ЗАДАНИЕ ПО ИНФОРМАТИКЕ
ВАРИАНТ 73101 для 10 класса

Для заданий 2, 3, 4, 5 требуется разработать алгоритм на языке блок-схем, псевдокоде или естественном языке

1. Утверждения $A \to C$, $A \,\&\, B \to D$, $\neg B \to E$ истинны. Чему равны A и B, если C, D и E ложны?

Решение. Таблица истинности для логической функции «импликация» представлена ниже.

X	Y	$X \to Y$
ложь	ложь	истина
ложь	истина	истина
истина	ложь	ложь
истина	истина	истина

Из таблицы видно, что если следствие ложно, то для того, чтобы вся формула была истинной, необходимо, чтобы посылка также была ложна. Таким образом, утверждение A должно быть ложно, в этом случае $A \,\&\, B$ также будет ложно. Кроме того, ложно должно быть $\neg B$, т.е. утверждение B должно быть истинно.

2. В археологических раскопках в Крыму при строительстве трассы «Таврида» археологи нашли табличку с таким текстом: $\sqrt{7} = 2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \ldots}}}}$

   Далее в формуле структура из 1, 1, 1, 4 повторяется бесконечное число раз. Пожалуйста, проверьте записанное предположение – разработайте алгоритм проверки на ЭВМ с точностью до 0.0001 справедливости этой формулы.

Решение. Данную дробь можно вычислить по формуле $f_1 = 1 / (1 + 1 / (1 + 1 / (1 + 1 / (4 + f_0))))$, где f_0 – отбрасываемая часть, обозначенная многоточием. Положим сначала f_0 равной 0 и вычислим f_1. Если разность по модулю между переменными f_0 и f_1 окажется меньше 0.0001, можно прекращать вычисления. Иначе положим переменную f_0 равной f_1 и снова вычислим f_1 по той же формуле. Итоговый результат равен $2 + f_1$.

Сравнивать необходимо именно два приближённых значения дроби, а не приближённое значение дроби со значением $\sqrt{7}$. Формула может оказаться неправильной, тогда разность между приближённым значением дроби и значением $\sqrt{7}$ никогда не станет меньше требуемой точности.

```
алг Дробь
нач
  вещ f0, f1

  f1 = 0
  повторять
    f0 = f1
    f1 = 1 / (1 + 1 / (1 + 1 / (1 + 1 / (4 + f0))))
  до abs(f0 – f1) < 0.0001

  если abs((2 + f1) – sqrt(7)) < 0.0001 то
    вывод 'Формула верна'
  иначе
    вывод 'Формула не верна'
  всё
кон
```

3. В прямоугольном зале размера $M \times N$ квадратных одинаковых плит двигается робот. Изначально робот находится в указанном углу (Н). Робот за 1 ход может передвигаться по одной из четырёх траекторий, показанных на рисунке. В таблице размера $M \times N$ указано, какие плиты заняты колоннами (Х). Разработайте алгоритм, отвечающий на вопрос: может

ли робот добраться до указанной на рисунке плиты (К). Выход за стены зала, а также проход сквозь колонну не являются возможными.

Траектории движения робота

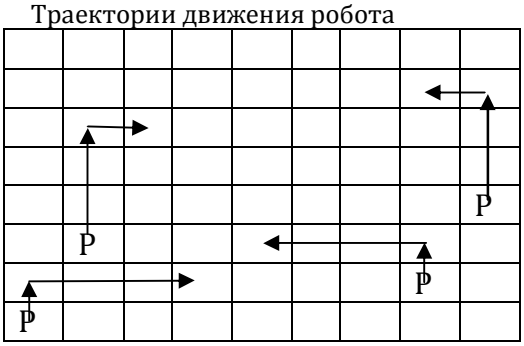
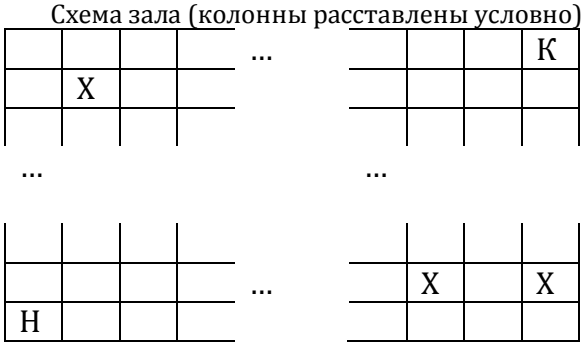


Схема зала (колонны расставлены условно)



Решение. Из каждой точки робот может, в принципе, пойти четырьмя путями. Однако, некоторые пути могут быть заблокированы. Поэтому на каждом шаге надо проверять, может ли робот попасть в каждую из четырёх возможных точек.

