 1
  j = n
  пока i < j
  нц
    пока i <= n и a[i] mod 2 = 0
    нц
      i = i + 1
    кц
    пока j >= 1 и a[j] mod 2 = 1
    нц
      j = j - 1
    кц
    если i < j
      c    = a[i]
      a[i] = a[j]
      a[j] = c
    всё
  кц

  QuickSort(a, 1, j)
  QuickSort(a, i, n)
кон
```

```
алг QuickSort(арг цел x[1000], арг цел n1, n2)
нач
  цел i, j
  цел 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 3

В качестве ключа для шифрования секретных сведений использовалось число S, являющееся суммой некоторых целых положительных чисел A, B и C ($A < B < C$). Причём $B - A = C - B$. Для дешифровки используется число B. Найти число B, если известно число S.

Единственная строка входных данных содержит целое положительное число не длиннее 100 знаков – число S.

Выходные данные содержат искомое число B, или слово "Ошибка", если не существует чисел A, B, C, удовлетворяющих условию задачи.

Примеры

Исходные данные	**Результат**
111111111	37037037
1000000000	Ошибка
603360336033	201120112011

Решение. $S = A + B + C$. Поскольку $B - A = C - B$, то можно записать следующее уравнение $S = A + (A + x) + (A + 2 \cdot x)$. Отсюда $S = 3 \cdot A + 3 \cdot x$. Следовательно, $B = A + x = S / 3$. Поэтому если S делится на 3, то $B = S / 3$, а если не делится – решения нет.

```
алг Ключ()
нач
  цел s

  ввод s

  если s <= 3 то
    вывод 'Неверные исходные данные'
  иначе
    если s mod 3 <> 0 то
      вывод 'Ошибка'
    иначе
      вывод 'B = ', S div 3
    всё
  всё
кон
```

Задание 4

У прилавка в магазине выстроилась очередь из n покупателей. Время обслуживания продавцом i-го покупателя равно t_i ($i = 1, 2, ..., n$). Пусть даны натуральное n и действительные $t_1, ..., t_n$. Получить $c_1, ..., c_n$, где c_i – время пребывания i-го покупателя в очереди. Указать номер покупателя, для обслуживания которого продавцу потребовалось самое малое время.

Решение. $c_1 = t_1$, $min = t_1$. Далее в цикле от 2 до n вычисляем $c_i = c_{i-1} + t_i$. В этом же цикле вычисляем минимум из элементов t_i по обычному алгоритму.

```
алг Очередь()
нач
  цел n, i
  вещ t[n], c[n], min

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

  c[1] = t[1]
  min  = t[1]
  для i от 2 до n
  нц
    c[i] = c[i – 1] + t[i]
    если t[i] < min то
      min = t[i]
    всё
  кц
кон
```

Задание 5

Разработайте алгоритм, который определяет (в порядке возрастания) номера разрядов, содержащих цифру 6 в десятичной записи числа 64^{513}.

Схема решения. Число 64^{513} 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 64^{513} можно будет просмотреть этот массив и найти номера разрядов, содержащих цифру 2. Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первое число 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 прибавить перенос из предыдущего разряда и 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 сложения чисел будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 10, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 10) и запоминаем наличие переноса в следующий разряд.

Решение варианта 7992 для 9 классов

Задание 1

Как известно, современная система измерения времени ведёт начало от древнего Вавилона, где использовались 60-ричная с.с. Разработайте алгоритм перевода натуральных чисел из десятичной с.с. в 60-ричную с.с. Каждая цифра 60-ричной с.с. записывается в десятичной системе в круглых скобках, например, (21).

Решение. Для хранения чисел будем использовать массив, каждый элемент которого будет хранить одну цифру числа в 60-ричной системе счисления. Для перевода будем использовать обычный алгоритм – делим число в десятичной системе счисления на 60 и записываем полученные цифры.

```
число = запись
  цел digits[100]                                  // Массив цифр
  цел count                                        // Количество цифр в числе
кон

алг ПереводЧислаВ60Систему()
нач
  цел n, tr, i
  число m

  ввод n

  count = 0
  выполнить
    m.digits[count] = n mod 60
    count = count + 1
    n = n div 60
  до n = 0 или count > 100

  если count > 100 то
    вывод 'Слишком большое число'
  иначе
    для i от count + 1 до 100
    нц
      m.digits[i] = 0
    кц
    для i от count до 1 шаг -1
    нц
      вывод '(', m.digits[i], ')'
    кц
  всё
кон
```

Задание 2

Числа Сабита – натуральные числа, задающиеся формулой $3 \cdot 2^n - 1$ для неотрицательных n. Разработайте алгоритм нахождения суммы чисел Сабита в диапазоне от P до Q.

Решение. Перебираем значения n, начиная с 0. Ищем первое число n, для которого выражение $3 \cdot 2^n - 1$ будет больше или равно P. Используем полученное число как начальное значение суммы. Для следующих n, пока значение выражения $3 \cdot 2^n - 1$ будет меньше или равно Q, суммируем полученные числа Сабита.

```
алг ЧислаСабита()
нач
  цел p, q, n, s, sum

  ввод p, q

  n = 0
  s = 3 * pow(2, n) - 1
  пока s < p
  нц
    n = n + 1
```

```
    s = 3 * pow(2, n) – 1
  кц

  sum = s
  n = n + 1
  s = 3 * pow(2, n) – 1
  пока s <= q
  нц
    sum = sum + s
    n = n + 1
    s = 3 * pow(2, n) – 1
  кц
кон
```

Задание 3

В основе алгоритма RSA лежит использование пары простых чисел P и Q и производного числа (модуля) $N = P * Q$. Простое число – это натуральное число, которое имеет ровно два различных натуральных делителя: единицу и самого себя.

Принципиальным отличием нового алгоритма RSA++ от алгоритма RSA состоит в выборе ключей. Если в алгоритме RSA требуется пара простых чисел P и Q, то в алгоритме RSA++ числа P и Q должны быть взаимно простыми, т.е. они имеют только один общий делитель, равный 1.

Для анализа надёжности нового алгоритма необходимо узнать количество различных пар чисел P и Q, таких, что $1 < P < Q$ и соответствующий им модуль удовлетворяет условию $N \le K$.

Первая строка входных данных содержит одно целое число K ($1 \le K \le 109$).

Результат должен содержать одно целое число – количество различных пар чисел P и Q.

Примеры

Входные данные	**Результат**
12	3
18	6

Решение. Число P меняется в диапазоне от 2 до $\sqrt{K}$. Число Q меняется в диапазоне от $P + 1$ до K / P. Перебираем все возможные значения P и соответствующие им значения Q. Если числа являются взаимно простыми, прибавляем единицу к итоговому результату.

Для проверки, что числа являются взаимно простыми, надо найти НОД по алгоритму Евклида. Если НОД равен 1, то числа являются взаимно простыми.

```
алг АнализНадёжности()
нач
  цел k, p, q, n

  ввод k

  если k < 1 или k > 109 то
    вывод 'Неверное значение K'
  иначе
    n = 0
    для p от 2 до целая_часть(sqrt(k))
    нц
      для q от p + 1 до k div p
      нц
        если НОД(p, q) = 1 то
          n = n + 1
        всё
      кц
    кц
    вывод 'Количество пар взаимно простых чисел равно ', n
```

```
  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
```

Задание 4

Службой Внешней разведки был перехвачен фрагмент двоичного сообщения

xxxx0101010100110101010001010101010101000101001001010000010101000101010101010100
01010000010xxxx,

где xxxx – неизвестное количество (т.е. не обязательно 4) неизвестных бит – утерянная часть сообщения.

Достоверно известно, что сообщение кодирует текст, записанный русскими буквами без пробелов и знаков препинания с использованием следующей кодировки в шестнадцатеричной системе исчисления:

А – 50_{16}

Б – 51_{16}

В – 52_{16}

Г – 53_{16}

Д – 54_{16}

Е – 55_{16}

При этом на каждый символ отводится по 8 бит.

Расшифруйте текст доступного дешифровке фрагмента сообщения, учитывая возможность наличия в начале и конце сообщения произвольного (меньшего, чем по 8) количества бит, оставшихся от утерянной части сообщения.

