**Демоверсия для подготовки к олимпиаде по информатике**

**10-11 классы**

## Посчитать количество единичных битов числа

**Формулировка.** Дано натуральное число меньше 16. Посчитать количество его единичных битов. Например, если дано число 9, запись которого в двоичной системе счисления равна 10012 (подстрочная цифра 2 справа от числа означает, что оно записано в двоичной системе счисления), то количество его единичных битов равно 2.

**Решение.** Нам необходима переменная для ввода с клавиатуры. Обозначим ее как **n**. Так как мы должны накапливать количество найденных битов, то возникает потребность в еще одной переменной. Обозначим ее как **count** («*count*» в переводе с англ. означает «считать», «подсчет» и т. д.). Переменные возьмем типа **byte** (они могут принимать значения от 0 до 255), и пусть в данном случае такой объем избыточен, но это не принципиально важно.

Как же сосчитать количество битов во введенном числе? Ведь число же вводится в десятичной системе счисления, и его нужно переводить в двоичную?

На самом деле все гораздо проще. Здесь нам поможет одно интересное правило:

*Остаток от деления любого десятичного числа* ***x*** *на число* ***p*** *дает нам разряд единиц числа* ***x*** *(его крайний разряд справа) в системе счисления с основанием* ***p****.*

То есть, деля некоторое десятичное число, например, на 10, в остатке мы получаем разряд единиц этого числа в системе счисления с основанием 10. Возьмем, например, число 3468. Остаток от деления его на 10 равен 8, то есть разряду единиц этого числа.

Понятно, что такие же правила господствуют и в арифметике в других системах счисления, и в том числе в двоичной системе. Предлагаю поэкспериментировать: запишите на бумаге десятичное число, затем, используя любой калькулятор с функцией перевода из одной системы счисления в другую, переведите это число в двоичную систему счисления и также запишите результат. Затем разделите исходное число на 2 и снова переведите в двоичную систему. Как оно изменилось в результате? Вполне очевидно, что у него пропал крайний разряд справа, или, как мы уже говорили ранее, разряд единиц.

Но как это использовать для решения задачи? Воспользуемся тем, что в двоичной записи числа нет цифр, кроме 0 и 1. Легко убедиться в том, что сложив все разряды двоичного числа, мы получаем как раз таки количество его единичных битов. Это значит, что вместо проверки значений разрядов двоичного представления числа мы можем прибавлять к счетчику сами эти разряды – если в разряде был 0, значение счетчика не изменится, а если 1, то повысится на единицу.

Теперь, резюмируя вышеприведенный итог, можно поэтапно сформировать сам алгоритм:

1. Вводим число **n**;
2. Обнуляем счетчик разрядов **count**. Это делается потому, что значения всех переменных при запуске программы считаются неопределенными, и хотя в большинстве компиляторов **Pascal** они обнуляются при запуске, все же считается признаком «хорошего тона» в программировании обнулить значение переменной, которая будет изменяться в процессе работы без предварительного присваивания ей какого-либо значения.
3. Прибавляем к **count** разряд единиц в двоичной записи числа **n**, то есть остаток от деления **n** на 2:

count := count + n mod 2;

Строго говоря, мы могли бы не прибавлять предыдущее значение переменной **count** к остатку от деления, так как оно все равно было нулевым. Но мы поступили так для того, чтобы сделать код более однородным, далее это будет видно. Учтя разряд единиц в двоичной записи **n**, мы должны отбросить его, чтобы исследовать число далее. Для этого разделим **n** на 2. На языке **Pascal** это будет выглядеть так:

n := n div 2;

1. Теперь нам нужно еще два раза повторить **п. 3** , после чего останется единственный двоичный разряд числа **n**, который можно просто прибавить к счетчику без каких-либо дополнений:

count := count + n;

1. В результате в переменной **count** будет храниться количество единичных разрядов в двоичной записи исходного числа. Осталось лишь вывести ее на экран.

