П Р И Л О Ж Е Н И Е

З А Д А Ч И П О П Р О Г Р А М М И Р О В А Н И Ю

1. О д н о м е р н ы е м а с с и в ы

1.0. Общие замечания.

Задачи, приведенные в приложении, являются дополнением к основному материалу учебного пособия. Для сокращения текста программ ввод исходных данных и вывод результатов, как правило, заменяется фразой на русском языке, но при этом обязательно указывается, что именно вводится и выводится в программе.

Для одномерных массивов ввод-вывод подробно рассмотрен в разделе "Ввод-вывод одномерного массива".

При рассмотрении задач по одномерным массивам обязательным условием является предварительное изучение материала, изложенного в основных разделах пособия, в частности, в разделах "Примеры обработки одномерных массивов", "Вставка элементов в упорядоченный массив", "Удаление элементов из массива", "Поиск однородной группы в массиве".

1.1. Элементы, расположенные в массиве X с индекса *k1* по индекс *k2*, переставить в обратном порядке ($1 < k1 < n,\ 1 < k2 < n,\ k1 \le k2$).

Методика решения задачи рассмотрена в разделе «Примеры обработки одномерных массивов».

```
Program Task101;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of real;
Var  i,j,n,k1,k2 : integer;
     R : real;
     X : Ar;
Begin
  Ввод и печать  n, X, k1, k2
  i:=k1; j:=k2;
  While i<j do
    Begin
      R:=x[i]; x[i]:=x[j]; x[j]:=R;
      Inc(i); Dec(j);
    End;
  Печать массива X
End.
```

1.2. Удалить из массива X элементы, расположенные с индекса *k1* по индекс *k2* ($1 < k1 < n,\ 1 < k2 < n,\ k1 \le k2$).

```
Program Task102;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of real;
Var  i,n,k,k1,k2 : integer;
     X : Ar;
Begin
  Ввод и печать  n, X, k1, k2
```

```
    k:=k2-k1+1;                    { кол-во удаляемых элементов }
    For i:=k1 to n-k do
      x[i]:=x[i+k];
    Dec(n,k);
  Печать массива X
  End.
```

1.3. В массиве *X* найти значение и положение последнего нечетного элемента.

Четными или нечетными могут быть лишь целые числа (типы *shortint, byte, integer, word, longint*), по отношению к вещественным числам понятие четности не имеет смысла.

Методы проверки четности целочисленных переменных рассмотрены в разделе "Вычисление степенной функции".

В программе Task103 просмотр массива *X* выполняется справа налево. Как только обнаружен нечетный элемент, запоминаются его значение и индекс, после чего производится выход из цикла просмотра.

До начала цикла переменной *NotEvenPos* присвоено нулевое значение. Если после его окончания значение *NotEvenPos* не изменилось, то это свидетельствует о том, что в массиве *X* не обнаружено нечетных элементов.

```
Program Task103;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,n,
     NotEvenPos : word;  { индекс посл.нечетного элемента }
     NotEven : integer;  { значение посл.нечетного элемента }
     X : Ar;
Begin
  В в о д   n, X
  NotEvenPos:=0;
  For i:=n downto 1 do
    If odd(x[i]) then
      Begin
        NotEvenPos:=i; NotEven:=x[i];
        Break
      End;
  If NotEvenPos=0 then
    Writeln('В массиве X нет нечетных элементов')
  Else
    Writeln('NotEven=',NotEven,'  NotEvenPos=',NotEvenPos);
End.
```

1.4. В массиве *X* обменять местами первый и последний четные элементы.

В программе Task104 вначале производится поиск первого четного элемента. Если он обнаружен (*EvenPos1* > 0), то выполняется поиск последнего четного элемента. Обмен осуществляется, если найден последний четный элемент (*EvenPos2* > 0) и если массив *X* содержит не менее двух таких элементов (*EvenPos1* ≠ *EvenPos2*).

Обмен двух элементов в массиве *X* можно было бы произвести тремя операторами присваивания по обычному методу треугольника:

```
k:=x[EvenPos1]; x[EvenPos1]:=x[EvenPos2]; x[EvenPos2]:=k,
```

где *k* - переменная типа *integer*. Однако в программе Task104 такой обмен выполняется двумя операторами, так как к моменту обмена известны не только индексы обмениваемых элементов, но и их значения.

Примечание. Производить обмен заданных четных элементов операторами

```
k := Even1; Even1 := Even2; Even1 := k
```

было бы неправильно, так как при этом обмениваются значения переменных *Even1* и *Even2*, а не значения элементов массива *X* с индексами *EvenPos1* и *EvenPos2*.

```
Program Task104;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,n,EvenPos1,EvenPos2 : word;
     Even1,Even2 : integer;
     X : Ar;
Begin
  В в о д   n, X
  EvenPos1:=0; EvenPos2:=0;
  For i:=1 to n do              { Определение значения и }
    If not odd(x[i]) then       { положения первого чет- }
      Begin                     { ного элемента          }
        EvenPos1:=i; Even1:=x[i];
        Break
      End;
  If EvenPos1>0 then
    Begin
      For i:=n downto 1 do      { Определение значения и }
        If not odd(x[i]) then   { положения последнего   }
          Begin                 { четного элемента       }
            EvenPos2:=i; Even2:=x[i];
            Break
          End;
      If (EvenPos2>0) and (EvenPos1<>EvenPos2) then
        Begin                   { Обмен элементов массива }
          x[EvenPos1]:=Even2; x[EvenPos2]:=Even1;
        End;
    End;
  П е ч а т ь   м а с с и в а   X
End.
```

1.5. Из целочисленного массива *X* удалить все нечетные элементы, кроме последних двух таких элементов.

В программе Task105 просмотр массива осуществляется справа налево. Сигнал об удалении элемента формируется счетчиком *Count*: элемент удаляется, если *Count* > 2.

```
Program Task105;
Const  Nmax = 200;
Type  Ar = array[1..Nmax] of integer;
Var  i,j,n,Count : byte;
     X : Ar;
Begin
  В в о д   n, X
  Count:=0;
  For i:=n downto 1 do
    If odd(x[i]) then
      Begin
```

```
        Inc(Count);
        If Count>2 then
          Begin
            For j:=i to n-1 do
              x[j]:=x[j+1];
            Dec(n);
          End;
      End;
  П е ч а т ь  n,  X
End.
```

1.6. При однократном просмотре массива *X* выбрать из него три наибольших элемента.

Будем считать, что $x_{\max_1} \geq x_{\max_2} \geq x_{\max_3}$.

```
Program Task106a;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of real;
Var   i,n : word;
      Xmax1,Xmax2,Xmax3,R : real;
      X : Ar;
Begin
  В в о д   n,X
  If n>2 then
    Begin
      Xmax1:=x[1]; Xmax2:=x[1]; Xmax3:=x[1];
      For i:=2 to n do
        If x[i]>Xmax1 then
          Begin
            Xmax3:=Xmax2; Xmax2:=Xmax1; Xmax1:=x[i]
          End
        Else
          If x[i]>Xmax2 then
            Begin
              Xmax3:=Xmax2; Xmax2:=x[i]
            End
          Else
            If x[i]>Xmax3 then
              Xmax3:=x[i];
      П е ч а т ь  Xmax1, Xmax2, Xmax3
    End
  Else
    Writeln('В массиве меньше трех элементов');
End.
```

Пусть массив *X* имеет вид:

$$X = (200, 100, 150, 170, 60, 10).$$

В этом случае по программе Task106a будет получен неверный результат:

```
Xmax1:=200; Xmax2:=200; Xmax3:=200.
```

Внесем следующее изменение в программу:

```
Xmax1:=x[1]; Xmax2:=x[2]; Xmax3:=x[3].
```

Будем считать теперь, что массив *X* заполнен следующими элементами:

$$X = (50, 100, 200, 300, 60, 10).$$

В этом случае также будет выдан неправильный результат:

```
Xmax1:=300; Xmax2:=60; Xmax3:=50.
```

Вывод. Начальные значения *Xmax1, Xmax2 и Xmax3,* равные соответственно *x[1]*, *x[2]* и *x[3]*, должны быть предварительно сгруппированы по убыванию.

```
Program Task106b;
Const  Nmax = 500;
Type   Ar = array[1..Nmax] of real;
Var    i,n : word;
       Xmax1,Xmax2,Xmax3 : real;
       X : Ar;
{ --------------------------------- }
Procedure Swap(Var x1,x2:real);
Var  R : real;
Begin
  If x1<x2 then
    Begin
      R:=x1; x1:=x2; x2:=R
    End;
End { Swap };
{ --------------------------------- }
Begin
  В в о д   n,X
  If n>2 then
    Begin
      Xmax1:=x[1]; Xmax2:=x[2]; Xmax3:=x[3];
      Swap(Xmax1,Xmax2);
      Swap(Xmax1,Xmax3);
      Swap(Xmax2,Xmax3);
      For i:=4 to n do
        If x[i]>Xmax1 then
          Begin
            Xmax3:=Xmax2; Xmax2:=Xmax1; Xmax1:=x[i]
          End
        Else
          If x[i]>Xmax2 then
            Begin
              Xmax3:=Xmax2; Xmax2:=x[i]
            End
          Else
            If x[i]>Xmax3 then
              Xmax3:=x[i];
      П е ч а т ь  Xmax1, Xmax2, Xmax3
    End
  Else
    Writeln('В массиве меньше трех элементов');
End.
```

Примечание. В рассматриваемой задаче можно было бы предварительно сгруппировать по убыванию весь массив *X*, а затем взять его первые три элемента. Однако при этом был бы испорчен исходный массив, что в общем случае не допускается. Если же для группировки использовать буферный массив, то будет нарушен принцип эффективности программы по затратам памяти. К тому же при этом не соблюдается требование однократного просмотра массива.