Решение. Поскольку количество потерянных бит неизвестно, возможно, что биты, относящиеся к какой-либо букве, могут начинаться не с начала известной части сообщения. Поэтому следует рассмотреть 8 вариантов. Берём первые 8 бит известной части сообщения, выводим соответствующую букву, затем берём следующие 8 бит и т.д. Если последняя часть будет содержать меньше 8 бит, отбрасываем эту часть. Затем аналогично рассматриваем сообщение, в котором условно отброшен один бит, два бита и т.д. Таким образом, мы получим 8 вариантов расшифровки, и человек сможет определить, какой вариант является осмысленным.

```
алг Расшифровка()
нач
  строка s = '0101010100110101010001010101010101000101001001010000010101000101010101010100010100000010'
  строка res
  цел i, j, k, n

  для i от 1 до 8
  нц
    res = ''
    j = i
    пока j + 8 <= Length(s)
    нц
```

```
      n = 0
      для k от 0 до 7
      нц
        n = n * 2
        если str[k + j] = '1' то
          n = n + 1
        всё
      кц
      res = res + chr(n – 0x50 + ord('A'))      // 0x50 – число, записанное в 16-ричной системе счисления
      j = j + 8
    кц
    вывод res
  кц
кон
```

Задание 5

Разработайте алгоритм, который определяет (в порядке убывания) номера разрядов, содержащих цифру 8 в десятичной записи числа 64^{216}.

Схема решения. Число 64^{216} 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 64^{216} можно будет просмотреть этот массив и найти номера разрядов, содержащих цифру 8. Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первое число 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 прибавить перенос из предыдущего разряда и 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 сложения чисел будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 10, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 10) и запоминаем наличие переноса в следующий разряд.

Решение варианта 7101 для 10 классов

Задание 1

Древние Майя использовали 20-ичную систему счисления за одним исключением: во втором разряде было не 20, а 18 ступеней, то есть за числом (17)(19) сразу следовало число (1)(0)(0). Это было сделано для облегчения расчётов календарного цикла, поскольку (1)(0)(0) = 360 – примерно равно числу дней в солнечном году. Разработайте алгоритм сложения натуральных чисел в системе Майя.

Решение. Для хранения чисел будем использовать массив, каждый элемент которого будет хранить одну цифру числа в системе счисления Майя. Положим, что 100 цифр хватит для хранения чисел. Для сложения чисел будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 20, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 20) и запоминаем наличие переноса в следующий разряд. Второй разряд обрабатывается отдельно – если сумма цифр и переноса больше или равна 18, то вычитаем 18. Поскольку второй разряд нужно обрабатывать отдельно, то и первый разряд тоже проще обрабатывать отдельно, а цикл начинать с обработки третьего разряда.

Предполагается, что заполнены все элементы массива (т.е. в старших разрядах стоят незначащие нули). При переполнении возвращается некорректный результат.

Для перевода строки с числом Майя во внутреннее представление просматриваем все элементы строки – она должна состоять из нескольких групп, состоящих в свою очередь из открывающей скобки, нескольких цифр и закрывающей скобки. Если эта последовательность нарушается, то возвращаем сообщение об ошибке. Цифры записываем в другую строку и после нахождения закрывающей скобки переводим строку в число. Если число оказывается слишком большим, возвращаем сообщение об ошибке. Сообщение об ошибке также возвращается, если в строке оказывается больше 100 цифр Майя. После просмотра строки, переставляем полученные элементы массива в обратном порядке, т.к. в строке сначала расположены цифры старших разрядов, а нам надо, чтобы в элементах массива с меньшими номерами находились цифры младших разрядов. Оставшиеся разряды заполняем нулями. Также отдельно надо проверить цифру второго разряда, которая не должна быть больше 17.

```
число = запись
  цел digits[100]                              // Массив цифр
  цел count                                    // Количество цифр в числе
кон

цел константа NoError = 0, TooManyDigits = 1, IncorrectDigit = 2, IncorrectString = 3

алг ЧислаВСистемеМайя(арг строка num, рез число res, цел error)
нач
  строка str
  число num1, num2, res
  цел error, i

  ввод str
  ВводЧислаВСистемеМайя(str, num1, error)
  если error = TooManyDigits то
    вывод 'Слишком много цифр в числе'
  иначе
    если error = IncorrectDigit то
      вывод 'Неверное значение цифры'
    иначе
      если error = IncorrectString то
        вывод 'Строка имеет неверный формат'
      иначе
        ввод str
        ВводЧислаВСистемеМайя(str, num2, error)
        если error = TooManyDigits то
          вывод 'Слишком много цифр в числе'
        иначе
          если error = IncorrectDigit то
```

```
            вывод 'Неверное значение цифры'
          иначе
            если error = IncorrectString то
              вывод 'Строка имеет неверный формат'
            иначе
              СложениеЧиселВСистемеМайя(num1, num2, res)
              для i от res.count до 1 шаг -1
              нц
                вывод '(', res.digits[i], ')'
              кц
            всё
          всё
        всё
      всё
    всё
  всё
кон

алг ВводЧислаВСистемеМайя(арг строка num, рез число res, цел error)
нач
  цел константа BRACKET = 1, DIGIT = 2
  цел state = BRACKET, i, n
  строка d

  error = NoError
  res.count = 0
  i = 1
  пока i <= Length(num) и error = NoError
  нц
    d = ''
    если state = BRACKET и num[i] = '(' то
      state = DIGIT
    иначе
      если state = DIGIT и '0' <= num[i] и num[i] <= '9' то
        d = d + num[i]
      иначе
        если state = DIGIT и num[i] == ')' то
          если res.count = 100 то
            error = TooManyDigits
          иначе
            state = BRACKET
            res.count = res.count + 1
            res.digits[res.count] = цел(d)      // Преобразуем строку в целое число
            если res.digits[res.count] > 19 то
              error = IncorrectDigit
            всё
          всё
        иначе
          error = IncorrectString
        всё
      всё
    всё
    i = i + 1
  кц

  для i от 1 до res.count / 2
  нц
    n = res.digits[i]
    res.digits[i] = res.digits[res.count – i + 1]
    res.digits[res.count – i + 1] = n
  кц

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
  кц

  если res.digits[2] > 17 то
    error = IncorrectDigit
  всё
кон

алг СложениеЧиселВСистемеМайя(арг число num1, арг число num2, рез число res)
нач
```

```
  цел tr, i

  res.digits[1] = num1.digits[1] + num2.digits[1]
  tr = 0
  если res.digits[1] >= 20 то
    tr = 1
    res.digits[1] = res.digits[1] – 20
  всё
  res.digits[2] = num1.digits[2] + num2.digits[2] + tr
  tr = 0
  если res.digits[2] >= 18 то
    tr = 1
    res.digits[2] = res.digits[2] – 18
  всё

  res.count = max(num1.count, num2.count)
  для i от 3 до res.count
  нц
    res.digits[i] = num1.digits[i] + num2.digits[i] + tr
    tr = 0
    если res.digits[i] >= 20 то
      tr = 1
      res.digits[i] = res.digits[i] – 20
    всё
    i = i + 1
  кц

  если tr = 1 и res.count < 100 то
    res.count = res.count + 1
    res.digits[res.count] = 1
  всё

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
  кц
кон

алг max(арг цел n1, арг цел n2)
нач
  если n1 > n2 то
    вернуть n1
  иначе
    вернуть n2
  всё
кон
```

Задание 2

В теории чисел триморфное число – это число, десятичная запись куба которого оканчивается цифрами самого этого числа. Например, $4^3 = 64$, $24^3 = 13\,824$, $249^3 = 15\,438\,249$. Разработайте алгоритм нахождения триморфных чисел в диапазоне от *u* до *v*.

Решение. Перебираем числа из диапазона от *u* до *v*. Каждое число возводим в третью степень. Далее надо сравнить последние цифры нового числа с исходным. Для этого надо взять остаток от деления нового числа на 10 в некоторой степени. Степень зависит от количества цифр в исходном числе. Поэтому исходное число делим несколько раз на 10, пока не получится 0, одновременно столько же раз умножаем на 10 делитель, изначально равный 1. Затем берём остаток от деления нового числа на полученный делитель, и если этот остаток от деления равен исходному числу, то исходное число является триморфным.

```
алг ТриморфныеЧисла()
нач
  цел u, v, n, t, d

  ввод u, v

  для n от u до v
  нц
```

```
    d = 10
    i = n
    пока i >= 10
    нц
      d = d * 10
      i = i div 10
    кц
    если (n * n * n) mod d = n то
      вывод 'Число ', n, ' является триморфным'
  кц
кон
```

Задание 3

В качестве ключа для шифрования секретных сведений использовалось число S, являющееся суммой некоторых целых положительных чисел A, B и C ($A < B < C$). Причём $B - A = C - B$. Для дешифровки используется число B. Найти число B, если известно число S.