**Код:**

|  |
| --- |
| 1. program BinaryUnits;
2. var
3. n, count: byte;
4. begin
5. readln(n);
6. count := 0;
7. count := count + n mod 2;
8. n := n div 2;
9. count := count + n mod 2;
10. n := n div 2;
11. count := count + n mod 2;
12. n := n div 2;
13. count := count + n;
14. writeln(count)
15. end.
 |

## Проверить, является ли четырехзначное число счастливым билетом

**Формулировка.** Дано четырехзначное число. Проверить, является ли оно «счастливым билетом».

Примечание: счастливым билетом называется число, в котором: а) при четном количестве цифр в числе сумма цифр его левой половины равна сумме цифр его правой половины; б) при нечетном количестве цифр – то же самое, но с отбрасыванием серединной цифры. Например, рассмотрим число 1322. Его левая половина равна 13, а правая – 22, и оно является счастливым билетом (т. к. 1 + 3 = 2 + 2). Аналогично: 1735 (1 + 7 = 3 + 5), 1111 (1 + 1 = 1 + 1) и т. д.

Примеры других счастливых билетов за рамками условия текущей задачи: 7 (отбросили единственную цифру), 39466 (3 + 9 = 6 + 6, а 4 отбросили), 11 (1 = 1), и т. д.

**Решение.** Для ввода достаточно одной переменной **n** типа **word**. Все, что необходимо сделать для решения – это последовательно получить все разряды исходного числа, причем из двух младших разрядов (единиц и десятков) сформировать первую сумму, а из двух старших разрядов – вторую, после чего вывести на экран результат булевского выражения равенства полученных сумм. Первую сумму будем хранить в переменной **right**, а вторую – в переменной **left** (выбрано по расположению цифр в записи числа). Значение обоих из них не может превосходить 18 (т. к. для наибольшего допустимого числа 9999 обе суммы равны 9 + 9 = 18), поэтому для их описания используем тип **byte**.

Более детальный разбор алгоритма:

1. Вводим число **n**;
2. Присваиваем переменной **right** значение последней цифры числа **n**, потом отбрасываем эту цифру, затем повторяем то же самое, но на этот раз уже прибавляем добытую цифру к прежнему значению **right**:

right := n mod 10;

n := n div 10;

right := right + n mod 10;

n := n div 10;

1. Присваиваем переменной **left** последнюю цифру **n**, отбрасываем ее и прибавляем к **right** единственную оставшуюся в переменной **n** цифру:

left := n mod 10;

n := n div 10;

left := left + n;

1. Выводим на экран результат сравнения накопленных сумм:

writeln(left = right);

**Код:**

|  |
| --- |
| 1. program HappyTicket;
2. var
3. n: word;
4. left, right: byte;
5. begin
6. readln(n);
7. right := n mod 10;
8. n := n div 10;
9. right := right + n mod 10;
10. n := n div 10;
11. left := n mod 10;
12. n := n div 10;
13. left := left + n;
14. writeln(left = right)
15. end.
 |

## Найти наибольший нетривиальный делитель натурального числа

**Формулировка.** Дано натуральное число. Найти его наибольший нетривиальный делитель или вывести единицу, если такового нет.

Примечание 1: делителем натурального числа **a** называется натуральное число **b**, на которое **a** делится без остатка. То есть выражение «**b** – делитель **a**» означает: **a / b = k**, причем **k** – натуральное число.

Примечание: нетривиальным делителем называется делитель, который отличен от 1 и от самого числа (так как на единицу и само на себя делится любое натуральное число).

**Решение.** Пусть ввод с клавиатуры осуществляется в переменную **n**. Попробуем решить задачу перебором чисел. Для этого возьмем число на единицу меньшее **n** и проверим, делится ли **n** на него. Если да, то выводим результат и выходим из цикла с помощью оператора **break**. Если нет, то снова уменьшаем число на 1 и продолжаем проверку. Если у числа нет нетривиальных делителей, то на каком-то шаге проверка дойдет до единицы, на которую число гарантированно поделится, после чего будет выдан соответствующий условию ответ.

Хотя, если говорить точнее, следовало бы начать проверку с числа, равного **n div 2** (чтобы отбросить дробную часть при делении, если **n** нечетно), так как ни одно натуральное число не имеет делителей больших, чем половина это этого числа. В противном случае частное от деления должно быть натуральным числом между 1 и 2, которого просто не существует.