1.7. В начале массива *X* расположить в исходном относительном порядке все положительные, затем - все нулевые, а потом - все отрицательные элементы массива.

В программе Task107 используются два буферных массива: в массив *Y* по мере просмотра исходного массива *X* записываются положительные элементы, в массив *Z* - отрицательные. Количество элементов в массиве *Y* равно *j*, в массиве *Z* - *k*. После этого в массив *X* переписывается содержимое массива *Y*, затем дописываются нулевые элементы, количество которых равно $m = n - j - k$, и на последнем этапе переписывается массив *Z*.

```
Program Task107;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of real;
Var  i,j,k,m,n : word;
     X,Y,Z : Ar;
Begin
  В в о д   n, X
  j:=0; k:=0;
  For i:=1 to n do       { Перенос в массив Y }
    If x[i]>0 then       { положительных элементов }
      Begin
        Inc(j); y[j]:=x[i];
      End
    Else
      If x[i]<0 then     { Перенос в массив Z }
        Begin            { отрицательных элементов }
          Inc(k); z[k]:=x[i]
        End;
  m:=n-j-k;
  For i:=1 to j do       { Запись в массив X }
    x[i]:=y[i];          { положительных элементов }
  For i:=j+1 to j+m do   { Запись нулей }
    x[i]:=0;             { в массив X }
  For i:=j+m+1 to n do   { Запись в массив X }
    x[i]:=z[i-j-m];      { отрицательных элементов }
  П е ч а т ь  X
End.
```

1.8. В начале массива *X* расположить в исходном относительном порядке все положительные, затем - все отрицательные, а потом - все нулевые элементы данного массива. Буферные массивы не использовать.

Запрет использования буферных массивов существенно отличает реализацию данной задачи от программы Task107. На первом этапе работы программы Task108 производится перестановка положительных элементов в начало массива *X* с сохранением их исходного относительного порядка. Просмотр элементов производится в цикле ***While,*** начиная с индекса *i*=1. Если *i*-ый элемент положительный, то производится переход к следующему элементу x_{i+1}; в противном случае с помощью функции *SearchePos* выполняется поиск ближайшего положительного элемента x_j, после чего подмассив x_i .. x_{j-1} сдвигается на один элемент вправо, а элементу x_i присваивается значение x_j. Цикл ***While*** прекращает свою работу в двух случаях:

- в цикле проанализирован последний элемент с индексом $i = n$;
- начиная с индекса $i + 1$, в массиве не обнаружено положительных элементов.

В дальнейшем из подмассива x_{i+1} .. x_n удаляются нулевые элементы, их количество формируется в счетчике *CountZero*. На заключительном этапе последним элементам массива (их количество равно *CountZero*) присваиваются нулевые значения.

```
Program Task108;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,j,k,n,CountZero : word;
     R : integer;
     CondPos : boolean;
     X : Ar;
{ ------------------------------------ }
Function SearchePos(k:word):word;
{ Поиск ближайшего положит.элемента, начиная с индекса k }
Var  i : word;
Begin
  SearchePos:=0;
  For i:=k to n do
    If x[i]>0 then
      Begin
        SearchePos:=i; Exit
      End;
End { SearchePos };
{ ------------------------------------ }
Begin
  В в о д   n, X
  i:=1; CondPos:=true;   {    Ц и к л  1 : }
  While CondPos do       { Перестановка положительных }
    If x[i]>0 then       { элементов в начало массива }
      Begin
        Inc(i);
        If i>n then
          CondPos:=false;
      End
    Else
      Begin
        j:=SearchePos(i+1);
        If j=0 then
          CondPos:=false
        Else
          Begin
            R:=x[j];
            For k:=j downto i+1 do
              x[k]:=x[k-1];
            x[i]:=R; Inc(i);
          End;
      End;
    CountZero:=0;          {    Ц и к л  2 : }
    For k:=n downto i do   { Подсчет количества нулевых эле- }
      If x[k]=0 then       { ментов и удаление их из массива }
        Begin
          Inc(CountZero);
          For j:=k to n-1 do
            x[j]:=x[j+1];
        End;
    If CountZero<n then              {   Ц и к л  3 : }
      For i:=n-CountZero+1 to n do   { Запись нул.элементов }
```

```
        x[i]:=0;                        { в конец массива      }
    П е ч а т ь  X
End.
```

Рассмотрим частные случаи.

1. Все элементы массива *X* положительные. В этом случае циклы 2 и 3 работать не будут.

2. В массиве *X* имеются положительные и отрицательные элементы, но нет нулевых элементов: цикл 3 не работает.

1.9. Все положительные числа в одномерном вещественном массиве переставить в обратном порядке, не изменяя положения остальных чисел.

В программе Task109 с помощью функций *FirstNumber* и *LastNumber* производится одновременно поиск положительных чисел слева (индекс k_1) и справа (индекс k_2) в массиве *X*. Если $k_1 < k_2$, то выполняется обмен элементов x_{k_1} и x_{k_2}, в противном случае обработка массива прекращается.

```
Program Task109;
Const  Nmax = 500;
Type   Ar = array[1..Nmax] of real;
Var    k1,k2,n : word;
       R : real;
       X : Ar;
{ --------------------------- }
Function FirstNumber(k:word):word;
{ Поиск положит.числа слева направо, начиная с позиции k }
Var  i : word;
Begin
  FirstNumber:=0;
  For i:=k to n do
    If x[i]>0 then
      Begin
        FirstNumber:=i; Exit
      End
End { FirstNumber };
{ --------------------------- }
Function LastNumber(k:word):word;
{ Поиск положит.числа справа налево, начиная с позиции k }
Var  i : word;
Begin
  LastNumber:=0;
  For i:=k downto 1 do
    If x[i]>0 then
      Begin
        LastNumber:=i; Exit
      End
End { LastNumber };
{ --------------------------- }
Begin
  В в о д  n, X
  k1:=0; k2:=n+1;
  Repeat
    k1:=FirstNumber(k1+1); k2:=LastNumber(k2-1);
```

```
    If k1<k2 then
      Begin
        R:=x[k1]; x[k1]:=x[k2]; x[k2]:=R
      End;
  Until k1>=k2;
  П е ч а т ь  X
End.
```

Если в массиве нет положительных чисел, то FirstNumber = LastNumber = 0; если в массиве имеется только одно число $x_k > 0$, то FirstNumber = LastNumber = k. В обоих случаях никакого обмена элементов массива *X* не производится.

1.10. Элементами целочисленного массива *X* являются числа 0 и 1. Определить
- количество единичных серий, т.е. подмассивов, состоящих из единичных элементов;
- среднюю длину этих серий;
- положение наиболее длинной серии (индекс ее первого элемента).

Признак начала серии - это первый ее элемент $x_{k_1} = 1$, признак конца серии - элемент $x_{k_2} = 0$. Поиск начала и конца очередной серии осуществляется функциями *SignBegin* и *SignEnd.*

Алгоритм обработки массива *X* аналогичен рассмотренному в разделе "Поиск однородной группы в массиве".

При последовательном просмотре массива *X* производится отбор наиболее длинной серии, подсчитывается общее количество серий и их суммарная длина, а после завершения цикла определяется средняя длина серий.

```
Program Task110;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of byte;
Var  n,                { размер массива }
     m,                { количество серий }
     lmax,             { размер наиболее длинной серии }
     kmax,             { позиция начального элемента }
                       { наиболее длинной серии }
     k1,k2,            { индексы начального и конечного }
                       { элементов очередной серии }
     l,                { размер очередной серии }
     lm : word;        { общая длина серий }
     ls : real;        { средняя длина серии }
     Cond : boolean;   { переменная для управления циклом }
     X : Ar;           { исходный массив }
{ --------------------------- }
Function SignBegin(k:word):word;
{ Определение индекса начального элемента серии }
Var  i : word;
Begin
  SignBegin:=0;
  For i:=k to n do
    If x[i]=1 then
      Begin
        SignBegin:=i; Exit
      End;
End { SignBegin };
```

```
{ --------------------------- }
Function SignEnd(k:word):word;
{ Определение индекса конечного элемента серии }
Var  i : word;
Begin
  SignEnd:=0;
  For i:=k to n do
    If x[i]=0 then
      Begin
        SignEnd:=i; Exit
      End;
End { SignEnd };
{ --------------------------- }
Begin
  В в о д  n, X
  k2:=0; Cond:=true; lmax:=0; kmax:=0;
  m:=0; lm:=0; ls:=0;
  While Cond do
    Begin
      k1:=SignBegin(k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=SignEnd(k1+1);
          If k2=0 then
            Begin
              k2:=n+1; Cond:=false
            End;
          l:=k2-k1;
          Inc(m); Inc(lm,l);
          If l>lmax then
            Begin
              lmax:=l; kmax:=k1
            End;
        End;
    End;
  If m>0 then      { Блокировка деления, если массив }
    ls:=lm/m;      { содержит только нулевые элементы }
  П е ч а т ь   m, lmax, kmax, ls
End.
```

1.11. Заданы массивы $X=(x_1,x_2,...,x_n)$ и $Y=(y_1,y_2,...,y_m)$, каждый из которых содержит неповторяющиеся элементы. Сформировать массив Z, в состав которого включить элементы, которые одновременно содержатся в массивах X и Y.

Задача 1.11 - это задача пересечения множеств, заданных в виде массивов неповторяющихся элементов. Количество элементов в результирующем множестве не может превышать количество элементов в меньшем из двух исходных множеств (в частном случае это количество *k* может быть равным нулю).

```
Program Task111;
Const  Nmax = 500; Mmax = 300;
Type   Ar1 = array[1..Nmax] of integer;
       Ar2 = array[1..Mmax] of integer;
```

```
Var  i,j,k,n,m : word;
     X : Ar1;
     Y,Z : Ar2;
Begin
  В в о д  и  п е ч а т ь  n, X, m, Y
  k:=0;
  For i:=1 to n do
    For j:=1 to m do
      If x[i]=y[j] then
        Begin
          Inc(k); z[k]:=x[i];
          Break                { выход за пределы цикла по j }
        End;
  П е ч а т ь  k, Z
End.
```

1.12. Задан целочисленный массив $X=(x_1,x_2,...,x_n)$. Сформировать массив Y, включив в него попарно различные числа из массива X. При этом должен сохраняться исходный относительный порядок этих чисел.