Единственная строка входных данных содержит целое положительное число не длиннее 100 знаков – число S.

Выходные данные содержат искомое число B, или слово "Ошибка", если не существует чисел A, B, C, удовлетворяющих условию задачи.

Примеры

Исходные данные	Результат
111111111	37037037
1000000000	Ошибка
603360336033	201120112011

Решение. $S = A + B + C$. Поскольку $B - A = C - B$, то можно записать следующее уравнение $S = A + (A + x) + (A + 2 \cdot x)$. Отсюда $S = 3 \cdot A + 3 \cdot x$. Следовательно, $B = A + x = S / 3$. Поэтому если S делится на 3, то $B = S / 3$, а если не делится – решения нет. Решения также нет при $S = 3$, т.к. в этом случае $B = 1$, и нельзя подобрать положительное значение A, меньшее B.

```
алг Ключ()
нач
  цел s

  ввод s

  если s <= 3 то
    вывод 'Неверные исходные данные'
  иначе
    если s mod 3 <> 0 то
      вывод 'Ошибка'
    иначе
      вывод 'B = ', S div 3
    всё
  всё
кон
```

Задание 4

Даны 4 слова. Разработайте алгоритм, проверяющий, можно ли из данных слов составить кроссворд при условии, что каждое слово обязательно пересекается с двумя другими и располагается только горизонтально или вертикально в обычной последовательности (сверху вниз или слева направо). Если решение существует, то вывести его на экран.

Решение. Необходимо перебрать все возможные перестановки слов. Первое слово из перестановки располагаем вертикально, второе и третье – горизонтально, а четвёртое – снова

вертикально. Нужно найти одинаковые буквы в 1-ом и 2-ом словах и в 1-ом и 3-ем словах, при этом буква 1-го слова, совпадающая с буквой 2-го слова, должна находиться раньше, чем буква 1-го слова, совпадающая с буквой 3-го слова. Аналогично нужно найти одинаковые буквы в 4-ом и 2-ом словах и в 4-ом и 3-ем словах. Если в какой-то перестановке мы находим буквы, удовлетворяющие всем условиям, значит, мы нашли подходящее решение.

Для перебора всех перестановок будем использовать следующий алгоритм. Сначала формируем первую отсортированную перестановку (1, 2, 3, 4). В текущей перестановке надо найти первый с конца элемент, который меньше следующего за ним элемента (самый последний, естественно, не рассматривается). Положим, такой элемент стоит на некотором *i*-ом месте. Далее среди элементов от последнего до (i + 1)-го надо найти элемент, который больше *i*-го, и поменять их местами. После этого элементы от (i + 1)-го до последнего надо расставить в порядке возрастания. В данном случае для этого не нужно использовать методы сортировки, достаточно переставить эти элементы в обратном порядке. Если нельзя найти элемент, который меньше следующего за ним, значит, перестановка была последней.

Задание 5

В теории чисел факторионом первого рода называются числа, равные произведению факториалов своих цифр. На сегодня неизвестно, конечно ли их число. Разработайте алгоритм нахождения таких чисел в диапазоне от *u* до *v*.

Решение. Перебираем все числа из диапазона от *u* до *v*. Для каждого числа выделяем цифры путём последовательного получения остатка от деления на 10 и деления на 10, находим факториал выделенной цифры и произведение всех факториалов. Если произведение равно исходному числу, значит, исходное число является факторионом. Чтобы не искать факториалы цифр многократно, подсчитаем их один раз, занесём в массив и будем брать значения из массива.

```
алг Факторион()
нач
  цел u, v, n, i, p
  цел fact[0..9]

  fact[0] = 1
  для i от 1 до 9
  нц
    fact[i] = fact[i – 1] * i
  кц

  ввод u, v

  для n от u до v
  нц
    i = n
    p = 1
    пока i > 0
    нц
      p = p * fact[i mod 10]
      i = i div 10
    кц
    если p = n то
      вывод 'Число ', n, ' является факторионом'
    всё
  кц
кон
```

Решение варианта 7102 для 10 классов

Задание 1

Как известно, современная система измерения времени ведёт начало от древнего Вавилона, где использовались 60-ричная с.с. Разработайте алгоритм умножения натуральных чисел в 60-й с.с. Каждая цифра 60-ричной с.с. записывается в десятичной системе в круглых скобках, например, (21).

Решение. Для хранения чисел будем использовать массив, каждый элемент которого будет хранить одну цифру числа в 60-ричной системе счисления. Для умножения используем алгоритм умножения «в столбик». Первое число надо умножить на каждую цифру второго числа и полученные результаты сложить, учитывая, что результат умножения на очередную цифру надо сдвигать влево на соответствующее количество позиций. Для ускорения работы алгоритма желательно отдельно рассмотреть случаи умножения числа на цифру 0 и цифру 1.

Для умножения числа на другое число из одной цифры будем последовательно, начиная с последней, умножать цифры первого числа на второе. Если произведение, сложенное с переносом из предыдущего разряда, больше или равно 60, вычисляем остаток от деления на 60 и результат деления на 60. Очередная цифра произведения есть остаток от деления. Результат деления есть перенос в следующий разряд.

Для сложения чисел будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 60, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 60) и запоминаем наличие переноса в следующий разряд.

Предполагается, что заполнены все элементы массива (т.е. в старших разрядах стоят незначащие нули). При переполнении возвращается некорректный результат.

Для перевода строки с числом в 60-ричной системе счисления во внутреннее представление просматриваем все элементы строки – она должна состоять из нескольких групп, состоящих в свою очередь из открывающей скобки, нескольких цифр и закрывающей скобки. Если эта последовательность нарушается, то возвращаем сообщение об ошибке. Цифры записываем в другую строку и после нахождения закрывающей скобки переводим строку в число. Если число оказывается слишком большим, возвращаем сообщение об ошибке. Сообщение об ошибке также возвращается, если в строке оказывается больше 100 цифр. После просмотра строки, переставляем полученные элементы массива в обратном порядке, т.к. в строке сначала расположены цифры старших разрядов, а нам надо, чтобы в элементах массива с меньшими номерами находились цифры младших разрядов. Оставшиеся разряды заполняем нулями.

```
число = запись
  цел digits[100]                              // Массив цифр
  цел count                                    // Количество цифр в числе
кон

цел константа NoError = 0, TooManyDigits = 1, IncorrectDigit = 2, IncorrectString = 3

алг ЧислаВ60Системе(арг строка num, рез число res, цел error)
нач
  строка str
  число num1, num2, res
  цел error, i

  ввод str
  ВводЧислаВ60Системе(str, num1, error)
  если error = TooManyDigits то
    вывод 'Слишком много цифр в числе'
  иначе
    если error = IncorrectDigit то
      вывод 'Неверное значение цифры'
    иначе
      если error = IncorrectString то
        вывод 'Строка имеет неверный формат'
```

```
      иначе
        ввод str
        ВводЧислаВ60Системе(str, num2, error)
        если error = TooManyDigits то
          вывод 'Слишком много цифр в числе'
        иначе
          если error = IncorrectDigit то
            вывод 'Неверное значение цифры'
          иначе
            если error = IncorrectString то
              вывод 'Строка имеет неверный формат'
            иначе
              УмножениеЧиселВ60Системе(num1, num2, res)
              для i от res.count до 1 шаг -1
              нц
                вывод '(', res.digits[i], ')'
              кц
            всё
          всё
        всё
      всё
    всё
  всё
кон

алг ВводЧислаВ60Системе(арг строка num, рез число res, цел error)
нач
  цел константа BRACKET = 1, DIGIT = 2
  цел state = BRACKET, i, n
  строка d

  error = NoError
  res.count = 0
  i = 1
  пока i <= Length(num) и error = NoError
  нц
    d = ''
    если state = BRACKET и num[i] = '(' то
      state = DIGIT
    иначе
      если state = DIGIT и '0' <= num[i] и num[i] <= '9' то
        d = d + num[i]
      иначе
        если state = DIGIT и num[i] == ')' то
          если res.count = 100 то
            error = TooManyDigits
          иначе
            state = BRACKET
            res.count = res.count + 1
            res.digits[res.count] = цел(d)      // Преобразуем строку в целое число
            если res.digits[res.count] > 59 то
              error = IncorrectDigit
            всё
          всё
        иначе
          error = IncorrectString
        всё
      всё
    всё
    i = i + 1
  кц

  для i от 1 до res.count / 2
  нц
    n = res.digits[i]
    res.digits[i] = res.digits[res.count – i + 1]
    res.digits[res.count – i + 1] = n
  кц

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
  кц
кон
```

```
алг УмножениеЧиселВ60Системе(арг число num1, арг число num2, рез число res)
нач
  число mul
  цел i, j, tr

  для i от 1 до 100
  нц
    res.digits[i] = 0
  кц
  res.count = 1

  для i от 1 до num2.count
  нц
    если num2.digits[i] <> 0 то
      для j от 1 до  i – 1
      нц
        mul[j] = 0
      кц
      tr = 0
      для j от 1 до num1.count
      нц
        mul.digits[j + i – 1] = num1.digits[j] * num2.digits[i] + tr
        tr = mul.digits[j + i – 1] div 60
        mul.digits[j + i – 1] = mul.digits[j + i – 1] mod 60
      кц
      mul.count = num1.count + i - 1
      если tr <> 0 и mul.count < 100 то
        mul.count = mul.count + 1
        mul.digits[mul.count] = tr
      всё
      для i от mul.count + 1 до 100
      нц
        mul.digits[i] = 0
      кц
      СложениеЧиселВ60Системе(res, res, mul)
    всё
  кц

кон

алг СложениеЧиселВ60Системе(арг число num1, число цел num2, рез число res)
нач
  цел tr, i

  tr = 0
  res.count = max(num1.count, num2.count)
  для i от 1 до res.count
  нц
    res.digits[i] = num1.digits[i] + num2.digits[i] + tr
    tr = 0
    если res.digits[i] >= 60 то
      tr = 1
      res.digits[i] = res.digits[i] – 60
    всё
    i = i + 1
  кц

  если tr = 1 и res.count < 100 то
    res.count = res.count + 1
    res.digits[res.count] = 1
  всё

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
  кц
кон

алг max(арг цел n1, арг цел n2)
нач
  если n1 > n2 то
    вернуть n1
  иначе
```

```
    вернуть n2
  всё
кон
```

