ЗАДАНИЕ ПО ИНФОРМАТИКЕ
ВАРИАНТ 32113 для 11 класса

Для заданий 1-5 требуется разработать алгоритмы на языке блок-схем, псевдокоде или естественном языке.

1. Найти сумму ряда чисел с точностью ε, $|x| \le 1$

$$1-\sqrt[4]{1-x}=\frac{x}{4}+\frac{3x^2}{4\cdot 8}+\frac{3\cdot 7x^3}{4\cdot 8\cdot 12}+\frac{3\cdot 7\cdot 11x^4}{4\cdot 8\cdot 12\cdot 16}+\frac{3\cdot 7\cdot 11\cdot 15x^5}{4\cdot 8\cdot 12\cdot 16\cdot 20}+\cdots+\frac{3\cdot 7\cdot 11\cdot\cdots\cdot(4n-5)x^n}{4\cdot 8\cdot 12\cdot 16\cdot\cdots\cdot(4n)}$$

Решение. Положим переменную *a* равной первому слагаемому $\frac{1x}{4}$, а переменную *s* равной переменной *a*. Далее в цикле переменную *a* надо умножать на рекуррентное соотношение, которое в данном случае равно $\frac{(4n-5)x}{(4n)}$, и прибавлять к переменной *s*. Цикл продолжается пока модуль очередного слагаемого не станет меньше точности ε.

```
алг СуммаРяда()
нач
  вещ x, e, a, s
  цел n

  ввод x, e
  если abs(x) > 1 или e > 1 или e <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    a = x / 4
    s = a
    n = 2
    пока abs(a) >= e
    нц
      a = a * (4 * n – 3) * x / (4 * n)
      s = s + a
      n = n + 1
    кц
    вывод s

  всё
кон
```

2. Даны *n* целых чисел $a_1, a_2, ..., a_n$ и натуральное число $m > 1$. Составить алгоритм для распределения чисел $a_1, a_2, ..., a_n$ так, чтобы сначала шли (по возрастанию абсолютной величины) все числа, дающие при делении на *m* остаток 0, затем 1, 2, ..., $m - 1$.

Решение. Для решения этой задачи надо использовать любой метод сортировки, но сравнивать не сами числа, а остаток от деления числа на *m*. При равенстве остатков от деления сравнивать модули чисел.

```
алг Перестановка()
нач
  цел n, a[n], m, i

  ввод m, n
  если m <= 0 или n <= 0 то
    вывод "Некорректные исходные данные"
  иначе

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

  QuickSortByMod(a, 1, n, m)
```

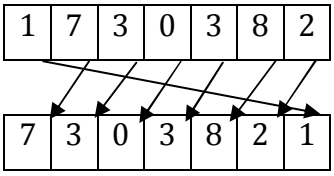
```
  для i от 1 до n
  нц
    вывод a[i]
  кц

  всё
кон

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

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

3. К массиву цифр натуральных чисел a ($a > 9$) применяется операция циклический **сдвиг** влево. Пример применения этой операции к числу 1730382 показан на рисунке.



Из числа 1730382 получено число 7303821. К этому числу опять можно применить сдвиг. К полученному тоже. Получается последовательность чисел 1730382, 7303821, 3038217, 0382173, 3821730, 8217303, 2173038, 1730382, ….

Составьте алгоритм, который для массива натуральных чисел a, которые могут содержать до 100 цифр, находит и выводит на экран массив наибольших чисел, получаемых сдвигами.

Решение (1 способ). Для каждого числа a сформируем массив цифр этого числа путём деления его на 10 и сохранения остатков от деления. При этом самая младшая цифра (с весом 10^0) будет записана в элемент массива с индексом 1, предпоследняя цифра (с весом 10^1) будет записана в элемент массива с индексом 2, и т.д. Также надо

подсчитать n – количество цифр числа. Само число a используем как начальное значение максимума. Далее надо $(n - 1)$ раз выполнить сдвиг массива цифр влево (после циклического сдвига на n позиций получится исходное число), получить соответствующее число и сравнить его с максимумом. Для выполнения сдвига влево надо запомнить n-ую (самую левую) цифру, далее все цифры с $(n - 1)$-ой до 1-ой записать в соседний слева элемент массива (т.е. записать i-ый элемент в $(i + 1)$-ый), а в 1-ый элемент записать запомненную цифру.