Запишем в массив координаты начальной точки. Далее в цикле извлекаем (удаляем) из массива координаты первой находящейся в нём точки, взамен кладём в него координаты точек, куда робот может попасть из этой точки. Таких точек может оказаться от 0 до 4. Если среди них есть конечная точка, значит, робот может туда добраться. Если же робот в принципе не может добраться до конечной точки, то в какой-то момент массив станет пустым, и можно будет прекратить поиск.

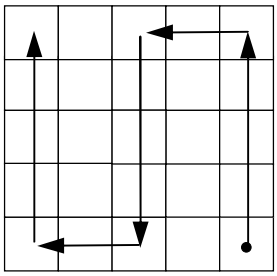
Пусть координаты начальной точки равны (1, 1), а координаты конечной – (*M*, *N*). Пусть также в таблице стоит 0, если плита не занята колонной, и 1, если плита занята колонной. Координаты точек, куда может попасть робот, будем хранить в массивах *x* и *y*, при этом *x*-координата соответствует номеру строки, а *y*-координата – номеру столбца.

```
цел m, n
цел hall[m, n]

алг Робот
нач
  цел x[1000], y[1000]
  цел xc, yc
  цел k, i, j
  лог roborCanReach

  ввод m, n
  для i от 1 до m
  нц
    для j от 1 до n
    нц
      ввод hall[i, j]
    кц
  кц

  k = 1
  x[1] = 1
  y[1] = 1
  roborCanReach = ложь
  пока k <> 0 и не roborCanReach
  нц
    xc = x[1]
    yc = y[1]
    СдвинутьЭлементыМассивовВперёд(x, y, k)
    если РоботМожетПойтиПоПервойТраектории(xc, yc) то
      k = k + 1
      x[k] = xc + 1
      y[k] = yc + 3
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
    если РоботМожетПойтиПоВторойТраектории(xc, yc) то
      k = k + 1
      x[k] = xc + 3
```

```
      y[k] = yc + 1
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
    если РоботМожетПойтиПоТретьейТраектории(xc, yc) то
      k = k + 1
      x[k] = xc - 1
      y[k] = yc + 3
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
    если РоботМожетПойтиПоЧетвёртойТраектории(xc, yc) то
      k = k + 1
      x[k] = xc - 3
      y[k] = yc + 1
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
  кц

  если roborCanReach то
    вывод 'Робот может достичь конечной точки'
  иначе
    вывод 'Робот не может достичь конечной точки'
  всё
кон

алг СдвинутьЭлементыМассивовВперёд(арг рез цел x[1000], алг рез цел y[1000], арг рез цел k)
нач
  цел i

  для i от 2 до k
  нц
    x[i - 1] = x[i]
    y[i - 1] = y[i]
  кц
  k = k - 1
кон

алг РоботМожетПойтиПоПервойТраектории(арг цел x, арг цел y)
нач
  если x <= m - 1 и y <= n - 3 и
       hall[x, y + 1] = 0 и hall[x, y + 2] = 0 и hall[x, y + 3] = 0 и hall[x + 1, y + 3] = 0 то
    вернуть истина
  иначе
    вернуть ложь
  всё
кон

алг РоботМожетПойтиПоВторойТраектории(арг цел x, арг цел y)
нач
  если x <= m - 3 и y <= n - 1 и
       hall[x, y + 1] = 0 и hall[x + 1, y + 1] = 0 и hall[x + 2, y + 1] = 0 и hall[x + 3, y + 1] = 0 то
    вернуть истина
  иначе
    вернуть ложь
  всё
кон

алг РоботМожетПойтиПоТретьейТраектории(арг цел x, арг цел y)
нач
  если x <= m - 3 и y > 1 и
       hall[x + 1, y] = 0 и hall[x + 2, y] = 0 и hall[x + 3, y] = 0 и hall[x + 3, y - 1] = 0 то
    вернуть истина
  иначе
    вернуть ложь
  всё
кон

алг РоботМожетПойтиПоЧетвёртойТраектории(арг цел x, арг цел y)
нач
  если x <= m - 1 и y > 3 и
       hall[x + 1, y] = 0 и hall[x + 1, y - 1] = 0 и hall[x + 1, y - 2] = 0 и hall[x + 1, y - 3] = 0 то
    вернуть истина
```

```
  иначе
    вернуть ложь
  всё
кон
```

4. В поисках яблок шинигами Рюк попал в лабиринт. На стене он увидел изображение схемы лабиринта (квадратная таблица размером $N \times N$). Стрелками на рисунке обозначен путь, по которому можно пройти лабиринт, не попав в ловушку. Если Рюк попадёт в ловушку, то он исчезнет из лабиринта. На каждом шаге при движении по лабиринту Рюку встречаются яблоки с вырезанными на них натуральными числами, которые можно собирать. Помогите шинигами Рюку пройти по лабиринту из правого нижнего угла в левый верхний, подбирая при этом только вкусные яблоки (т.е. те, на которых вырезаны простые числа). Число называется простым, если оно делится только на само себя и на 1.