Задание 2

Функция Эйлера $\varphi(x)$ – это функция, равная количеству натуральных чисел, меньших x и взаимно простых с x. Число n называется нетотиентным, если $\varphi(x) \neq n$ для любого x. Разработайте алгоритм нахождения нетотиентных чисел в диапазоне от c до d. Два натуральных числа называются взаимно простыми, если они не имеют никаких общих делителей, кроме единицы. Доказано, что для некоторого n функция $\varphi(x)$ может быть равна n, если x лежит в диапазоне от $n + 1$ до $A(n)$, где $A(n) = n \cdot \prod_{p-1|n} \frac{p}{p-1}$ для всех $(p - 1)$, являющихся делителями n (начиная от 1 и включая n), при этом p должно быть простым числом.

Решение. Перебираем все числа n из диапазона от c до d. Известно, что все нечётные числа, кроме 1, являются нетотиентным, т.к. функция Эйлера принимает только чётные значения (исключением является $\varphi(1) = 1$). Поэтому нечётные числа можно не проверять.

Для каждого чётного числа n надо вычислить оценку $A(n)$. Все числа делятся на 1, а $2 = 1 + 1$ является простым числом. Кроме того, чётные числа делятся на 2, а $3 = 2 + 1$ также является простым числом. Поэтому начальное значение оценки n можно сразу умножить на $\frac{2}{1} \cdot \frac{3}{2}$. После этого перебираем все числа от 4 до $n / 2$, и если текущее число является делителем n, а число, на 1 большее, является простым, умножаем значение оценки на следующую дробь. Все числа также делятся на себя, поэтому после цикла проверяем, что число $n + 1$ является простым, и если это действительно так, умножаем оценку ещё на одну дробь. Поскольку дроби могут не сокращаться друг с другом, то нужно выполнять вещественное деление, чтобы значение оценки не оказалось заниженным.

При нахождении оценки требуется много чисел проверять на простоту. Чтобы ускорить этот процесс построим с помощью решета Эратосфена массив простых чисел в диапазоне от 2 до $d + 1$ (число d будет делиться на себя, и потребуется проверить на простоту число $d + 1$).

Для использования решета Эратосфена необходимо построить массив чисел в заданном диапазоне от 1 до $d + 1$. Поскольку число 1 не является простым, в элемент массива с индексом 1 занесём значение 0 – будет удобнее, если индекс массива и число в массиве совпадают. Затем для чисел i в диапазоне от 2 до $\sqrt{d+1}$, начиная с числа i, вычёркиваем из массива (заменяем нулями) все числа с шагом i (само число i не вычёркивается). Для нахождения следующего значения i нужно найти первый незачёркнутый (ненулевой) элемент массива после текущего значения i.

После нахождения оценки $A(n)$ перебираем все числа x в диапазоне от $n + 1$ до $A(n)$, и для каждого x вычисляем значение функции Эйлера, т.е. ищем количество чисел, меньших его и взаимно простых с ним. Если это количество для всех x из указанного диапазона не равно n, то n является нетотиентным числом.

Для проверки, что числа являются взаимно простыми, надо найти НОД по алгоритму Евклида. Если НОД равен 1, то числа являются взаимно простыми.

Для упрощения процесса разработки алгоритма используем несколько вспомогательных алгоритмов: для построения массива простых чисел, вычисления оценки $A(n)$, вычисления функции Эйлера и нахождения НОД.

В первом вспомогательном алгоритме мы будет строить массив простых чисел, а во втором – использовать этот массив. Для того чтобы использовать массив в двух алгоритмах, будет удобнее объявить его как глобальную переменную. Глобальные переменные – это такие

переменные, которые обычно объявляются вне любой процедуры и функции, и при этом доступны во всей программе.

Существует ряд критериев качества программы. В первую очередь, это, конечно, корректность, надёжность и эффективность. Но не менее важным критерием является читабельность.

Любые средства, используемые в программе, должны улучшать качество программы. В данном случае, использование глобальной переменной должно улучшать читабельность программы и упрощать её понимание. Но ни в коем случае нельзя увлекаться глобальными переменными. Если объявлять все переменные как глобальные, то получится одна большая куча. Но в данном случае массив простых чисел является уникальной переменной, она действительно глобальна в программе, поэтому объявление такого массива как глобального является оправданным.

```
цел nums[10000]

алг НетотиентныеЧисла()
нач
  цел c, d, n, k, i, x
  лог f

  ввод c, d

  РешетоЭратосфена(d + 1)

  для n от c до d
  нц
    если n mod 2 = 1 то
      если n <> 1 то
        вывод n, ' является нетотиентным числом'
      всё
    иначе
      f = истина
      x = n + 1
      пока x <= A(n) и f
      нц
        если phi(x) = n то
          f = ложь
        всё
        x = x + 1
      кц
      если f то
        вывод n, ' является нетотиентным числом'
      всё
    всё
  кц
кон

алг РешетоЭратосфена(арг цел n)
  цел i

  nums[1] = 0
  для i от 2 до n
  нц
    nums[i] = i
  кц

  i = 2
  пока i <= целая_часть(sqrt(n))
  нц
    для j от 2 * i до n шаг i
    нц
      nums[j] = 0
    кц
    выполнить
      i = i + 1
    до nums[i] <> 0
  кц
кон
```

```
алг A(арг цел n)
  цел i
  вещ a

  a = n * 3
  для i от 4 до n div 2
  нц
    если n mod i = 0 и nums[i + 1] <> 0 то
      a = a * (i + 1) / i
    всё
  кц
  если nums[n + 1] <> 0 то
    a = a * (n + 1) / n
  всё
  вернуть a
кон

алг phi(арг цел x)
  цел p, i

  p = 0
  для i от 1 до x – 1
  нц
    если НОД(x, i) = 1 то
      p = p + 1
    всё
  кц
  вернуть p
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
```

Задание 3

В основе алгоритма RSA лежит использование пары простых чисел P и Q и производного числа (модуля) $N = P * Q$. Простое число – это натуральное число, которое имеет ровно два различных натуральных делителя: единицу и самого себя.