```
алг Сдвиг()
нач
  цел a, d[100], dn, n, max, i, j

  ввод a
  если a <= 9 то
    вывод "Некорректные исходные данные"
  иначе

  max = a

  n = 0
  пока a > 0
  нц
    n = n + 1
    d[n] = a mod 10
    a = a div 10
  кц

  для i от 1 до n - 1
  нц
    dn = d[n]
    для j от n - 1 до 1 шаг -1
    нц
      d[j + 1] = d[j]
    кц
    d[1] = dn

    a = d1[1]
    для j от 2 до n
    нц
      a = a * 10 + d[j]
    кц
    если a > max то
      max = a
    всё
  кц

  вывод max

  всё
кон
```

Решение (2 способ). Для каждого числа a будем искать наибольшее число, получаемое с помощью сдвига, следующим образом. Прежде всего, надо сформировать массив цифр этого числа путём деления его на 10 и сохранения остатков от деления. При этом самая младшая цифра (с весом 10^0) будет записана в элемент массива с индексом 1, предпоследняя цифра (с весом 10^1) будет записана в элемент массива с индексом 2, и т.д. Также надо подсчитать n – количество цифр числа. Далее в массиве цифр надо найти максимальную цифру числа, а также подсчитать количество вхождений максимальной цифры и запомнить позиции, в которых стоит максимальная цифра. Далее для каждого вхождения максимальной цифры сдвигаем исходное число так, чтобы максимальная цифра оказывалась самой первой (старшей), получаем соответствующее число и находим максимум из полученных чисел.

Для сдвига максимальной цифры в первую позицию надо сделать следующее. Пусть эта цифра стоит в i-ой позиции. Тогда из исходного массива цифр переписываем в новый массив сначала цифры, начиная с i-ой позиции и заканчивая 1-ой позицией,

причём цифру из *i*-ой позиции записываем в *n*-ую позицию, цифру из (*i* – 1)-ой позиции записываем в (*n* – 1)-ую позицию и т.д. Затем дописываем в новый массив цифры, начиная с *n*-ой позиции и заканчивая (*i* + 1)-ой позицией.

```
алг Сдвиг()
нач
  цел a, d[100], d1[100], n, max, dmax, i, j

  ввод a
  если a <= 9 то
    вывод "Некорректные исходные данные"
  иначе

  n = 0
  пока a > 0
  нц
    n = n + 1
    d[n] = a mod 10
    a = a div 10
  кц

  dmax = d[1]
  для i от 1 до n
  нц
    если d[i] > dmax то
      dmax = d[i]
    всё
  кц

  max = 0
  для i от 1 до n
  нц
    если d[i] = dmax то
      для j от i до 1 шаг -1
      нц
        d1[n – i + j] = d[j]
      кц
      для j от n до i + 1 шаг -1
      нц
        d1[j – i] = d[j]
      кц
      a = d1[1]
      для j от 2 до n
      нц
        a = a * 10 + d[j]
      кц
      если a > max то
        max = a
      всё
    всё
  кц