Решение. При такой схеме лабиринта проход из правого нижнего угла в левый верхний возможен, только если количество столбцов N выражается формулой $4K + 1$, где K – любое число от 0. Поэтому если $N - 1$ не делится нацело на 4, то проход невозможен. Иначе K раз повторяем следующие действия: идём из по текущему столбцу j (j начинается с N) снизу-вверх, затем обрабатываем ячейку с индексами $(1, j - 1)$, затем идём по столбцу $j - 2$ сверху вниз, и, наконец, обрабатываем ячейку с индексами $(N, j - 3)$. Столбец с номером 1 проходится снизу вверх после завершения обработки остальных столбцов. При проходе проверяем числа в обрабатываемых ячеек на простоту. Если число является простым, выводим его.

Чтобы проверить, что число x является простым, необходимо перебрать его возможные делители от 2 до $\sqrt{x}$ и проверить, делится ли число x на какой-либо из возможных делителей. Если это так, то число не является простым. При этом число 2 является простым, а число 1 не является простым.

Для сокращения перебора можно сначала проверить делимость числа x на 2, а потом проверить его делимость на возможные нечётные делители от 3 до $\sqrt{x}$ с шагом 2. Не забываем, что само число 2 является простым.

Можно ещё больше сократить перебор, отбрасывая числа, которые делятся на 2 и на 3. Для этого для числа x проверяются возможные делители i от 5 до $\sqrt{x}$ с шагом 6, но на каждом шаге цикла проверяется делимость числа x на i и на $(i + 2)$, т.е. получается ряд 5, 7, 11, 13 и т.д. До цикла надо проверить делимость числа x на 2 и на 3, а также надо учесть, что сами числа 2 и 3 являются простыми.

```
алг Лабиринт
нач
  цел n
  цел maze[n, n]
  цел k, i, j, p

  ввод n
  если n <= 0 то
    вывод 'Некорректное значение'
  иначе
    если (n - 1) mod 4 <> 0 то
      вывод 'Выход из лабиринта невозможен'
    иначе
      для i от 1 до n
      нц
        для j от 1 до n
        нц
          ввод maze[i, j]
```

```
        кц
      кц

      для j от n до 2 шаг -4
      нц
        для i от n до 1 шаг -1
        нц
         если Простое(maze[i, j]) то
           вывод maze[i, j]
         всё
        кц
       если Простое(maze[1, j - 1]) то
         вывод maze[1, j - 1]
       всё
        для i от 1 до n
        нц
         если Простое(maze[i, j - 2]) то
           вывод maze[i, j - 2]
         всё
        кц
       если Простое(maze[n, j - 3]) то
         вывод maze[n, j - 3]
       всё
      кц
      для i от n до 1 шаг -1
      нц
       если Простое(maze[i, 1]) то
         вывод maze[i, 1]
       всё
      кц
    всё
  всё
кон


алг Простое(арг цел N)
нач
  цел i

  если N <= 1 то
    вернуть ложь
  всё
  если N = 2 или  N = 3 или  N = 5 или  N = 7 то
    вернуть истина
  всё
  если N mod 2 = 0 то
    вернуть ложь
  всё
  если N mod 3 = 0 то
    вернуть ложь
  всё

  для i от 5 до целая_часть(sqrt(N)) шаг 6              // Рассматриваем числа, меньшие корня (!) из N
  нц
    если N mod i = 0 то
      вернуть ложь
    всё
    если N mod (i + 2) = 0 то
      вернуть ложь
    всё
  кц
  вернуть истина

кон
```

5. В таблице размером $N \times 2$ записаны координаты точек на плоскости (x, y). N достаточно велико. Все точки лежат на графике некоторой функции, но их порядок нарушен. Таким образом, функция задана табличным способом. Разработайте алгоритм проверки того, что эта функция является монотонно убывающей.

Решение. Отсортируем точки по x-координате. После этого проверим, что y-координата каждой точки, начиная со второй, не больше y-координаты предыдущей точки.

```
алг Точки
нач
  цел n
  вещ table[n, 2]
```

```
  цел i
  лог decreasing

  ввод n
  если n <= 0 то
    вывод 'Некорректное значение'
  иначе
    для i от 1 до n
    нц
      ввод table[i, 1], table[i, 2]
    кц

    QuickSort(table, 1, n)

    decreasing = истина
    i = 2
    пока i <= n и decreasing
    нц
      если table[i - 1, 2] < table[i, 2] то
        decreasing = ложь
      всё
      i = i + 1
    кц

    если decreasing то
      вывод 'Функция является монотонно убывающей'
    иначе
      вывод 'Функция не является монотонно убывающей'
    всё
  всё
кон


алг QuickSort(арг рез цел x[10000, 2], арг цел n1, n2)
нач
  цел i, j,
  вещ y, k

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

```
    всё
    если i < n2 то
      QuickSort(x, i, n2)
    всё
  всё
кон
```