Данная задача также решается через **for**, но через другую его разновидность, и теперь счетчик будет убывать от **n div 2** до 1. Для этого **do** заменится на **downto**, при позиции начального и конечного значений остаются теми же.

Алгоритм на естественном языке:

1. Ввод **n**;
2. Запуск цикла, при котором **i** изменяется от **n div 2** до 1. В цикле:
	* + 1. Если **n** делится на **i** (то есть, остаток от деления числа **n** на **i** равен 0), то выводим **i** на экран и выходим из цикла с помощью **break**.

**Код:**

|  |
| --- |
| 1. program GreatestDiv;
2. var
3. i, n: word;
4. begin
5. readln(n);
6. for i := n div 2 downto 1 do begin
7. if n mod i = 0 then begin
8. writeln(i);
9. break
10. end
11. end
12. end.
 |

Кстати, у оператора ветвления **if** в цикле отсутствует **else**-блок. Такой условный оператор называется оператором ветвления с одной ветвью.

## Найти наименьший нетривиальный делитель натурального числа

**Формулировка.** Дано натуральное число. Найти его наименьший нетривиальный делитель или вывести само это число, если такового нет.

**Решение.** Задача решается аналогично предыдущей. При этом необходимо начать обычный цикл с увеличением, при котором переменная цикла **i** изменяется от 2 до **n** (такая верхняя граница нужна для того, чтобы цикл всегда заканчивался, так как когда **i** будет равно **n**, выполнится условие **n mod i = 0**). Весь остальной код при этом не отличается.

**Код:**

|  |
| --- |
| 1. program SmallestDiv;
2. var
3. i, n: word;
4. begin
5. readln(n);
6. for i := 2 to n do begin
7. if n mod i = 0 then begin
8. writeln(i);
9. break
10. end
11. end
12. end.
 |

## Подсчитать общее число делителей натурального числа

**Формулировка.** Дано натуральное число. Подсчитать общее количество его делителей.

**Решение.** Задача достаточно похожа на две предыдущие. В ней также необходимо провести перебор в цикле некоторого количества натуральных чисел на предмет обнаружения делителей **n**, но при этом необходимо найти не первый из них с какого-либо конца отрезка [1, **n**] (это отрезок, содержащий все числа от 1 до **n** включительно), а посчитать их. Это можно сделать с помощью счетчика **count**, который нужно обнулить непосредственно перед входом в цикл. Затем в условном операторе **if** в случае истинности условия делимости числа **n** (**n mod i = 0**) нужно увеличивать счетчик **count** на единицу (это удобно делать с помощью оператора **inc**).

Алгоритм на естественном языке:

1. Ввод **n**;
2. Обнуление переменной **count** (в силу необходимости работать с ее значением без предварительного присваивания ей какого-либо числа)
3. Запуск цикла, при котором **i** изменяется от 1до **n**. В цикле:
	* + 1. Если **n** делится на **i** (то есть, остаток от деления числа **n** на **i** равен 0), то увеличиваем значение переменной **count** на 1;
4. Вывод на экран значения переменной **count**.

**Код:**

|  |
| --- |
| 1. program CountDiv;
2. var
3. i, n, count: word;
4. begin
5. readln(n);
6. count := 0;
7. for i := 1 to n do begin
8. if n mod i = 0 then inc(count)
9. end;
10. writeln(count)
11. end.
 |

## Вывести на экран все простые числа до заданного

**Формулировка.** Дано натуральное число. Вывести на экран все простые числа до заданного включительно.

**Решение.** В решении данной задачи используется решение предыдущей.

Нам необходимо произвести тест простоты числа, который был описан в решении предыдущей задачи следующим кодом (обозначим его как **код 1**):

count := 0;

for i := 1 to n do begin

 if n mod i = 0 then inc(count)

end;

writeln(count = 2);

Здесь **n** – проверяемое на простоту число. Напомним, что данный фрагмент кода в цикле проверяет, делится ли **n** на каждое **i** в отрезке от 1 до самого **n**, и если **n** делится на **i**, то увеличиваетсчетчик **count** на 1. Когда цикл заканчивается, на экран выводится результат проверки равенства счетчика числу 2.