Условие задачи означает, что в массиве Y не должно быть повторяющихся элементов.

До начала обработки массива X в массив Y записывается первый элемент x_1. В дальнейшем для очередного числа x_i, $i=2..n$ предварительно проверяется, нет ли равного ему числа в массиве Y. Если такое число не обнаружено, в массив Y дописывается новый элемент x_i.

```
Program Task112;
Const  Nmax = 300;
Type   Ar = array[1..Nmax] of integer;
Var  i,j,n,m : word;
     R : integer;
     Cond : boolean;
     X,Y : Ar;
Begin
  В в о д  и  п е ч а т ь  n, X
  m:=1; y[1]:=x[1];
  For i:=2 to n do
    Begin
      R:=x[i]; Cond:=true;
      For j:=1 to m do
        If R=y[j] then
          Begin
            Cond:=false; Break
          End;
      If Cond then
        Begin
          Inc(m); y[m]:=R;
        End;
    End;
  П е ч а т ь  m, Y
End.
```

1.13. Заданы массивы $X=(x_1,x_2,...,x_n)$ и $Y=(y_1,y_2,...,y_m)$, упорядоченные по убыванию. Сформировать из их элементов массив Z, также упорядоченный по убыванию. Группировку элементов массива Z не производить.

В цикле 1 попарно сравниваются элементы x_i и y_j, больший из них переносится в массив *Z*. Цикл 1 работает до тех пор, пока не будет исчерпан один из исходных массивов. Тогда "хвост" оставшегося массива переносится в *Z* циклом 2 или циклом 3. Если массив *X* пустой (*nx* = 0) или массив *Y* пустой (*ny* = 0), то цикл 1 не работает, перепись непустого массива осуществляется циклом 2 или циклом 3.

```
Program Task113;
Const  Nmax = 300; Mmax = 500;
Type   ArX = array[1..Nmax] of real;
       ArY = array[1..Mmax] of real;
       ArZ = array[1..Nmax+Mmax] of real;
Var  i,j,k,
     nx,ny,nz : word;  { размеры массивов X, Y, Z }
     X : ArX; Y : ArY; Z : ArZ;
Begin
  В в о д  и  п е ч а т ь  nx, X, ny, Y
  i:=1; j:=1; nz:=0;
  While (i<=nx) and (j<=ny) do      { Цикл 1 }
    Begin
      Inc(nz);
      If x[i]>y[j] then
        Begin
          z[nz]:=x[i]; Inc(i);
        End
      Else
        Begin
          z[nz]:=y[j]; Inc(j);
        End;
    End;
  For k:=i to nx do                 { Цикл 2 }
    Begin
      Inc(nz); z[nz]:=x[k];
    End;
  For k:=j to ny do                 { Цикл 3 }
    Begin
      Inc(nz); z[nz]:=y[k];
    End;
  П е ч а т ь   nz, Z
End.
```

1.14. В массив *X* после каждого отрицательного элемента вставить нулевой элемент. Буферный массив не использовать.

Максимальное изменение массива *X* имеет место в случае, когда все его элементы отрицательные. Это учитывается в объявлении типа массива (при этом по-прежнему считается $n \le N_{\max}$).

```
Program  Task114;
Const  Nmax = 500;
```

```
Type  Ar = array[1..2*Nmax] of real;
Var i,j,n : word;
    X : Ar;
Begin
  В в о д   n, X
  For i:=n downto 1 do
    If x[i]<0 then
      Begin
        Inc(n);
        For j:=n downto i+2 do
          x[j]:=x[j-1];
        x[i+1]:=0;
      End;
  П е ч а т ь  n, X
End.
```

1.15. В целочисленном массиве *X* имеется один нулевой элемент. Обменять местами подмассивы, расположенные слева и справа от этого элемента.

Задача реализована в двух вариантах: с использованием и без использования буферного массива.

```
Program Task115a;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,j,k,n : word;
     X,Y : Ar;
Begin
  В в о д   n, X
  For i:=1 to n do      { Определение положения нулевого }
    If x[i]=0 then      { элемента                       }
      Begin
        k:=i; Break
      End;
  For i:=k+1 to n do    { Запись второй половины массива X }
    y[i-k]:=x[i];       { в начало массива Y }
  j:=n-k+1; y[j]:=0;    { Запись разделяющего нуля в массив Y }
  For i:=1 to k-1 do    { Запись первой половины массива X  в }
    Begin               { конец массива Y  }
      Inc(j); y[j]:=x[i]
    End;
  X:=Y;                 { Перепись массива Y в массив X }
  П е ч а т ь  X
End.
```

Если в массиве *X* второй подмассив был пустым ($x_n = 0$), то в массиве *Y* пустым будет первый подмассив ($y_1 = 0$). При $x_1 = 0$ получим $y_n = 0$.

```
Program Task115b;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,j,k,n : word;
     R : integer;
     X : Ar;
```

```
Begin
  В в о д   n, X
  For i:=1 to n do           { Определение положения нулевого }
    If x[i]=0 then           { элемента                       }
      Begin
        k:=i; Break
      End;
  If k>1 then                { Фрагмент 1: сдвиг элементов }
    Begin                    { x[1]..x[k-1] вправо и запись }
      For j:=k downto 2 do   { нуля в x[1] }
        x[j]:=x[j-1];
      x[1]:=0;
    End;
  For i:=1 to n-k do         { Фрагмент 2: поочередная пере- }
    Begin                    { сылка элементов  второго под- }
      R:=x[n];               { массива  в начало  массива X; }
      For j:=n downto 2 do   { n-k – количество элементов во }
        x[j]:=x[j-1];        { втором подмассиве }
      x[1]:=R;
    End;
  П е ч а т ь  X
End.
```

Работу программы Task115b рассмотрим на конкретном примере:
X = (1, 2, 3, 0, 4, 5, 6, 7, 8) .
Здесь k = 4; n = 9; n - k = 5 .
После окончания работы фрагмента 1 получим:
X = (0, 1, 2, 3, 4, 5, 6, 7, 8) .
Фрагмент 2:

```
  i = 1 :   X = (8,0,1,2,3,4,5,6,7).
  i = 2 :   X = (7,8,0,1,2,3,4,5,6).
  i = 3 :   X = (6,7,8,0,1,2,3,4,5).
  i = 4 :   X = (5,6,7,8,0,1,2,3,4).
  i = 5 :   X = (4,5,6,7,8,0,1,2,3).
```

Частные случаи:

а) $k = 1$ (нет левого подмассива). Фрагмент 1 не срабатывает.

а) $k = n$ (нет правого подмассива). Фрагмент 2 не срабатывает.

1.16. Задан целочисленный массив *X*, в котором могут быть одинаковые элементы. Найти минимальный и максимальный элементы среди неповторяющихся чисел и обменять их местами.

```
Program Task116;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,j,n,
     imax,imin : word;     { индексы макс. и мин. элементов }
     R,
     xmax,xmin : integer;  { значения макс. и мин. элементов }
     Cond : boolean;
     X : Ar;
Begin
  В в о д   n, X
  imax:=0; imin:=0;
```

```
  For i:=1 to n do
    Begin
      R:=x[i]; Cond:=true;            { Элемент x[i] }
      For j:=1 to n do                { не повторяется, }
        If (i<>j) and (R=x[j]) then   { если  Cond = true }
          Begin
            Cond:=false; Break
          End;
      If Cond then
        If imax=0 then                { Первое нахождение }
          Begin                       { неповторяющегося }
            imax:=i; imin:=i;         { элемента }
            xmax:=x[i]; xmin:=x[i];
          End
        Else                          { Очередное нахождение }
          If x[i]>xmax then           { неповторяющегося }
            Begin                     { элемента }
              xmax:=x[i]; imax:=i
            End
          Else
            If x[i]<xmin then
              Begin
                xmin:=x[i]; imin:=i
              End;
    End;
  If (imax>0) and (imax<>imin) then
    Begin                             { Обмен элементов }
      x[imax]:=xmin; x[imin]:=xmax;
    End;
  П е ч а т ь  xmin, imin, xmax, imax
End.
```

Обмен элементов в массиве X производится при соблюдении двух условий:

а) $i_{max} > 0$ (в массиве имеются неповторяющиеся элементы);

б) $i_{max} \neq i_{min}$ (в массив входит более одного неповторяющегося элемента).

1.17. Даны два равных по размеру целочисленных массива X и Y. Определить, верно ли утверждение, что эти массивы отличаются между собой лишь порядком следования своих элементов. Например, массивы

$$X = (8, 3, 5, 8, 5) \text{ и } Y = (5, 8, 5, 3, 8)$$

соответствуют этому утверждению, а массивы

$$X = (8, 3, 5, 8, 5) \text{ и } Y = (5, 3, 5, 3, 8) \text{ - нет.}$$

```
Program Task117;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,n : word;
     CondElem : boolean;
     X,Y : Ar;
{ ------------------------------ }
Procedure Buble(Var A:Ar; n:word);
{ Группировка массива по возрастанию методом "пузырька" }
Var   i : word;
      R : integer;
```

```
      Cond : boolean;
Begin
  Cond:=true;
  While Cond do
    Begin
      Cond:=false;
      For i:=1 to n-1 do
        If a[i]>a[i+1] then
          Begin
            R:=a[i]; a[i]:=a[i+1]; a[i+1]:=R;
            Cond:=true;
          End;
      Dec(n);
    End;
End { Buble };
{ ------------------------------ }
Begin
  В в о д   n, X, Y
  Buble(X,n); Buble(Y,n);
  CondElem:=true;
  For i:=1 to n do
    If x[i]<>y[i] then
      Begin
        CondElem:=false; Break
      End;
  If CondElem then
    Writeln('Массивы одинаковы по составу элементов')
  Else
    Writeln('Массивы различаются составом элементов');
End.
```

В процедуре *Buble* изменяется формальная переменная *n*, определяющая размер массива *X*, что, казалось бы, должно отрицательно повлиять на обработку этого массива в основной программе. Однако формальная переменная *n* - это параметр-значение (перед ее именем нет слова ***Var***); этой переменной в теле процедуры отводится отдельное поле памяти, в которое записывается значение фактической переменной *n*. Поскольку формальная и фактическая переменные *n* имеют различные адреса памяти, то изменение первой из них в теле процедуры не отражается на значении фактического параметра *n*.