Принципиальным отличием нового алгоритма RSA++ от алгоритма RSA состоит в выборе ключей. Если в алгоритме RSA требуется пара простых чисел P и Q, то в алгоритме RSA++ числа P и Q должны быть взаимно простыми, т.е. они имеют только один общий делитель, равный 1.

Для анализа надёжности нового алгоритма необходимо узнать количество различных пар чисел P и Q, таких, что $1 < P < Q$ и соответствующий им модуль удовлетворяет условию $N \le K$.

Первая строка входных данных содержит одно целое число K ($1 \le K \le 109$).

Результат должен содержать одно целое число – количество различных пар чисел P и Q.

Примеры

Входные данные	**Результат**
12	3
18	6

Решение. Число P меняется в диапазоне от 2 до $\sqrt{K}$. Число Q меняется в диапазоне от $P + 1$ до K / P. Перебираем все возможные значения P и соответствующие им значения Q. Если числа являются взаимно простыми, прибавляем единицу к итоговому результату.

Для проверки, что числа являются взаимно простыми, надо найти НОД по алгоритму Евклида. Если НОД равен 1, то числа являются взаимно простыми.

```
алг АнализНадёжности()
нач
  цел k, p, q, n

  ввод k

  если k < 1 или k > 109 то
    вывод 'Неверное значение K'
  иначе
    n = 0
    для p от 2 до целая_часть(sqrt(k))
    нц
      для q от p + 1 до k div p
      нц
        если НОД(p, q) = 1 то
          n = n + 1
        всё
      кц
    кц
    вывод 'Количество пар взаимно простых чисел равно ', n
  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
```

Задание 4

Даны 8 слов. Разработайте алгоритм, проверяющий, можно ли из данных слов составить кроссворд при условии, что слова располагаются лесенкой (см. рисунок) и каждое слово, кроме первого и последнего, пересекается с двумя другими и располагается только горизонтально или вертикально в обычной последовательности (сверху вниз или слева направо). Если решение существует, то вывести его на экран.

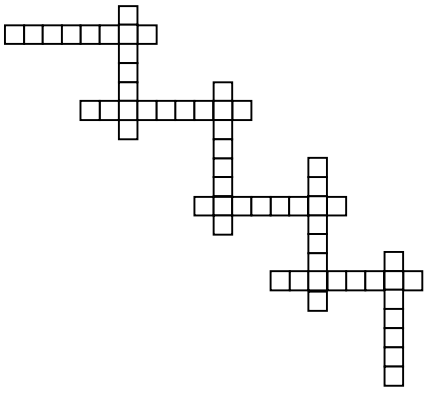

Схема решения. Надо найти такую перестановку слов, чтобы 1-ое слово из перестановки имело общую букву со 2-ым словом, 2-ое слово имело общую букву с 3-им словом, причём во 2-ом слове буква, общая с 3-им словом, должна стоять позже буквы, общей с 1-ым словом, и т.д. Если в какой-то перестановке мы находим буквы, удовлетворяющие всем условиям, значит, мы нашли подходящее решение.

Для перебора всех перестановок будем использовать следующий алгоритм. Сначала формируем первую отсортированную перестановку (1, 2, 3, 4, 5, 6, 7, 8). В текущей перестановке надо найти первый с конца элемент, который меньше следующего за ним элемента (самый последний, естественно, не рассматривается). Положим, такой элемент стоит на некотором i-ом месте. Далее среди элементов от последнего до $(i + 1)$-го надо найти элемент, который больше i-го, и

поменять их местами. После этого элементы от $(i+1)$-го до последнего надо расставить в порядке возрастания. В данном случае для этого не нужно использовать методы сортировки, достаточно переставить эти элементы в обратном порядке. Если нельзя найти элемент, который меньше следующего за ним, значит, перестановка была последней.

Задание 5

В теории чисел простым числом Вагстафа (Wagstaff) называется простое число p вида $p = \frac{2^q + 1}{3}$, где q – другое простое число. На сегодня неизвестно, конечно ли их число. Разработайте алгоритм нахождения таких чисел в диапазоне от u до v.

Решение. Построим массив простых чисел с помощью решета Эратосфена. Для этого сначала необходимо построить массив всех чисел в диапазоне от 1 до v. Поскольку число 1 не является простым, в элемент массива с индексом 1 занесём значение 0 – будет удобнее, если индекс массива и число в массиве совпадают. Затем для чисел i в диапазоне от 2 до $\sqrt{v}$, начиная с числа i, вычёркиваем из массива (заменяем нулями) все числа с шагом i (само число i не вычёркивается). Для нахождения следующего значения i нужно найти первый незачёркнутый (ненулевой) элемент массива после текущего значения i. Для удобства удалим нулевые элементы массива, чтобы найденные простые числа были расположены последовательно.

Далее ищем в построенном массиве первое простое число q, для которого значение выражения $\frac{2^q + 1}{3}$ будет больше или равно u. Для этого и следующих чисел q, вычисляем значение p по формуле $\frac{2^q + 1}{3}$, и пока p меньше или равно v, проверяем, встречается ли значение p в массиве простых чисел. Если встречается, значит, p является искомым простым числом Вагстафа.

```
алг ЧислаВагстафа()
нач
  цел u, v, p, i, j, nums[v]

  ввод u, v

  nums[1] = 0
  для i от 2 до v
  нц
    nums[i] = i
  кц

  i = 2
  пока i <= целая_часть(sqrt(v))
  нц
    для j от 2 * i до v шаг i
    нц
      nums[j] = 0
    кц
    выполнить
      i = i + 1
    до nums[i] <> 0
  кц

  j = 0
  для i от 2 до v
  нц
    если nums[i] <> 0 то
      j = j + 1
      nums[j] = nums[i]
    всё
  кц
```

```
  i = 1
  пока (pow(2, nums[i]) + 1) / 3 < u
  нц
    i = i + 1
  кц

  p = (pow(2, nums[i]) + 1) / 3
  j = 1
  пока p <= v
  нц
    пока nums[j] < p
    нц
      j = j + 1
    кц
    если nums[j] = p то
      вывод p, ' является простым числом Вагстафа'
    всё
    i = i + 1
    p = (pow(2, nums[i]) + 1) / 3
  кц
кон
```

Решение варианта 7111 для 11 классов

Задание 1

Древние Майя использовали 20-ичную систему счисления за одним исключением: во втором разряде было не 20, а 18 ступеней, то есть за числом (17)(19) сразу следовало число (1)(0)(0). Это было сделано для облегчения расчётов календарного цикла, поскольку (1)(0)(0) = 360 – примерно равно числу дней в солнечном году. Разработайте алгоритм сложения натуральных чисел в системе Майя.

Решение. Для хранения чисел будем использовать массив, каждый элемент которого будет хранить одну цифру числа в системе счисления Майя. Положим, что 100 цифр хватит для хранения чисел. Для сложения чисел будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 20, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 20) и запоминаем наличие переноса в следующий разряд. Второй разряд обрабатывается отдельно – если сумма цифр и переноса больше или равна 18, то вычитаем 18. Поскольку второй разряд нужно обрабатывать отдельно, то и первый разряд тоже проще обрабатывать отдельно, а цикл начинать с обработки третьего разряда.

Предполагается, что заполнены все элементы массива (т.е. в старших разрядах стоят незначащие нули). При переполнении возвращается некорректный результат.