В нашем же случае нужно провести проверку на простоту всех натуральных чисел от 1 до заданного числа (обозначим его как **n**). Следовательно, мы должны поместить **код 1** в цикл по всем **k** от 1 до **n**. Также в связи с этим необходимо заменить в **коде 1** идентификатор **n** на **k**, так как в данном решении проверяются на простоту все числа **k**. Кроме того, теперь вместо вывода ответа о подтверждении/опровержении простоты **k** необходимо вывести само это число в случае простоты:

if count = 2 then write(k, ' ');

Обобщая вышесказанное, приведем алгоритм на естественном языке:

1. Ввод **n**;
2. Запуск цикла, при котором **k** изменяется от 1до **n**. В цикле:
	* + 1. Обнуление переменной **count**;
			2. Запуск цикла, при котором **i** изменяется от 1 до **k**. В цикле:
			3. Если **k** делится на **i**, то увеличиваем значение переменной **count** на 1;
			4. Если **count** = 2, выводим на экран число **k** и символ пробела.

**Код:**

|  |
| --- |
| 1. program PrimesToN;
2. var
3. i, k, n, count: word;
4. begin
5. readln(n);
6. for k := 1 to n do begin
7. count := 0;
8. for i := 1 to k do begin
9. if k mod i = 0 then inc(count)
10. end;
11. if count = 2 then write(k, ' ')
12. end
13. end.
 |

**Вычислительная сложность.** В данной задаче мы впервые столкнулись с вложенным циклом (когда один цикл запускается внутри другого). Это означает, что наш алгоритм имеет нелинейную сложность (при которой количество выполненных операций равно размерности исходных данных (в нашем случае это **n** – количество чисел) плюс некоторое количество обязательных операторов).

Давайте посчитаем количество выполненных операций в частном случае. Итак, пусть необходимо вывести все натуральные простые числа до числа 5. Очевидно, что проверка числа 1 пройдет в 1 + 2 шага, числа 2 – в 2 + 2 шага, числа 3 – в 3 + 2 шага и т. д. (прибавленная двойка к каждому числу – два обязательных оператора вне внутреннего цикла), так как мы проверяем делители во всем отрезке от 1 до проверяемого числа. В итоге количество проведенных операций (выполненных операторов на низшем уровне) представляет собой сумму: 3 + 4 + 5 + 6 + 7 (также опущен обязательный оператор ввода вне главного (внешнего) цикла). Очевидно, что при выводе всех простых чисел от 1 до **n** приведенная ранее сумма будет такой:

1 + 3 + 5 + 6 + ... + (**n** – 1) + **n** + (**n** + 1) + (**n** + 2),

где вместо многоточия нужно дописать все недостающие члены суммы. Очевидно, что это сумма первых (**n** + 2) членов арифметической прогрессии с вычтенным из нее числом 2.

Как известно, сумма **k** первых членов арифметической прогрессии выражена формулой:

,

где **a1** – первый член прогрессии, **ak** – последний.

Подставляя в эту формулу наши исходные данные и учитывая недостающее до полной суммы число 2, получаем следующее выражение:



Чтобы найти асимптотическую сложность алгоритма, отбросим коэффициенты при переменных и слагаемые с низшими степенями (оставив, соответственно, слагаемое с самой высокой степенью). При этом у нас остается член ***n2***, значит, асимптотическая сложность алгоритма – **O(n2)**.

Конечно, в дальнейшем мы не будем так подробно находить асимптотическую сложность алгоритмов, а тем более, вычислять количество требуемых операций, что интересно только теоретически. На самом деле, конечно, нас интересует лишь порядок роста времени работы алгоритма (количества необходимых операций), который можно выявить из анализа вложенности циклов и некоторых других характеристик.

1. **Вычисление суммы и произведения элементов массива, удовлетворяющих заданному условию.**

Дан целочисленный одномерный массив, состоящий из n элементов. Найти сумму и произведение нечетных элементов, кратных 3.

**Введем обозначения**: n – количество элементов в массиве; А – имя массива; i – индекс элемента массива; Ai – i-й элемент массива A; s – сумма нечетных элементов массива, кратных 3; p – произведение нечетных элементов массива, кратных 3.