1.18. Элементы целочисленного массива X строго упорядочены в порядке возрастания. Если в массиве X имеется элемент, равный заданному значению b, то отпечатать количество и сумму предшествующих ему элементов. Среднее количество проверок массива не должно превышать $\log_2(n)$.

```
Program Task118;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,k,n : word;
     b,Num,Sum : integer;
     X : Ar;
{ ------------------------------ }
Function BinarySearche(Var A:Ar; n:word; b:integer):word;
{ Двоичный поиск элемента в массиве }
Var  k1,k2,m : word;
Begin
```

```
  k1:=1; k2:=n;
  BinarySearche:=0;
  While k1<=k2 do
    Begin
      m:=(k1+k2) div 2;
      If a[m]=b then
        Begin
          BinarySearche:=m; Exit
        End
      Else
        If a[m]<b then
          k1:=m+1
        Else
          k2:=m-1;
    End;
End { BinarySearche };
{ ------------------------------ }
Begin
  В в о д   n, X, b
  k:=BinarySearche(X,n,b);
  Sum:=0; Num:=k-1;
  For i:=1 to k-1 do
    Sum:=Sum+x[i];
  П е ч а т ь  Num, Sum
End.
```

Если $k = 1$, то цикл ***For*** работать не будет. При этом сохраняется значение $Sum = 0$, а переменная *Num* будет равна $k - 1 = 0$.

1.19. В массиве целых положительных чисел, упорядоченных по возрастанию, определить положение наиболее длинной группы, представляющей собой отрезок натурального ряда чисел.

Числовая последовательность является отрезком натурального ряда, если каждый элемент этой последовательности на единицу больше предыдущего элемента.

Организация программы Task119 соответствует методике, изложенной в разделе "Поиск однородной группы в массиве".

```
Program Task119;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of word;
Var  i,n,k1,k2,
     kmax,       { индекс первого элемента наибольшей группы }
     lmax,       { длина наибольшей группы }
     l : word;   { длина очередной группы }
     Cond : boolean;
     X : Ar;
{ ------------------------------- }
Function SignBegin(k:word):word;
Var  i : word;
Begin
  SignBegin:=0;
  For i:=k to n-1 do
    If x[i+1]-x[i]=1 then
```

```
      Begin
        SignBegin:=i; Exit
      End;
End { SignBegin };
{ ---------------------------------- }
Function SignEnd(k:word):word;
Var  i : word;
Begin
  SignEnd:=0;
  For i:=k to n-1 do
    If x[i+1]-x[i]<>1 then
      Begin
        SignEnd:=i; Exit
      End;
End { SignEnd };
{ ---------------------------------- }
Begin
  В в о д   n, X
  kmax:=0; lmax:=0;
  k2:=0; Cond:=true;
  While Cond do
    Begin
      k1:=SignBegin(k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=SignEnd(k1+1);
          If k2=0 then
            Begin
              k2:=n; Cond:=false;
            End;
          l:=k2-k1+1;
          If l>lmax then
            Begin
              lmax:=l; kmax:=i;
            End;
        End;
    End;
  П е ч а т ь   kmax, lmax
End.
```

1.20. Каждый элемент целочисленного массива *X* трактуется как звено цепи и определяет номер следующего звена в этой цепи. Последнее звено имеет численное значение вне диапазона 1 .. *n* или равно собственному индексу. Определить размер максимальной по длине цепи и номер ее первого элемента.

Например, в массиве

5 10 20 6 8 11 9 3 4 4 0 ($n = 11$)

такой цепью является 10, 4, 6, 11, 0 (5 звеньев, начальный элемент имеет индекс 2).

```
Program Task120;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of word;
Var  i,n,kmax,lmax,l,Elem : word;
     Cond : boolean;
```

```
     X : Ar;
Begin
  В в о д   n, X
  kmax:=0; lmax:=0;
  For i:=1 to n do
    Begin
      l:=1; Elem:=x[i]; Cond:=true;
      While Cond do
        Begin
          Cond:=(Elem>0) and (Elem<=n) and (Elem<>i);
          If Cond then
            Begin
              Inc(l); Elem:=x[Elem];
            End;
        End;
      If l>lmax then
        Begin
          lmax:=l; kmax:=i;
        End;
    End;
  П е ч а т ь  kmax, lmax
End.
```

1.21. Среди отрицательных элементов целочисленного массива *X* определить значение максимального элемента $x_{\max}$ и количество равных ему элементов. Просмотр массива выполнять один раз.

```
Program Task121;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,n,kmax : word;
     Xmax : integer;
     X : Ar;
Begin
  В в о д   n, X
  kmax:=0; Xmax:=0;
  For i:=1 to n do
    If x[i]<0 then
      If kmax=0 then            { Первое нахождение }
        Begin                   { отрицательного }
          Xmax:=x[i]; kmax:=1; { элемента }
        End
      Else
        If x[i]>Xmax then       { Повторное нахождение }
          Begin                 { отрицательного }
            Xmax:=x[i]; kmax:=1{ элемента }
          End
        Else
          If x[i]=Xmax then     { Подсчет }
            Inc(kmax);          { макс.элементов }
  П е ч а т ь   Xmax, kmax
End.
```

1.22. В целочисленном массиве имеются группы одинаковых положительных элементов. Для каждой группы, содержащей не менее трех элементов, отпечатать: значение и индекс первого элемента, длину группы.

Организация программы Task122 соответствует методике, изложенной в разделе "Поиск однородной группы в массиве".

```
Program Task122;
Const  Nmax = 500;
Type  Ar = array[1..Nmax] of integer;
Var  i,n,k1,k2,l : word;
     Number : integer;
     CondGroup,Cond : boolean;
     X : Ar;
{ ------------------------------------- }
Function SignBegin(k:word):word;
{ Поиск начала группы }
Var  i : word;
Begin
  SignBegin:=0;
  For i:=k to n do
    If x[i]>0 then
      Begin
        SignBegin:=i; Exit;
      End;
End { SignBegin };
{ ------------------------------------- }
Function SignEnd(Number:integer; k:word):word;
{ Поиск конца группы }
Var  i : word;
Begin
  SignEnd:=0;
  For i:=k to n do
    If x[i]<>Number then
      Begin
        SignEnd:=i; Exit
      End;
End { SignEnd };
{ ------------------------------------- }
Begin
  В в о д   n, X
  CondGroup:=false; Cond:=true; k2:=0;
  While Cond do
    Begin
      k1:=SignBegin(k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          Number:=x[k1]; k2:=SignEnd(Number,k1+1);
          If k2=0 then
            Begin
              k2:=n+1; Cond:=false;
            End;
          l:=k2-k1;
          If l>2 then
            Begin
              Writeln(Number,'    ',k1,'    ',l);
```

```
              CondGroup:=true;
            End;
          End;
      End;
    If not CondGroup then
      Writeln('Нет групп одинаковых положительных элементов');
End.
```

1.23. Определить количество различных остроугольных треугольников с вершинами в заданном множестве точек на плоскости.

При решении этой задачи нужно предварительно рассмотреть два вопроса:
- как разграничить остро-, прямо- и тупоугольные треугольники;
- как определить равенство двух треугольников.

В обоих случаях необходимо проверять отношение двух чисел. Такое отношение является точным лишь для целых чисел, для вещественных чисел должна производиться проверка по ε, где ε - достаточно малое число, обычно трактуемое как погрешность представления данных (например, равны ли числа 4.999999 и 5.000001, если погрешность их представления равна 0.001 ?). Тогда имеем

$$a < b, \quad \text{если } a - b < -\varepsilon$$
$$a = b, \quad \text{если } |a - b| \le \varepsilon$$
$$a > b, \quad \text{если } a - b > \varepsilon$$

Если $c = \max(a,b,c)$, то

треугольник прямоугольный, если $\left|c^2 - a^2 - b^2\right| \le \varepsilon$;

остроугольный, если $c^2 - a^2 - b^2 < -\varepsilon$;

тупоугольный, если $c^2 - a^2 - b^2 > \varepsilon$.

Два треугольника равны между собой, если равны соответствующие их стороны. Естественно, проверка равенства должна производиться по ε.