Для перевода строки с числом Майя во внутреннее представление просматриваем все элементы строки – она должна состоять из нескольких групп, состоящих в свою очередь из открывающей скобки, нескольких цифр и закрывающей скобки. Если эта последовательность нарушается, то возвращаем сообщение об ошибке. Цифры записываем в другую строку и после нахождения закрывающей скобки переводим строку в число. Если число оказывается слишком большим, возвращаем сообщение об ошибке. Сообщение об ошибке также возвращается, если в строке оказывается больше 100 цифр Майя. После просмотра строки, переставляем полученные элементы массива в обратном порядке, т.к. в строке сначала расположены цифры старших разрядов, а нам надо, чтобы в элементах массива с меньшими номерами находились цифры младших разрядов. Оставшиеся разряды заполняем нулями. Также отдельно надо проверить цифру второго разряда, которая не должна быть больше 17.

```
число = запись
  цел digits[100]                              // Массив цифр
  цел count                                    // Количество цифр в числе
кон

цел константа NoError = 0, TooManyDigits = 1, IncorrectDigit = 2, IncorrectString = 3

алг ЧислаВСистемеМайя(арг строка num, рез число res, цел error)
нач
  строка str
  число num1, num2, res
  цел error, i

  ввод str
  ВводЧислаВСистемеМайя(str, num1, error)
  если error = TooManyDigits то
    вывод 'Слишком много цифр в числе'
  иначе
    если error = IncorrectDigit то
      вывод 'Неверное значение цифры'
    иначе
      если error = IncorrectString то
        вывод 'Строка имеет неверный формат'
      иначе
        ввод str
        ВводЧислаВСистемеМайя(str, num2, error)
        если error = TooManyDigits то
          вывод 'Слишком много цифр в числе'
        иначе
          если error = IncorrectDigit то
```

```
            вывод 'Неверное значение цифры'
          иначе
            если error = IncorrectString то
              вывод 'Строка имеет неверный формат'
            иначе
              СложениеЧиселВСистемеМайя(num1, num2, res)
              для i от res.count до 1 шаг -1
              нц
                вывод '(', res.digits[i], ')'
              кц
            всё
          всё
        всё
      всё
    всё
  всё
кон

алг ВводЧислаВСистемеМайя(арг строка num, рез число res, цел error)
нач
  цел константа BRACKET = 1, DIGIT = 2
  цел state = BRACKET, i, n
  строка d

  error = NoError
  res.count = 0
  i = 1
  пока i <= Length(num) и error = NoError
  нц
    d = ''
    если state = BRACKET и num[i] = '(' то
      state = DIGIT
    иначе
      если state = DIGIT и '0' <= num[i] и num[i] <= '9' то
        d = d + num[i]
      иначе
        если state = DIGIT и num[i] == ')' то
          если res.count = 100 то
            error = TooManyDigits
          иначе
            state = BRACKET
            res.count = res.count + 1
            res.digits[res.count] = цел(d)      // Преобразуем строку в целое число
            если res.digits[res.count] > 19 то
              error = IncorrectDigit
            всё
          всё
        иначе
          error = IncorrectString
        всё
      всё
    всё
    i = i + 1
  кц

  для i от 1 до res.count / 2
  нц
    n = res.digits[i]
    res.digits[i] = res.digits[res.count – i + 1]
    res.digits[res.count – i + 1] = n
  кц

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
  кц

  если res.digits[2] > 17 то
    error = IncorrectDigit
  всё
кон

алг СложениеЧиселВСистемеМайя(арг число num1, арг число num2, рез число res)
нач
```

```
  цел tr, i

  res.digits[1] = num1.digits[1] + num2.digits[1]
  tr = 0
  если res.digits[1] >= 20 то
    tr = 1
    res.digits[1] = res.digits[1] – 20
  всё
  res.digits[2] = num1.digits[2] + num2.digits[2] + tr
  tr = 0
  если res.digits[2] >= 18 то
    tr = 1
    res.digits[2] = res.digits[2] – 18
  всё

  res.count = max(num1.count, num2.count)
  для i от 3 до res.count
  нц
    res.digits[i] = num1.digits[i] + num2.digits[i] + tr
    tr = 0
    если res.digits[i] >= 20 то
      tr = 1
      res.digits[i] = res.digits[i] – 20
    всё
    i = i + 1
  кц

  если tr = 1 и res.count < 100 то
    res.count = res.count + 1
    res.digits[res.count] = 1
  всё

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
  кц
кон

алг max(арг цел n1, арг цел n2)
нач
  если n1 > n2 то
    вернуть n1
  иначе
    вернуть n2
  всё
кон
```

Задание 2

В теории чисел триморфное число – это число, десятичная запись куба которого оканчивается цифрами самого этого числа. Например, $4^3 = 64$, $24^3 = 13\,824$, $249^3 = 15\,438\,249$. Разработайте алгоритм нахождения триморфных чисел в диапазоне от *u* до *v*.

Решение. Перебираем числа из диапазона от *u* до *v*. Каждое число возводим в третью степень. Далее надо сравнить последние цифры нового числа с исходным. Для этого надо взять остаток от деления нового числа на 10 в некоторой степени. Степень зависит от количества цифр в исходном числе. Поэтому исходное число делим несколько раз на 10, пока не получится 0, одновременно столько же раз умножаем на 10 делитель, изначально равный 1. Затем берём остаток от деления нового числа на полученный делитель, и если этот остаток от деления равен исходному числу, то исходное число является триморфным.

```
алг ТриморфныеЧисла()
нач
  цел u, v, n, t, d

  ввод u, v

  для n от u до v
  нц
```

```
    d = 10
    i = n
    пока i >= 10
    нц
      d = d * 10
      i = i div 10
    кц
    если (n * n * n) mod d = n то
      вывод 'Число ', n, ' является триморфным'
  кц
кон
```

Задание 3

В качестве ключа для шифрования секретных сведений использовалось число S, являющееся суммой некоторых целых положительных чисел A, B и C ($A < B < C$). Причём $B - A = C - B$. Для дешифровки используется число B. Найти число B, если известно число S.

Единственная строка входных данных содержит целое положительное число не длиннее 100 знаков – число S.

Выходные данные содержат искомое число B, или слово "Ошибка", если не существует чисел A, B, C, удовлетворяющих условию задачи.

Примеры

Исходные данные	Результат
111111111	37037037
1000000000	Ошибка
603360336033	201120112011

Решение. $S = A + B + C$. Поскольку $B - A = C - B$, то можно записать следующее уравнение $S = A + (A + x) + (A + 2 \cdot x)$. Отсюда $S = 3 \cdot A + 3 \cdot x$. Следовательно, $B = A + x = S / 3$. Поэтому если S делится на 3, то $B = S / 3$, а если не делится – решения нет. Решения также нет при $S = 3$, т.к. в этом случае $B = 1$, и нельзя подобрать положительное значение A, меньшее B.

```
алг Ключ()
нач
  цел s

  ввод s

  если s <= 3 то
    вывод 'Неверные исходные данные'
  иначе
    если s mod 3 <> 0 то
      вывод 'Ошибка'
    иначе
      вывод 'B = ', S div 3
    всё
  всё
кон
```

Задание 4

Разработайте алгоритм, который определяет (в порядке возрастания) номера разрядов, содержащих цифру 2 в десятичной записи числа 64^{513}.

Схема решения. Число 64^{513} 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 64^{513} можно будет просмотреть этот массив и найти номера разрядов, содержащих цифру 2. Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первое число 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 прибавить перенос из предыдущего разряда и 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 сложения чисел будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 10, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 10) и запоминаем наличие переноса в следующий разряд.

Задание 5

В теории чисел факторионом первого рода называются числа, равные произведению факториалов своих цифр. На сегодня неизвестно, конечно ли их число. Разработайте алгоритм нахождения таких чисел в диапазоне от *u* до *v*.

Решение. Перебираем все числа из диапазона от *u* до *v*. Для каждого числа последовательно выделяем цифры путём последовательного получения остатка от деления на 10 и деления на 10, находим факториал выделенной цифры и произведение всех факториалов. Если произведение равно исходному числу, значит, исходное число является факторионом. Чтобы не искать факториалы цифр многократно, подсчитаем их один раз, занесём в массив и будем брать значения из массива.

```
алг Факторион()
нач
  цел u, v, n, i, p
  цел fact[0..9]

  fact[0] = 1
  для i от 1 до 9
  нц
    fact[i] = fact[i – 1] * i
  кц

  ввод u, v

  для n от u до v
  нц
    i = n
    p = 1
    пока i > 0
    нц
      p = p * fact[i mod 10]
      i = i div 10
    кц
    если p = n то
      вывод 'Число ', n, ' является факторионом'
    всё
  кц
кон
```

Решение варианта 7112 для 11 классов

Задание 1

Как известно, современная система измерения времени ведёт начало от древнего Вавилона, где использовались 60-ричная с.с. Разработайте алгоритм умножения натуральных чисел в 60-й с.с. Каждая цифра 60-ричной с.с. записывается в десятичной системе в круглых скобках, например, (21).

Решение. Для хранения чисел будем использовать массив, каждый элемент которого будет хранить одну цифру числа в 60-ричной системе счисления. Для умножения используем алгоритм умножения «в столбик». Первое число надо умножить на каждую цифру второго числа и полученные результаты сложить, учитывая, что результат умножения на очередную цифру надо сдвигать влево на соответствующее количество позиций. Для ускорения работы алгоритма желательно отдельно рассмотреть случаи умножения числа на цифру 0 и цифру 1.