**Входные данные: n, A.**

**Выходные данные: s, p.**

Первоначально сумма искомых элементов равна нулю: не просуммировано ни одно слагаемое, то есть s:=0. Далее, используя любой оператор цикла, просматриваем весь массив от первого и до последнего элемента. И если при этом элемент массива нечетный и кратен 3, то к уже накопленной сумме добавляется очередное слагаемое, т.е. s:= s + A[i]. Здесь слева и справа от знака присваивания записано имя одной и той же переменной s, именно это обеспечивает постепенное накопление суммы: s справа – уже вычисленное известное значение суммы, s  - ее новое, вычисляемое значение.

При просмотре массива можно сразу вычислить и произведение элементов массива, удовлетворяющих заданному условию. Произведение вычисляется с помощью оператора p:=p\*A[i]. При этом p справа и p слева имеют разные значения: p справа – уже известное, вычисленное ранее значение произведения, p слева – новое, вычисляемое его значение. Первоначально искомое произведение равно единице, т.е. p:=1.

При решении этой задачи можно использовать любой из видов циклов. Рассмотрим несколько вариантов решения задачи.

**Первый способ**.  Для решения используется цикл с параметром:

**Program** Primer1\_1;

**Var** A: **Array**[1..20] **Of** Integer;

      i, n, s, p: Integer;

**Begin**

     Write (‘n=’); Readln (n);

     **For** i:=1 **To** n **Do** Readln (A[i]);   {ввод массива}

     s:= 0;  p:=1;

     **Fo**r i:=1 **To** n **Do**   {обработка массива}

      **If** (A[i] **mod** 2 <>0) **and** (A[i] **mod** 3 = 0) **Then**

      **Begin**

          s:=s+A[i];  p:= p\*A[i]

      **End;**

      Writeln (‘s=’, s, ‘p=’, p);

      Readln

      **End.**

**Второй способ**. Для решения используется цикл с предусловием:

**Program** Primer1\_2;

**Var** A: **Array**[1..20] **Of** Integer;

      i, n, s, p: Integer;

**Begin**

     Write (‘n=’); Readln (n); i:=1;

     **While** i <= n **Do**

      **Begin**

      Readln (A[i]);   i:= i + 1

      **End;**

     s:= 0;  p:=1; i:=1;

     **While** i<=n  **Do**

     **Begin**

             **If** (A[i] **mod** 2 <>0) **and** (A[i] **mod** 3 = 0) **Then**

**Begin**

          s:=s+A[i]; p:= p\*A[i]

  **End;**  i:= i + 1

**End;**

      Writeln (‘s=’, s, ‘p=’, p);

      Readln

      **End.**

1. **Нахождение количества элементов, удовлетворяющих заданному условию**

Дан массив целых чисел. Найти количество тех элементов, значения которых положительны и не превосходят заданного натурального числа А. (Презентация)

Введем обозначения***:*** n – количество элементов в массиве; X – имя массива; i – индекс элемента массива; Xi – i-й элемент массива X; А – заданное число; k – количество элементов, значения которых положительны и не превосходят заданного числа А.

**Входные данные:** n, X, A.

**Выходные данные:** k.

Вводим с клавиатуры значение числа А. Количество элементов, значения которых положительны и не превосходят заданного числа А, вначале полагаем равным нулю, то есть k:=0. Если очередной элемент массива положителен и не превосходят заданного числа A, то количество таких элементов нужно увеличить на единицу, то есть k:=k + 1. Таким образом, обрабатываются все элементы массива.

При решении этой задачи можно использовать любой из видов циклов. Рассмотрим несколько вариантов решения задачи.

**Первый способ**.  Для решения используется цикл с параметром:

**Program** Primer2\_1;

**Var** X: **Array**[1..20] **Of** Integer;

      i, n, k, A: Integer;

**Begin**

     Write (‘n=’); Readln (n);

     **For** i:=1 **To** n **Do** Readln (X[i]);   {ввод массива}

      Write (‘A=’);  Readln (A); k:= 0;

     **Fo**r i:=1 **To** n **Do**   {обработка массива}

      **If** (X[i] >0) **and** (X[i] <= A) **Then** k:=k + 1;

      Writeln (‘k=’, k);