В программе Task123 стороны очередного треугольника d_1, d_2, d_3, сгруппированные по возрастанию, записываются в массивы *A, B, C*. Запись нового остроугольного треугольника производится лишь в случае, если в этих массивах отсутствует треугольник с такими же сторонами.

```
Program Task123;
Const  Nmax = 500;  eps = 0.001;
Type  Ar = array[1..Nmax] of real;
Var  i,j,k,it,
     n,                  { количество точек }
     nt : word;          { количество треугольников }
     d1,d2,d3,           { длины сторон треугольника }
     r  : real;
     Cond : boolean;
     X,Y,                { координаты точек на плоскости }
     A,B,C : Ar;         { массивы сторон треугольников }
{ ------------------------------- }
Procedure Swap(Var p,q ^ real);
Var  r : real;
Begin
  If p>q then
```

```
  Begin
    r:=p; p:=q; q:=r;
  End;
End { Swap };
{ ---------------------------------- }
Begin
  Ввод массивов координат точек  (n,X,Y)
  If n>2 then
    Begin
      nt:=0;
      For i:=1 to n-2 do     { Перебор вершин треугольников }
        For j:=i+1 to n-1 do
          For k:=j+1 to n do
            Begin
              d1:=sqr(x[i]-x[j])+sqr(y[i]-y[j]);
              d2:=sqr(x[j]-x[k])+sqr(y[j]-y[k]);
              d3:=sqr(x[k]-x[i])+sqr(y[k]-y[i]);
              Swap(d1,d2);       { Группировка d1, d2, d3 }
              Swap(d1,d3);       { по возрастанию }
              Swap(d2,d3);
              If d3-d2-d1<-eps then   { Проверка остроуголь- }
                Begin                 { ности треугольника   }
                  d1:=sqrt(d1); d2:=sqrt(d2); d3:=sqrt(d3);
                  Cond:=false;
                  For it:=1 to nt do      { Проверка наличия }
                    If (abs(d1-a[it])<=eps) and     { такого }
                       (abs(d2-b[it])<=eps) and     { тр-ка }
                       (abs(d3-c[it])<=eps) then    { в мас- }
                      Begin                  { сивах A, B, C }
                        Cond:=true; Break;
                      End;
                  If not Cond then      { Запись нового тре- }
                    Begin               { угольника в }
                      Inc(nt);          { массивы A, B, C }
                      a[nt]:=d1; b[nt]:=d2; c[nt]:=d3;
                    End;
                End;
            End;
        If nt=0 then
          Writeln('Нет остроугольных треугольников')
        Else
          Writeln('Кол-во остроуг.треугольников ',nt);
    End
  Else
    Writeln('На плоскости задано меньше трех точек');
End.
```

1.24. Заданный вещественный массив $X = (x_1, x_2, ..., x_n)$ разделить на две части таким образом, чтобы разница сумм элементов двух подмассивов была минимальной.

Решение задачи сводится к перебору всех сочетаний из n по m элементов, где $m = 1..m_{\max}$; $m_{\max} = n$ ***div*** 2.

Рассмотрим возможные значения массива индексов *Ind*, состоящего из m элементов, при $n = 8$, $m = 3$.

1 2 3	1 4 8	2 4 7	3 5 8
1 2 4	1 5 6	2 4 8	3 6 7
1 2 5	1 5 7	2 5 6	3 6 8
1 2 6	1 5 8	2 5 7	3 7 8
1 2 7	1 6 7	2 5 8	4 5 6
1 2 8	1 6 8	2 6 7	4 5 7
1 3 4	1 7 8	2 6 8	4 5 8
1 3 5	2 3 4	2 7 8	4 6 7
1 3 6	2 3 5	3 4 5	4 6 8
1 3 7	2 3 6	3 4 6	4 7 8
1 3 8	2 3 7	3 4 7	5 6 7
1 4 5	2 3 8	3 4 8	5 6 8
1 4 6	2 4 5	3 5 6	5 7 8
1 4 7	2 4 6	3 5 7	6 7 8

Для решения задачи необходимо исследовать закономерность изменения элементов массива *Ind.*

Обозначим через *i* индекс элемента массива *Ind,* $i = 1 .. m$.

Нетрудно заметить, что максимальное значение каждого элемента Ind_i равно $n - m + i$, а перебор значений *Ind* заканчивается, когда элемент Ind_1 становится равным $n - m + 1$.

До начала цикла перебора установим $Ind_i := i$, $i = 1 .. m$. После этого в каждом цикле формирования массива *Ind* проверяем, имеется ли в нем элемент Ind_k ($k = 2 .. m$), принявший значение $n - m + k$. Если такой факт обнаруживается, то к элементу Ind_{k-1} добавляем 1, а последующие элементы массива *Ind* принимают значения $Ind_i := Ind_{i-1} + 1$, $i = k..m$. В противном случае добавляется 1 к последнему элементу Ind_m.

В программе Task124 используется описанный выше алгоритм формирования элементов индексного массива *Ind.* При этом последовательно проверяются сочетания из *m* элементов, где $m = 1 .. mm$, $mm = n \ div \ 2$. Для каждого сочетания вычисляется сумма S_1 элементов массива *X,* индексы которых определены элементами массива *Ind.* S_1 - это сумма элементов первого подмассива. Для второго подмассива имеем $S_2 = S - S_1$, где *S* - сумма всех элементов массива *X*. Если разность $dS = |S_1 - S_2|$ меньше значения $dS_{\min}$, то запоминаются новое значение $dS_{\min}$, сумма элементов первого подмассива $S_{\min}$, количество элементов в этом подмассиве $m_{\min}$ и индексы этих элементов $Ind_{\min}$.

```
Program Task124;
Const  Nmax = 50; Mmax = Nmax div 2;
Type   ArReal = array[1..Nmax] of real;
       ArInd  = array[1..Mmax] of byte;
Var    i,j,k,
       n,            { кол-во эл-тов в исходном массиве }
       m,            { кол-во эл-тов в первом подмассиве }
       mm,           { макс.кол-во эл-тов в первом подмассиве }
       Mmin : byte;{ кол-во эл-тов в первом подмассиве }
                     { при  dS = dSmin }
       X : ArReal; { исходный массив }
       Ind,          { массив индексов эл-тов первого подмассива }
       IndMin : ArInd; { то же при  dS = dSmin }
       S,            { сумма эл-тов исходного массива }
       S1,           { сумма эл-тов первого подмассива }
```

```
      S2,          { то же для второго подмассива }
      dS,          { dS = abs(S1 - S2) }
      S1min,       { значение S1 при  dS = dSmin }
      dSmin        { минимальное значение dS }
          : real;
Begin
  В в о д   n, X
  S:=0;
  For i:=1 to n do
    S:=S+x[i];
  dSmin:=1E35; mm:=n div 2;
  For m:=1 to mm do
    Begin
      For i:=1 to m do
        Ind[i]:=i;
      While Ind[1]<=n-m+1 do
        Begin
          S1:=0;
          For j:=1 to m do
            S1:=S1+x[Ind[j]];
          S2:=S-S1; dS:=abs(S1-S2);
          If dS<dSmin then
            Begin
              dSmin:=dS; S1min:=S1;
              IndMin:=Ind; Mmin:=m;
            End;
          k:=0;
          For i:=2 to m do
            If Ind[i]=n-m+i then
              Begin
                k:=i; Break;
              End;
          If k>0 then
            Begin
              Inc(Ind[k-1]);
               For i:=k to m do
                 Ind[i]:=Ind[i-1]+1;
             End
           Else
             Inc(Ind[m]);
        End;
    End;
  Печать результатов:
    Mmin, S, S1min, dSmin
    Массив индексов  IndMin
    Элементы  x[IndMin[i]],  i = 1..Mmin
End.
```

1.25. Заданы два вещественных массива A и B по n чисел в каждом. Сформировать массив индексов C такой, чтобы сумма

$$S = a_1 b_{c_1} + a_2 b_{c_2} + ... + a_n b_{c_n}$$

была минимальной.

Для решения задачи нужно формировать все перестановки индексов, записанных в массиве C. Первой такой перестановкой будет 1,2,3, ..., $n-1, n$, последней - $n, n-1$, ..., 2, 1. Для разработки алгоритма запишем в явном виде перестановки из $n = 4$ элементов:

1 2 3 4	2 3 1 4	3 4 1 2
1 2 4 3	2 3 4 1	3 4 2 1
1 3 2 4	2 4 1 3	4 1 2 3
1 3 4 2	2 4 3 1	4 1 3 2
1 4 2 3	3 1 2 4	4 2 1 3
1 4 3 2	3 1 4 2	4 2 3 1
2 1 3 4	3 2 1 4	4 3 1 2
2 1 4 3	3 2 4 1	4 3 2 1

Алгоритм формирования перестановок можно сформулировать следующим образом.

1. Записать начальный массив индексов в виде $1, 2, 3, ..., n-1, n$.

2. Просматривая массив индексов справа налево, определить ближайшее значение k, при котором $c_k < c_{k+1}$. Если такой элемент не обнаружен, работа алгоритма закончена.

3. Для k-го элемента найти значение c_t $(t = k+1, ..., n)$, чтобы разница $c_t - c_k$ была минимальной при условии $c_t > c_k$.

4. Обменять местами элементы c_t и c_k.

5. Сгруппировать элементы $c_{k+1}, ... c_n$ в порядке возрастания. Поскольку к этому моменту эти элементы расположены в порядке убывания, то достаточно переставить их в обратном порядке.

```
Program Task125;
Const  Nmax = 50;
Type   ArReal = array[1..Nmax] of real;
       ArInd  = array[1..Nmax] of byte;
Var   i,j,k,t,dt,r,
      n : byte;        { кол-во эл-тов в исходных массивах }
      A,B : ArReal;    { исходные массивы }
      C,               { массив индексов }
      Cmin : ArInd;    { то же при  S = Smin }
      S,               { сумма эл-тов a[i]*b[i] }
      Smin : real;     { минимальное значение S }
      Cond : boolean;
Begin
  Ввод и печать массивов A и B
  For i:=1 to n do     { Формирование исходного }
    c[i]:=i;           { массива индексов }
  Smin:=0;             { Определение начального }
  For i:=1 to n do     { значения Smin и Cmin }
    Smin:=Smin+a[i]*b[i];
  Cmin:=C;
  Cond:=true;
  While Cond do        { формирование перестановок }
    Begin              { индексов и определение }
      k:=n-1;          { значения Smin }
      While (k>0) and (c[k]>c[k+1]) do
        Dec(k);
      If k=0 then
        Cond:=false
      Else
        Begin
          t:=k+1; dt:=c[t]-c[k];
```

```
            For i:=t+1 to n do
              If (c[i]>c[k]) and (c[i]-c[k]<dt) then
                Begin
                  t:=i; dt:=c[i]-c[k];
                End;
            r:=c[k]; c[k]:=c[t]; c[t]:=r;
            i:=k+1; j:=n;
            While i<j do
              Begin
                r:=c[i]; c[i]:=c[j]; c[j]:=r;
                Inc(i); Dec(j);
              End;
            S:=0;
            For i:=1 to n do
              S:=S+a[i]*b[c[i]];
            If S<Smin then
              Begin
                Smin:=S; Cmin:=C;
              End;
          End;
      End;
    Печать  Smin, Cmin
End.
```

Примечание. На примере этой задачи можно продемонстрировать основной недостаток алгоритмов полного перебора: экспоненциальное увеличение времени решения задачи с ростом размерности обрабатываемых массивов.