Для умножения числа на другое число из одной цифры будем последовательно, начиная с последней, умножать цифры первого числа на второе. Если произведение, сложенное с переносом из предыдущего разряда, больше или равно 60, вычисляем остаток от деления на 60 и результат деления на 60. Очередная цифра произведения есть остаток от деления. Результат деления есть перенос в следующий разряд.

Для сложения чисел будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 60, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 60) и запоминаем наличие переноса в следующий разряд.

Предполагается, что заполнены все элементы массива (т.е. в старших разрядах стоят незначащие нули). При переполнении возвращается некорректный результат.

Для перевода строки с числом в 60-ричной системе счисления во внутреннее представление просматриваем все элементы строки – она должна состоять из нескольких групп, состоящих в свою очередь из открывающей скобки, нескольких цифр и закрывающей скобки. Если эта последовательность нарушается, то возвращаем сообщение об ошибке. Цифры записываем в другую строку и после нахождения закрывающей скобки переводим строку в число. Если число оказывается слишком большим, возвращаем сообщение об ошибке. Сообщение об ошибке также возвращается, если в строке оказывается больше 100 цифр. После просмотра строки, переставляем полученные элементы массива в обратном порядке, т.к. в строке сначала расположены цифры старших разрядов, а нам надо, чтобы в элементах массива с меньшими номерами находились цифры младших разрядов. Оставшиеся разряды заполняем нулями.

```
число = запись
  цел digits[100]                              // Массив цифр
  цел count                                    // Количество цифр в числе
кон

цел константа NoError = 0, TooManyDigits = 1, IncorrectDigit = 2, IncorrectString = 3

алг ЧислаВ60Системе(арг строка num, рез число res, цел error)
нач
  строка str
  число num1, num2, res
  цел error, i

  ввод str
  ВводЧислаВ60Системе(str, num1, error)
  если error = TooManyDigits то
    вывод 'Слишком много цифр в числе'
  иначе
    если error = IncorrectDigit то
      вывод 'Неверное значение цифры'
    иначе
```

```
      если error = IncorrectString то
        вывод 'Строка имеет неверный формат'
      иначе
        ввод str
        ВводЧислаВ60Системе(str, num2, error)
        если error = TooManyDigits то
          вывод 'Слишком много цифр в числе'
        иначе
          если error = IncorrectDigit то
            вывод 'Неверное значение цифры'
          иначе
            если error = IncorrectString то
              вывод 'Строка имеет неверный формат'
            иначе
              УмножениеЧиселВ60Системе(num1, num2, res)
              для i от res.count до 1 шаг -1
              нц
                вывод '(', res.digits[i], ')'
              кц
            всё
          всё
        всё
      всё
    всё
  всё
кон

алг ВводЧислаВ60Системе(арг строка num, рез число res, цел error)
нач
  цел константа BRACKET = 1, DIGIT = 2
  цел state = BRACKET, i, n
  строка d

  error = NoError
  res.count = 0
  i = 1
  пока i <= Length(num) и error = NoError
  нц
    d = ''
    если state = BRACKET и num[i] = '(' то
      state = DIGIT
    иначе
      если state = DIGIT и '0' <= num[i] и num[i] <= '9' то
        d = d + num[i]
      иначе
        если state = DIGIT и num[i] == ')' то
          если res.count = 100 то
            error = TooManyDigits
          иначе
            state = BRACKET
            res.count = res.count + 1
            res.digits[res.count] = цел(d)      // Преобразуем строку в целое число
            если res.digits[res.count] > 59 то
              error = IncorrectDigit
            всё
          всё
        иначе
          error = IncorrectString
        всё
      всё
    всё
    i = i + 1
  кц

  для i от 1 до res.count / 2
  нц
    n = res.digits[i]
    res.digits[i] = res.digits[res.count – i + 1]
    res.digits[res.count – i + 1] = n
  кц

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
```

```
  кц
кон

алг УмножениеЧиселВ60Системе(арг число num1, арг число num2, рез число res)
нач
  число mul
  цел i, j, tr

  для i от 1 до 100
  нц
    res.digits[i] = 0
  кц
  res.count = 1

  для i от 1 до num2.count
  нц
    если num2.digits[i] <> 0 то
      для j от 1 до  i – 1
      нц
        mul[j] = 0
      кц
      tr = 0
      для j от 1 до num1.count
      нц
        mul.digits[j + i – 1] = num1.digits[j] * num2.digits[i] + tr
        tr = mul.digits[j + i – 1] div 60
        mul.digits[j + i – 1] = mul.digits[j + i – 1] mod 60
      кц
      mul.count = num1.count + i - 1
      если tr <> 0 и mul.count < 100 то
        mul.count = mul.count + 1
        mul.digits[mul.count] = tr
      всё
      для i от mul.count + 1 до 100
      нц
        mul.digits[i] = 0
      кц
      СложениеЧиселВ60Системе(res, res, mul)
    всё
  кц

кон

алг СложениеЧиселВ60Системе(арг число num1, число цел num2, рез число res)
нач
  цел tr, i

  tr = 0
  res.count = max(num1.count, num2.count)
  для i от 1 до res.count
  нц
    res.digits[i] = num1.digits[i] + num2.digits[i] + tr
    tr = 0
    если res.digits[i] >= 60 то
      tr = 1
      res.digits[i] = res.digits[i] – 60
    всё
    i = i + 1
  кц

  если tr = 1 и res.count < 100 то
    res.count = res.count + 1
    res.digits[res.count] = 1
  всё

  для i от res.count + 1 до 100
  нц
    res.digits[i] = 0
  кц
кон

алг max(арг цел n1, арг цел n2)
нач
  если n1 > n2 то
```

```
    вернуть n1
  иначе
    вернуть n2
  всё
кон
```

Задание 2

Функция Эйлера $\varphi(x)$ – это функция, равная количеству натуральных чисел, меньших x и взаимно простых с x. Число n называется нетотиентным, если $\varphi(x) \neq n$ для любого x. Разработайте алгоритм нахождения нетотиентных чисел в диапазоне от c до d. Два натуральных числа называются взаимно простыми, если они не имеют никаких общих делителей, кроме единицы. Доказано, что для некоторого n функция $\varphi(x)$ может быть равна n, если x лежит в диапазоне от $n + 1$ до $A(n)$, где $A(n) = n \cdot \prod_{p-1|n} \frac{p}{p-1}$ для всех $(p - 1)$, являющихся делителями n (начиная от 1 и включая n), при этом p должно быть простым числом.

Решение. Перебираем все числа n из диапазона от c до d. Известно, что все нечётные числа, кроме 1, являются нетотиентным, т.к. функция Эйлера принимает только чётные значения (исключением является $\varphi(1) = 1$). Поэтому нечётные числа можно не проверять.

Для каждого чётного числа n надо вычислить оценку $A(n)$. Все числа делятся на 1, а $2 = 1 + 1$ является простым числом. Кроме того, чётные числа делятся на 2, а $3 = 2 + 1$ также является простым числом. Поэтому начальное значение оценки n можно сразу умножить на $\frac{2}{1} \cdot \frac{3}{2}$. После этого перебираем все числа от 4 до $n / 2$, и если текущее число является делителем n, а число, на 1 большее, является простым, умножаем значение оценки на следующую дробь. Все числа также делятся на себя, поэтому после цикла проверяем, что число $n + 1$ является простым, и если это действительно так, умножаем оценку ещё на одну дробь. Поскольку дроби могут не сокращаться друг с другом, то нужно выполнять вещественное деление, чтобы значение оценки не оказалось заниженным.

При нахождении оценки требуется много чисел проверять на простоту. Чтобы ускорить этот процесс построим с помощью решета Эратосфена массив простых чисел в диапазоне от 2 до $d + 1$ (число d будет делиться на себя, и потребуется проверить на простоту число $d + 1$).

Для использования решета Эратосфена необходимо построить массив чисел в заданном диапазоне от 1 до $d + 1$. Поскольку число 1 не является простым, в элемент массива с индексом 1 занесём значение 0 – будет удобнее, если индекс массива и число в массиве совпадают. Затем для чисел i в диапазоне от 2 до $\sqrt{d+1}$, начиная с числа i, вычёркиваем из массива (заменяем нулями) все числа с шагом i (само число i не вычёркивается). Для нахождения следующего значения i нужно найти первый незачёркнутый (ненулевой) элемент массива после текущего значения i.