      Readln

      **End.**

**Второй способ.** Для решения используется цикл с постусловием:

**Program** Primer2\_2;

**Var** X: **Array**[1..20] **Of** Integer;

      i, n, k, A: Integer;

**Begin**

     Write (‘n=’); Readln (n); i:=1;

**Repeat**

           Read (X[i]); i := i + 1

     **Until** i > n;

      Write (‘A=’); Readln (A); k:= 0; i:=1;

     **Repeat**

      **If** (X[i] >0) **and** (X[i] <= A) **Then**

          k:=k + 1; i := i + 1

     **Until** I > n;

      Writeln (‘k=’, k);

      Readln

      **End.**

1. **Нахождение номеров элементов, обладающих заданным свойством**

Найти номера четных элементов, стоящих на нечетных местах. (Презентация)

**Введем обозначения**: n – количество элементов в массиве; X – имя массива; i – индекс элемента массива; Xi – i-й элемент массива Х.

**Входные данные:** n, X.

**Выходные данные:** i.

Необходимо просмотреть весь массив. Если просматриваемый элемент является четным, а его порядковый номер – нечетный, то вывести его номер. При решении этой задачи можно использовать любой из видов циклов.

Составим программу с помощью цикла с параметром:

**Program** Primer3\_1;

**Var** X: **Array**[1..20] **Of** Integer;

      i, n:  Integer;

**Begin**

     Write (‘n=’); Readln (n);

     **For** i:=1 **To** n **Do** Readln (X[i]);   {ввод массива}

      **Fo**r i:=1 **To** n **Do**   {обработка массива}

      **If** (X[i] **mod** 2 = 0) **and** (i **mod** 2<>0) **Then** Write (i:5);

      Readln

      **End.**

Найти номер последнего отрицательного элемента массива.

***Введем обозначения***: n – количество элементов в массиве; А – имя массива; i – индекс элемента массива; Ai – i-й элемент массива A; m – номер последнего отрицательного элемента массива.

***Входные данные:*** n, A.

***Выходные данные:*** m.

Последний отрицательный элемент массива – это первый отрицательный элемент, который встретится при просмотре массива с конца. Если очередной элемент не является отрицательным, то нужно уменьшать значение текущего индекса (i:=i-1), пока он не станет меньше номера первого элемента или не будет найден отрицательный элемент. Переменная m получает значение i (номер отрицательного элемента), т.е. m := i. Если в массиве нет отрицательного элемента, то значение переменной m остается равным нулю.

**Program** Primer3\_2;

**Const** n=10;

**Var** A: **Array**[1..n] **Of** Integer;

      i, m: Integer;

**Begin**

      **For** i:=1 **To** n **Do** Readln (A[i]);   {ввод массива}

      m := 0;  i:=n;

      While  (i >= 1) **and** (A[i] >=0) **Do** i:=i-1;    m:=i;

       Writeln (‘m=’, m);

      Readln

      **End.**

1. **Требуется заполнить массив** 10 целыми числами ( случайно ) от -10 до 10 и вывести его на экран. После чего изменить его таким образом , чтобы числа находились в нем в порядке возрастания. Вывести исходный массив на экран.

## Решение задачи :

program mypascal;
uses crt;
var
a: array[1..10] of integer;
i,i1,c:integer;
begin

randomize;

for i:= 1 to 10 do
begin
a[i]:= random(21)-10;
write(a[i],' ');
end;

writeln(' ');

for i:= 1 to 9 do
for i1:= i+1 to 10 do
if ( a[i1] < a[i] ) then
begin
c:= a[i];
a[i]:= a[i1];
a[i1]:= c;
end;

for i:= 1 to 10 do
write(a[i],' ');

end.

## Разбор задачи :

В начале программы заполняем массив 10 случайными числами от -10 до 10. В этом же цикле выводим массив на экран.

Далее идут 2 цикла вложенных друг в друга. Для того , чтобы отсортировать последовательность чисел :

Первый цикл берет одно число , а второй сравнивает его с остальными числами.

Первое число сравнивается со 2,3,4,5... числом. Второе число с 3,4,5... пока предпоследнее число в последовательности не будет сравниваться с последним.

Сравнение происходит так : если одно число меньше другого , то обмениваем их.