Количество перестановок из n элементов

$$P_n = n!$$

Численный эксперимент, проведенный на Pentium-IV (тактовая частота 1700 Мгц), показал следующие результаты:

n	Время решения	P_n
8	0,16 с	40 320
9	1,54 с	362 880
10	17,41 с	3 628 800
11	3 мин. 28,28 с	39 916 800
12	45 мин. 44,46 с	479 001 600
13	свыше 10 часов	6 227 020 800

Задания для самостоятельной работы

1.26. Определить, является ли целочисленный массив $X(n)$ перестановкой элементов арифметической или геометрической прогрессии.

Рекомедация. Для анализа арифметической прогрессии целесообразно использовать зависимость

$$2x_i = x_{i-1} + x_{i+1},$$

а для геометрической прогрессии – зависимость

$$x_i^2 = x_{i-1} \cdot x_{i+1}.$$

1.27. Для заданного значения n сформировать массив $X(n)$, являющийся последовательностью чисел Фибоначчи. Определить, насколько отличается положение элемента,

наиболее близкого к среднему арифметическому значению массива X, от положения элемента, наиболее близкого к среднему геометрическому значению этого же массива.

1.28. Заданы два вещественных массива $X(n)$ и $Y(m)$. Сформировать массив Z, в который включить из массива X элементы, превышающие среднее арифметическое значение массива Y, а из массива Y - элементы, превышающие среднее арифметическое значение массива X.

1.29. Определить, сколько различных чисел содержит целочисленный массив $X(n)$. Буферный массив не использовать.

Например, в массиве (5, 8, 5, 7, 8) таких чисел три: 5, 7 и 8.

1.30. В целочисленном массиве $X(n)$ найти число, повторяющееся максимальное количество раз. Если таких чисел несколько, то одно из них.

1.31. Определить, имеются ли в массиве целых чисел $X(n)$ совпадающие элементы. Если такие элементы имеются, то в каждой группе совпадающих элементов оставить два первых по порядку элемента, а остальные удалить из состава массива.

1.32. Дан массив целых чисел $X(n)$. Сформировать массив $Y(m)$, поместив в него в порядке возрастания все различные числа, входящие в массив X. Группировку массивов не производить.

1.33. Заданы целочисленные массивы $X(n)$ и $Y(m)$, каждый из которых содержит неповторяющиеся элементы. Объединить массивы X и Y, обеспечив неповторяемость элементов нового массива.

1.34. В каждом из целочисленных массивов $X(n)$ и $Y(m)$ нет повторяющихся элементов. Сформировать массив Z, в который включить из X элементы, отсутствующие в Y, а из Y - элементы, отсутствующие в X.

1.35. В массиве $X(n)$ переставить местами первый и второй отрицательные элементы, третий и четвертый отрицательные элементы и т.д. Определить, как изменилось положение минимального и максимального элементов массива X при его преобразовании.

1.36. Числовая последовательность формируется по правилу:

$$u_0 = \cos(x); \quad u_1 = \cos(x+h); \quad u_2 = \cos(x+2h); \quad \ldots, \quad u_n = \cos(x+nh)$$

(значения x, n, h заданы).

Среди тех элементов последовательности $U = (u_0, u_1, \ldots, u_n)$, которые превышают по модулю заданное значение b, найти максимальный и минимальный элементы, после чего обменять их местами ($0 < b < 1$).

1.37. В массиве $X(n)$ подсчитать количество положительных k_1 и количество отрицательных k_2 элементов. Если $k = |k_1 - k_2| > 1$, то изменить знаки стольких положительных или отрицательных элементов, чтобы выполнялось условие $k \le 1$. Нулевые элементы при определении k_1 и k_2 не учитывать.

1.38. Минимальный элемент массива $X(n)$ обменять местами с последним нулевым элементом этого же массива. Учесть два частных случая:

- массив X не содержит нулевых элементов;
- минимальный элемент массива X равен нулю.

1.39. В целочисленном массиве $X(n)$ имеется единственный нулевой элемент, разделяющий его на две части. В частном случае любой из подмассивов может быть пустым. Элементы первого подмассива сгруппировать по убыванию, а второго - переставить в обратном порядке.

1.40. В неупорядоченном массиве целых положительных чисел определить положение наиболее длинной группы, представляющей собой перестановку элементов отрезка натурального ряда чисел.

1.41. Задан произвольный массив целых положительных чисел, не превышающих значения 1000. В массиве могут быть указаны диапазоны, в этом случае конечному элементу диапазона условно присваивается отрицательное значение. Развернуть исходный массив в возрастающую последовательность чисел. Например, для массива (15,18,-22,10,4,-7) получим (4,5,6,7,10,15,18,19,20,21,22). Группировку элементов массива не производить. В новом массиве не должно быть совпадающих элементов вне зависимости от того, имеются ли такие элементы в исходном массиве.

1.42. Вещественный массив $X(n)$ содержит несколько отрицательных элементов, разделяющих его на отдельные подмассивы. Первый и последний элементы массива неотрицательны. Элементы первого подмассива сгруппировать в порядке возрастания, второго - в порядке убывания, третьего - в порядке возрастания и т.д. Учесть частные случаи (в массиве нет отрицательных элементов; подмассив пустой или содержит только один элемент).

1.43. Массив *X* содержит несколько нулевых элементов, разделяющих его на подмассивы. Определить подмассивы максимальной длины, являющиеся взаимно обратными, т.е один из подмассивов можно получить из другого путем записи его элементов в обратном порядке. Определить среднее арифметическое четных по значению элементов этих подмассивов.

1.44. Задан целочисленный массив $X(n)$, в котором могут быть одинаковые элементы. Найти минимальный и максимальный элементы среди повторяющихся чисел и обменять их местами.

Пример.

Массив

$$X = (52, 21, 12, 45, 35, 21, 19, 41, 12, 21, 35, 41, 12, 12, 41, 21, 14, 41, 34, 41)$$

после обработки будет иметь вид

$$X = (52, 21, 41, 45, 35, 21, 19, 12, 12, 21, 35, 41, 12, 12, 41, 21, 14, 41, 34, 41).$$

1.45. Даны два целочисленных массива $X(n)$ и $Y(m)$ ($m \le n$). Определить, является ли массив Y подмассивом массива X и, если это так, то отпечатать значение k, при котором выполняются равенства $x_k = y_1$; $x_{k+1} = y_2$,...

Пример.

$$X = (52, 21, 12, 14, 17, 33, 19, 22, 12, 17, 35, 41, 12, 12, 19, 37, 14, 16, 34, 25);$$

$$Y = (19, 22, 12, 17, 35, 41, 12).$$

Здесь *Y* является подмассивом массива *X* ($k = 7$), в то время как массив

$$Y = (19, 22, 12, 12, 17, 35, 41, 12)$$

не является подмассивом массива *X*.

1.46. Задан вещественный массив $X(n)$, где x_i - это результаты n спортсменов в беге на 100 м. Составить команду из 4-х лучших бегунов в эстафете 4×100 м, указав индексы и значения соответствующих элементов массива X. Допускается лишь один просмотр массива X.

1.47. Удалить из вещественного массива все элементы, отличающиеся от его максимального элемента не более чем на ε, кроме самого максимального элемента (ε - малое число, например, 0.001).

1.48. В целочисленном массиве определить количество серий, состоящих из квадратов натуральных чисел.

1.49. В целочисленном массиве определить количество серий, элементы которых представляют собой целые степени чисел 2 или 5.

Примечание. Условию задачи удовлетворяют, в частности, числа 32 и 125, но не 100.

1.50. По целочисленному массиву $X(n)$ сформировать массив $Y(m)$, включив в него по убыванию k несовпадающих между собой наибольших элементов массива X ($1 \le k \le n$). Рекомендуется предварительно сгруппировать массив X по убыванию. В частном случае может иметь место $m < k$, если в массиве X много одинаковых элементов. Значение пере-

менной *k* должно быть введено с клавиатуры; переменная *m* – это реальное количество чисел, включенных в состав массива *Y*.

1.51. Удалить из массива *X* наиболее длинную серию чередующихся по четности элементов. Например, такими сериями являются подмассив (7,8,15,22,11,46) или подмассив (16,9,30,31,22,19,24).

1.52. Выбрать три разные точки заданного на плоскости множества точек, составляющие равносторонний треугольник наибольшего периметра, после чего удалить координаты этих точек из соответствующих массивов.

1.53. Заданы два целочисленных массива $X(n)$ и $Y(m)$, $m \le n$, интерпретируемые как множества. Последнее означает, что в каждом из массивов X и Y нет повторяющихся элементов. Определить, является ли массив Y подмножеством массива X, т.е. содержатся ли в массиве X все элементы из массива Y. Группировку элементов массивов X и Y не производить.

1.54. Сформировать таблицу для пересчета милей в километры и километров в мили для расстояний от 1 до 100 км (1 миля = 1.609344 км).