  вывод max

  всё
кон
```

4. Рассмотрим возрастающий ряд всех положительных несократимых правильных дробей, знаменатель которых меньше или равен *n*. Разработайте алгоритм нахождения суммы *P* тех членов данного ряда, для которых знаменатель нацело делится на 21.

Решение. Знаменатели дробей составляют ряд 21, 42, 63 и т.д. пока знаменатель меньше или равен *n*. Для каждого знаменателя *x* перебираем все возможные числители дроби, которые находятся в пределах от 1 до *x* – 1. Если дробь является несократимой, запоминаем её. Дробь является несократимой, если НОД числителя и знаменателя равен 1. Для нахождения НОД используем алгоритм Евклида: из большего числа надо вычитать меньшее, пока числа не станут равными.

Для ускорения работы алгоритма можно использовать не простой, а расширенный алгоритм Евклида, который 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. Значение в правой части уравнения на предпоследнем шаге даёт наибольший общий делитель чисел x и y.

Чтобы сложить запомненные дроби, надо найти НОК всех знаменателей, привести дроби к общему знаменателю, сложить числители и упростить дробь.

НОК нескольких чисел вычисляется через последовательные вычисления НОК пар чисел – НОК(НОК($a_1, a_2, \ldots, a_{n-1}$), a_n). Получить НОК двух чисел можно, воспользовавшись формулой НОК(x, y) = $\dfrac{x \cdot y}{\text{НОД}(x, y)}$.

```
алг СуммаДробей()
нач
  цел n, x, y, num[n * n], den[n * n], k, nok, p, wp, i

  ввод n
  если n <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    k = 0
    для x от 21 до n шаг 21
    нц
      для y от 1 до x - 1
      нц
        если НОД(x, y) = 1 то
          k = k + 1
          num[k] = y
          den[k] = x
        всё
      кц
    кц

    nok = den[1]
    для i от 2 до k
    нц
      nok = nok * den[i] / НОД(nok, den[i])
    кц

    p = 0
    для i от 1 до k
    нц
      p = p + num[i] * nok / den[i]
    кц

    wp = p div nok
    p  = p mod nok

    вывод "Результат = "
    если wp > 0 то
      вывод wp, " "
    всё
    вывод p, " / ", nok
```

```
  всё
кон

алг НОД(арг цел x, y)
нач
  цел q, r, u1, u2, v1, v2, u, v, d

  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

  вернуть x
кон
```

5. Числа Пелля задаются соотношением:

$$P_n = \begin{cases} 0, n = 0 \\ 1, n = 1 \\ 2P_{n-1} + P_{n-2}, n > 1 \end{cases}$$

. Разработайте алгоритм, находящий простые числа Пелля в диапазоне от *P* до *Q*.

Решение. Создадим массив и проинициализируем его первые два элемента значениями 0 и 1. Далее будем вычислять следующие элементы массива – каждый *n*-ый элемент вычисляется по формуле $2P_{n-1} + P_{n-2}$. Прекращаем вычисления, когда очередной элемент массива будет больше или равен *Q*. Далее надо в полученном массиве найти простые числа.

Можно обойтись без массива. Объявляем две переменных и инициализируем их значениями 0 и 1. Далее вычисляем значение третьей переменной по такой же формуле. Пока значение третьей переменной меньше *P*, просто вычисляем эти значения. Далее пока получаемые значения меньше или равны *Q*, проверяем их на простоту. При переходе к следующему шагу цикла надо сместить значения, записав вторую переменную в первую, а третью – во вторую.

Все найденные числа надо проверить на простоту. Можно примитивно проверять делится ли число *x* на какое-либо другое число из диапазона от 2 до $\sqrt{x}$. А можно построить массив простых чисел в диапазоне от 2 до *Q* с помощью решета Эратосфена и проверять каждое число Пелля на попадание в этот массив.

Для использования решета Эратосфена необходимо построить массив чисел в заданном диапазоне от 1 до *Q*. 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 $\sqrt{Q}$, 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*.

```
цел nums[q]

алг ПростыеЧислаПелля()
нач
  цел p, q, p1, p2, p3

  ввод p, q
  если p <= 0 или q <= 0 или p > q то
    вывод "Некорректные исходные данные"
  иначе

    РешетоЭратосфена(q)

    p1 = 0
    p2 = 1
    p3 = 2 * p2 + p1
    пока p3 < p
    нц
      p1 = p2
      p2 = p3
      p3 = 2 * p2 + p1
    кц

    пока p3 <= q
    нц
      если nums[p3] <> 0 то
        вывод p3
      всё
      p1 = p2
      p2 = p3
      p3 = 2 * p2 + p1
    кц

  всё
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
```