После нахождения оценки $A(n)$ перебираем все числа x в диапазоне от $n + 1$ до $A(n)$, и для каждого x вычисляем значение функции Эйлера, т.е. ищем количество чисел, меньших его и взаимно простых с ним. Если это количество для всех x из указанного диапазона не равно n, то n является нетотиентным числом.

Для проверки, что числа являются взаимно простыми, надо найти НОД по алгоритму Евклида. Если НОД равен 1, то числа являются взаимно простыми.

Для упрощения процесса разработки алгоритма используем несколько вспомогательных алгоритмов: для построения массива простых чисел, вычисления оценки $A(n)$, вычисления функции Эйлера и нахождения НОД.

В первом вспомогательном алгоритме мы будет строить массив простых чисел, а во втором – использовать этот массив. Для того чтобы использовать массив в двух алгоритмах, будет удобнее объявить его как глобальную переменную. Глобальные переменные – это такие переменные, которые обычно объявляются вне любой процедуры и функции, и при этом доступны во всей программе.

Существует ряд критериев качества программы. В первую очередь, это, конечно, корректность, надёжность и эффективность. Но не менее важным критерием является читабельность.

Любые средства, используемые в программе, должны улучшать качество программы. В данном случае, использование глобальной переменной должно улучшать читабельность программы и упрощать её понимание. Но ни в коем случае нельзя увлекаться глобальными переменными. Если объявлять все переменные как глобальные, то получится одна большая куча. Но в данном случае массив простых чисел является уникальной переменной, она действительно глобальна в программе, поэтому объявление такого массива как глобального является оправданным.

```
цел nums[10000]

алг НетотиентныеЧисла()
нач
  цел c, d, n, k, i, x
  лог f

  ввод c, d

  РешетоЭратосфена(d + 1)

  для n от c до d
  нц
    если n mod 2 = 1 то
      если n <> 1 то
        вывод n, ' является нетотиентным числом'
      всё
    иначе
      f = истина
      x = n + 1
      пока x <= A(n) и f
      нц
        если phi(x) = n то
          f = ложь
        всё
        x = x + 1
      кц
      если f то
        вывод n, ' является нетотиентным числом'
      всё
    всё
  кц
кон

алг РешетоЭратосфена(арг цел n)
  цел i

  nums[1] = 0
  для i от 2 до n
  нц
    nums[i] = i
  кц

  i = 2
  пока i <= целая_часть(sqrt(n))
  нц
    для j от 2 * i до n шаг i
    нц
      nums[j] = 0
    кц
    выполнить
      i = i + 1
    до nums[i] <> 0
  кц
```

```
кон

алг A(арг цел n)
  цел i
  вещ a

  a = n * 3
  для i от 4 до n div 2
  нц
    если n mod i = 0 и nums[i + 1] <> 0 то
      a = a * (i + 1) / i
    всё
  кц
  если nums[n + 1] <> 0 то
    a = a * (n + 1) / n
  всё
  вернуть a
кон

алг phi(арг цел x)
  цел p, i

  p = 0
  для i от 1 до x - 1
  нц
    если НОД(x, i) = 1 то
      p = p + 1
    всё
  кц
  вернуть p
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
```

Задание 3

В основе алгоритма RSA лежит использование пары простых чисел P и Q и производного числа (модуля) $N = P * Q$. Простое число – это натуральное число, которое имеет ровно два различных натуральных делителя: единицу и самого себя.

Принципиальным отличием нового алгоритма RSA++ от алгоритма RSA состоит в выборе ключей. Если в алгоритме RSA требуется пара простых чисел P и Q, то в алгоритме RSA++ числа P и Q должны быть взаимно простыми, т.е. они имеют только один общий делитель, равный 1.

Для анализа надёжности нового алгоритма необходимо узнать количество различных пар чисел P и Q, таких, что $1 < P < Q$ и соответствующий им модуль удовлетворяет условию $N \le K$.

Первая строка входных данных содержит одно целое число K ($1 \le K \le 109$).

Результат должен содержать одно целое число – количество различных пар чисел P и Q.

Примеры

Входные данные	**Результат**
12	3
18	6

Решение. Число P меняется в диапазоне от 2 до $\sqrt{K}$. Число Q меняется в диапазоне от $P + 1$ до K / P. Перебираем все возможные значения P и соответствующие им значения Q. Если числа являются взаимно простыми, прибавляем единицу к итоговому результату.

Для проверки, что числа являются взаимно простыми, надо найти НОД по алгоритму Евклида. Если НОД равен 1, то числа являются взаимно простыми.

```
алг АнализНадёжности()
нач
  цел k, p, q, n

  ввод k

  если k < 1 или k > 109 то
    вывод 'Неверное значение K'
  иначе
    n = 0
    для p от 2 до целая_часть(sqrt(k))
    нц
      для q от p + 1 до k div p
      нц
        если НОД(p, q) = 1 то
          n = n + 1
        всё
      кц
    кц
    вывод 'Количество пар взаимно простых чисел равно ', n
  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
```

Задание 4

Разработайте алгоритм, который определяет (в порядке убывания) номера разрядов, содержащих цифру 4 в десятичной записи числа 64^{216}.

Схема решения. Число 64^{216} 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 64^{216} можно будет просмотреть этот массив и найти номера разрядов, содержащих цифру 4. Для умножения чисел реализуем школьный алгоритм умножения «в столбик». Первое число 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 прибавить перенос из предыдущего разряда и 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 сложения чисел

будем последовательно складывать цифры, начиная с последней. Если сумма двух цифр и переноса из предыдущего разряда больше или равна 10, то оставляем значение (<цифра$_1$> + <цифра$_2$> + <перенос> – 10) и запоминаем наличие переноса в следующий разряд.

Задание 5

В теории чисел простым числом Вагстафа (Wagstaff) называется простое число p вида $p = \frac{2^q + 1}{3}$, где q – другое простое число. На сегодня неизвестно, конечно ли их число. Разработайте алгоритм нахождения таких чисел в диапазоне от u до v.

Решение. Построим массив простых чисел с помощью решета Эратосфена. Для этого сначала необходимо построить массив всех чисел в диапазоне от 1 до v. Поскольку число 1 не является простым, в элемент массива с индексом 1 занесём значение 0 – будет удобнее, если индекс массива и число в массиве совпадают. Затем для чисел i в диапазоне от 2 до $\sqrt{v}$, начиная с числа i, вычёркиваем из массива (заменяем нулями) все числа с шагом i (само число i не вычёркивается). Для нахождения следующего значения i нужно найти первый незачёркнутый (ненулевой) элемент массива после текущего значения i. Для удобства удалим нулевые элементы массива, чтобы найденные простые числа были расположены последовательно.

Далее ищем в построенном массиве первое простое число q, для которого значение выражения $\frac{2^q + 1}{3}$ будет больше или равно u. Для этого и следующих чисел q, вычисляем значение p по формуле $\frac{2^q + 1}{3}$, и пока p меньше или равно v, проверяем, встречается ли значение p в массиве простых чисел. Если встречается, значит, p является искомым простым числом Вагстафа.

```
алг ЧислаВагстафа()
нач
  цел u, v, p, i, j, nums[v]

  ввод u, v

  nums[1] = 0
  для i от 2 до v
  нц
    nums[i] = i
  кц

  i = 2
  пока i <= целая_часть(sqrt(v))
  нц
    для j от 2 * i до v шаг i
    нц
      nums[j] = 0
    кц
    выполнить
      i = i + 1
    до nums[i] <> 0
  кц

  j = 0
  для i от 2 до v
  нц
    если nums[i] <> 0 то
      j = j + 1
      nums[j] = nums[i]
    всё
  кц

  i = 1
  пока (pow(2, nums[i]) + 1) / 3 < u
  нц
```

```
    i = i + 1
  кц

  p = (pow(2, nums[i]) + 1) / 3
  j = 1
  пока p <= v
  нц
    пока nums[j] < p
    нц
      j = j + 1
    кц
    если nums[j] = p то
      вывод p, ' является простым числом Вагстафа'
    всё
    i = i + 1
    p = (pow(2, nums[i]) + 1) / 3
  кц
кон
```