Таблица должна иметь следующий вид:

Мили	Километры
0.6214	1.0000
1.0000	1.6093
1.2428	2.0000
1.8641	3.0000
2.0000	3.2187
.....	.....
62.1371	100.0000

Формирование таблицы выполнить за один цикл.

1.55. В целочисленном массиве имеется несколько групп элементов, образующих строго возрастающие последовательности. Определить среднюю длину тех последовательностей, которые начинаются с отрицательного элемента.

Примечание. В строго возрастающей последовательности выполняются отношения

$$x_1 < x_2 < x_3 < \ldots < x_{n-1} < x_n ,$$

в отличие от возрастающей последовательности, которая определена отношениями

$$x_1 \le x_2 \le x_3 \le \ldots \le x_{n-1} \le x_n .$$

1.56. На прямой задано n точек с равными расстояниями между ними. Определить номер точки, для которой суммарное расстояние до всех остальных точек минимальное, а также значение этого расстояния. Выполнить вариант этой же задачи, когда расстояния между точками неодинаковые и заданы элементами массива *X*(*n*-1).

1.57. Каждый элемент одномерного массива X равен 0 или 1. Найти положение наиболее длинной серии пар 1 и 0. Например, для массива

X = (0,1,1,0,1,0,0,0,1,0,1,1,0,1,0,1,0,1,0,1,1,1,0,1)

получим: длина серии 4, индекс начального элемента 12.

1.58. Элементы массива $X(n)$ представляют собой длины сторон треугольников. Определить, сколько равносторонних треугольников можно построить из элементов этого массива, причем каждый элемент имеет право принимать участие в формировании лишь одного треугольника.

1.59. Повторяющиеся элементы целочисленного массива *X* расположить в порядке убывания, не изменяя положения остальных элементов.

1.60. Заданы вещественные числа $a_1, b_1,\ a_2, b_2,\ \ldots,\ a_n, b_n$. Пары чисел (a_i, b_i) - это левые и правые концы окрашенных отрезков на одной прямой $(a_i \le b_i)$. Некоторые из этих отрезков частично или полностью перекрывают друг друга. Определить

а) наиболее длинный окрашенный отрезок прямой;

б) количество изолированных окрашенных отрезков.

1.61. Массив *X* содержит несколько нулевых элементов, разделяющих его на отдельные подмассивы. Первый и последний элементы массива ненулевые. Удалить из массива *X* пустые подмассивы (вместе с одной из их границ), после чего оставшиеся подмассивы сгруппировать в порядке убывания сумм входящих в них элементов.

1.62. В вещественном массиве $X(n)$ определить номер числа, по абсолютной величине наиболее близкого к своему номеру (индексу).

1.63. Заменить каждый элемент массива $X(n)$ средним арифметическим всех предшествующих ему элементов. Буферный массив не использовать.

1.64. Заданы целочисленные массивы $X(n)$ и $Y(m)$, каждый из которых содержит неповторяющиеся элементы. Найти в массиве *X* наименьший элемент среди тех, которых нет в массиве *Y*, а в массиве *Y* – наибольший элемент среди тех, которых нет в массиве *X*, после чего обменять эти элементы местами.

1.65. Произвольный выпуклый многоугольник задан координатами своих вершин на плоскости. Найти самую длинную диагональ данного многоугольника.

1.66. Вычислить площадь произвольного выпуклого многоугольника, заданного координатами своих вершин на плоскости, разбив многоугольник на треугольники.

Указание. Площадь треугольника с вершинами в точках (x_1, y_1), (x_2, y_2) и (x_3, y_3) вычислять по формуле

$$S = \frac{1}{2}\left| x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_3) \right|$$

1.67. На плоскости заданы координаты *n* вершин произвольного многоугольника. Перенести все вершины многоугольника в первый квадрант, после чего определить его площадь, используя формулу трапеций

$$S = \frac{1}{2}(y_i + y_{i+1})(x_{i+1} - x_i),$$

где $(x_i, y_i), (x_{i+1}, y_{i+1})$ - координаты конечных точек i- ой стороны многоугольника.

2. Матрицы

2.0. Общие замечания.

В программах обработки матриц ввод исходных данных и вывод результатов заменены соответствующими фразами на русском языке. Подробно формирование текстового файла, содержащего элементы матрицы, ввод из этого файла и вывод матрицы на экран рассмотрены в разделе "Ввод и печать элементов матрицы".

2.1. В прямоугольной матрице обменять местами первый отрицательный элемент с последним элементом матрицы.

```
Program Task201;
Label  10;
Const  Mmax = 20; Nmax = 15;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
Var  i,j,m,n,ineg,jneg : byte;
     R : real;
     A : Matrix;
Begin
  В в о д   m, n, A
```

```
  ineg:=0;
  For i:=1 to m do
    For j:=1 to n do
      If a[i,j]<0 then
        Begin
          ineg:=i; jneg:=j;
          Goto 10
        End;
  10:
  If ineg>0 then
    Begin
      R:=a[ineg,jneg]; a[ineg,jneg]:=a[m,n];
      a[m,n]:=R;
    End;
  Печать   A
End.
```

На этом примере нужно еще раз подчеркнуть, что оператор **Break** производит выход за пределы лишь того цикла, где он сам находится. Следовательно, если заменить **Goto 10** на оператор **Break,** то произойдет лишь выход за пределы цикла по *j,* а цикл по *i* работу продолжит, что в общем случае приведет к получению неверных результатов.

2.2. В прямоугольной матрице поменять местами строки, содержащие максимальный и минимальный ее элементы.

По условию задачи не требуется определять положение максимального и минимального элементов в строке. Поэтому в программе используются лишь переменные $i_{\max}$ и $i_{\min}$, отмечающие номера строк, содержащих эти элементы. Если $i_{\max} = i_{\min}$, обмен строк не производится (нет смысла менять строку саму с собой).

```
Program Task202;
Const  Mmax = 20; Nmax = 15;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
Var  i,j,m,n,
     imax,imin : byte;
     Amax,Amin,R : real;
     A : Matrix;
Begin
  В в о д   m, n, A
  Amax:=a[1,1]; imax:=1;
  Amin:=a[1,1]; imin:=1;
  For i:=1 to m do              { Поиск в матрице }
    For j:=1 to n do            { максимального и }
      If a[i,j]>Amax then       { минимального }
        Begin                   { элементов }
          Amax:=a[i,j]; imax:=i;
        End
      Else
        If a[i,j]<Amin then
          Begin
            Amin:=a[i,j]; imin:=i;
          End;
  If imax<>imin then
    For j:=1 to n do            { Обмен строк матрицы }
```

```
      Begin
        R:=a[imin,j]; a[imin,j]:=a[imax,j];
        a[imax,j]:=R
      End;
  Печать   A
End.
```

2.3. Найти максимальный элемент среди элементов квадратной матрицы, расположенных выше главной диагонали, и минимальный элемент среди тех элементов, которые расположены ниже этой диагонали, после чего найденные элементы обменять местами.

Выпишем в явном виде элементы, расположенные выше главной диагонали (верхний треугольник квадратной матрицы):

```
a[1,2]  a[1,3]  a[1,4]  ...  a[1,n-1]   a[1,n]
        a[2,3]  a[2,4]  ...  a[2,n-1]   a[2,n]
                a[3,4]  ...  a[3,n-1]   a[3,n]
               ..............................
                           a[n-2,n-1] a[n-2,n]
                                      a[n-1,n]
```

Для произвольного элемента $a_{i,j}$ номер строки изменяется от 1 до n-1, т.е. $i = 1 .. (n-1)$; номер столбца - от i+1 до n, т.е. $j = (i+1) .. n$.

Элементы нижнего треугольника:

```
a[2,1]
a[3,1]   a[3,2]
a[4,1]   a[4,2]  a[4,3]
.....................
a[n,1]   a[n,2]  a[n,3]  ...  a[n,n-2]  a[n,n-1]
```

Здесь $i = 2 .. n$, $j = 1 .. (i-1)$.

```
Program  Task203;
Const  Nmax = 20;
Type  Matrix = array[1..Nmax,1..Nmax] of real;
Var  i,j,n,imax,imin,
     jmax,jmin : byte;
     Amax,Amin : real;
     A : Matrix;
Begin
  В в о д   n, A
  Amax:=a[1,2]; imax:=1; jmax:=2;
  For i:=1 to n-1 do              { Поиск максимального }
    For j:=i+1 to n do            { элемента выше }
      If a[i,j]>Amax then         { главной диагонали }
        Begin
          Amax:=a[i,j]; imax:=i; jmax:=j
        End;
  Amin:=a[2,1]; imin:=2; jmin:=1;
  For i:=3 to n do                { Поиск минимального }
    For j:=1 to i-1 do            { элемента ниже }
      If a[i,j]<Amin then         { главной диагонали }
```

```
        Begin
          Amin:=a[i,j]; imin:=i; jmin:=j
        End;
  a[imin,jmin]:=Amax; a[imax,jmax]:=Amin;
  Печать   A
End.
```

2.4. Среди диагоналей квадратной матрицы, параллельных главной и расположенных выше нее (включая главную диагональ), найти такую, сумма модулей элементов которой максимальная.

Запишем в отдельных строках индексы элементов, входящих в рассматриваемые диагонали:

```
1,1    2,2    3,3    4,4   ...   n-2,n-2   n-1,n-1   n,n
1,2    2,3    3,4    4,5   ...   n-2,n-1   n-1,n
1,3    2,4    3,5    4,6   ...   n-2,n
.....................................
1,n-1  2,n
1,n
```

Будем считать номер диагонали k равным номеру столбца ее начального элемента: $k = 1..n$. Тогда индексы элементов, входящих в k-ую диагональ, равны $j = k .. n$; $i = j - k + 1$.

```
Program Task204;
Const Nmax = 20;
Type  Matrix = array[1..Nmax,1..Nmax] of real;
Var   i,j,k,kmax,n : byte;
      S,SumMax : real;
      A : Matrix;
Begin
  В в о д   n, A
  SumMax:=0;
  For k:=1 to n do
    Begin
      S:=0;
      For j:=k to n do
        Begin
          i:=j-k+1; S:=S+abs(a[i,j]);
        End;
      If S>SumMax then
        Begin
          SumMax:=S; kmax:=k
        End;
    End;
  П е ч а т ь  SumMax, kmax
End.
```

Примечание. Нетрудно заметить, что для диагоналей, параллельных главной, соблюдается следующая закономерность между индексами их элементов: $j - i = k - 1$, где k - номер диагонали.

Используя эту закономерность, программу можно организовать следующим образом:

- объявить массив S для сумм модулей элементов диагоналей;
- обнулить элементы массива S;

- перебирая все элементы матрицы, расположенные выше главной диагонали и на главной диагонали (по циклу $i = 1 .. n$, $j = i .. n$), модуль очередного элемента $a_{i,j}$ добавлять к сумме S_{j-i+1};

- определить порядковый номер максимального элемента в одномерном массиве *S*.

2.5. Побочной диагональю квадратной матрицы называют диагональ, соединяющую левый нижний угол матрицы с ее верхним правым углом. Среди диагоналей, параллельных побочной и расположенных ниже нее, найти такую, среднее арифметическое значение элементов которой максимальное.

Индексы элементов, входящих в рассматриваемые диагонали:

```
n,2    n-1,3   n-2,4   ...   4,n-2   3,n-1   2,n
n,3    n-1,4   n-2,5   ...   4,n-1   3,n
n,4    n-1,5   n-2,6   ...   4,n
.....................................
n,n-1  n-1,n
n,n
```

Будем считать номер диагонали k равным номеру столбца ее начального элемента: $k = 2..n$. Тогда индексы элементов, входящих в *k*-ую диагональ, равны $i = n .. k$; $j = n - i+k$. Количество элементов в диагонали равно $n – k + 1$.

```
Program Task205;
Const  Nmax = 20;
Type  Matrix = array[1..Nmax,1..Nmax] of real;
      Ar = array[2..Nmax] of real;
Var  i,j,k,kmax,n : byte;
     ArMax : real;
     A : Matrix;
     Sar : Ar;
Begin
  В в о д  n,A
  For k:=2 to n do            { Формирование массива Sar, }
    Begin                     { содержащего средние }
      Sar[k]:=0;              { арифметические значения }
      For i:=n downto k do    { элементов диагоналей }
        Begin
          j:=n-i+k; Sar[k]:=Sar[k]+a[i,j];
        End;
      Sar[k]:=Sar[k]/(n-k+1);
    End;
  ArMax:=Sar[2]; kmax:=2;
  For k:=3 to n do            { Определение }
    If Sar[k]>ArMax then      { максимального }
      Begin                   { значения в массиве Sar }
        ArMax:=Sar[k]; kmax:=k
      End;
  П е ч а т ь  ArMax, kmax
End.
```

Примечание. Для диагоналей, параллельных побочной, соблюдается следующая закономерность между индексами их элементов: $i + j = n + k$, где k - номер диагонали. Исполь-

зуя эту закономерность, программу можно организовать аналогично рекомендациям, отраженным в примечании к задаче 2.4.

2.6. Расположить элементы прямоугольной матрицы в обратном порядке.

В программе Task206 сначала переставляются в обратном порядке строки матрицы, а затем - ее столбцы.

```
Program Task206;
Const  Mmax = 20; Nmax = 15;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
Var  i,j,k,m,n : byte;
     R : real;
     A : Matrix;
Begin
  В в о д   m, n, A
  i:=1; k:=m;              { Перестановка строк матрицы }
  While i<k do             { в обратном порядке }
    Begin
      For j:=1 to n do
        Begin
          R:=a[i,j]; a[i,j]:=a[k,j]; a[k,j]:=R;
        End;
      Inc(i); Dec(k);
    End;
  j:=1; k:=n;              { Перестановка столбцов матрицы }
  While j<k do             { в обратном порядке }
    Begin
      For i:=1 to m do
        Begin
          R:=a[i,j]; a[i,j]:=a[i,k]; a[i,k]:=R;
        End;
      Inc(j); Dec(k);
    End;
  Печать   A
End.
```

2.7. Найти максимальное из чисел, встречающихся в целочисленной прямоугольной матрице более одного раза.

Функция *YesNo* в программе Task207 определяет, является ли элемент с индексами *i* и *j* повторяющимся в матрице *A*. Если *YesNo(i, j) = true* и *imax* = 0 (повторяющийся элемент встретился впервые), то переменные $i_{\max}$, $j_{\max}$ запоминают индексы, а переменная $a_{\max}$ - значение этого элемента. При *imax* > 0 изменение значений переменных $i_{\max}$, $j_{\max}$, $a_{\max}$ производится, если новый повторяющийся элемент превышает значение $a_{\max}$.

```
Program Task207;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  m,n,                  { размер матрицы }
     imax,jmax,            { позиция макс.элемента }
     i,j : byte;
```

```
     Amax : integer;       { значение макс.элемента }
     A : Matrix;           { исходная матрица }
{ ---------------------------------------- }
Function YesNo(ik,jk:byte):boolean;
{ YesNo = true, если элемент a[ik,jk] повторяющийся }
Var  i,j : byte;
     R : integer;
Begin
  R:=a[ik,jk]; YesNo:=false;
  For i:=1 to m do
    For j:=1 to n do
      If (R=a[i,j]) and (i<>ik) and (j<>jk) then
        Begin
          YesNo:=true; Exit
        End;
End { YesNo };
{ ---------------------------------------- }
Begin
  imax:=0; jmax:=0; Amax:=0;
  For i:=1 to m do
    For j:=1 to n do
      Begin
        If YesNo(i,j) then
          If imax=0 then
            Begin
              imax:=i; jmax:=j;
              Amax:=a[i,j];
            End
          Else
            If a[i,j]>Amax then
              Begin
                imax:=i; jmax:=j; Amax:=a[i,j];
              End
      End;
  Writeln('imax = ',imax,'  jmax = ',jmax,'  Amax = ',Amax);
End.
```

2.8. Элементам целочисленной квадратной матрицы присвоить значения натурального ряда чисел в порядке прохождения правосторонней спирали, начиная с заданного углового элемента.

Например, для матрицы 6-го порядка с начальным элементом (1,6) имеем:

```
16  17  18  19  20   1
15  30  31  32  21   2
14  29  36  33  22   3
13  28  35  34  23   4
12  27  26  25  24   5
11  10   9   8   7   6
```

```
Program Task208;
{ Нумерация углов матрицы: 1 - левый верхний; 2 - правый }
{ верхний; 3 - правый нижний; 4 - левый нижний           }
Const  Nmax = 30;
Type   Matrix = array[1..Nmax,1..Nmax] of word;
Var    n,        { размер матрицы A }
```

```
      i,j,
      k,         { условный номер угла матрицы }
      is,        { индекс нач.эл-та формируемой части строки }
      ik,        { то же для конечного элемента строки }
      js,        { индекс нач.эл-та формируемой части столбца }
      jk : byte;   { то же для конечного элемента столбца }
      Number : word;  { заполняемое значение }
      Cond : boolean;
      A : Matrix;
{ ------------------------------------------- }
Procedure ToRight;
{ Движение вправо по строке }
Var  j : byte;
Begin
  For j:=js to jk do
    Begin
      a[is,j]:=Number; Inc(Number);
    End;
End { ToRight };
{ ------------------------------------------- }
Procedure ToDown;
{ Движение вниз по столбцу }
Var  i : byte;
Begin
  For i:=is to ik do
    Begin
      a[i,jk]:=Number; Inc(Number);
    End;
End { ToDown };
{ ------------------------------------------- }
Procedure ToLeft;
{ Движение влево по строке }
Var  j : byte;
Begin
  For j:=jk downto js do
    Begin
      a[ik,j]:=Number; Inc(Number);
    End;
End { ToLeft };
{ ------------------------------------------- }
Procedure ToUp;
{ Движение вверх по столбцу }
Var  i : byte;
Begin
  For i:=ik downto is do
    Begin
      a[i,js]:=Number; Inc(Number);
    End;
End { ToUp };
{ ------------------------------------------- }
Begin
  Ввод размера матрицы n и номера начального угла k
  Number:=1; Cond:=true;
  is:=1; js:=1; ik:=n; jk:=n;
  While Cond do
    Begin
      Case k of
```

```
          1 : Begin
                ToRight; Inc(is);
              End;
          2 : Begin
                ToDown; Dec(jk);
              End;
          3 : Begin
                ToLeft; Dec(ik);
              End;
          4 : Begin
                ToUp; Inc(js);
              End;
        end;
        Inc(k);
        If k>4 then k:=1;
        If (ik<is) or (jk<js) then
          Cond:=false;
      End;
    Печать  n, A
End.
```

2.9. Элементы целочисленной прямоугольной матрицы *A* строго упорядочены по возрастанию:

```
a[1,1] < a[1,2] < a[1,3] < ... < a[1,n] < a[2,1] < ...
... < a[2,n] < a[3,1] < ... < a[m,n]
```

Найти элемент, равный заданному числу b, и отпечатать его индексы. Число действий в решении должно быть порядка $m+n$, а не $m \cdot n$.

Последнее ограничение означает, что последовательный перебор всех элементов матрицы не допускается.

Вариант 1.

Если $b < a_{1,1}$ или $b > a_{m,n}$, то матрица A не содержит элемента, равного значению b. В противном случае будем сравнивать переменную b с первым элементом каждой строки матрицы. Если $b < a_{i,1}$, $i = 2 .. m$, то элемент, равный b, может находиться в строке i-1. Тогда производится последовательный просмотр элементов (i-1)-ой строки. Если условие $b < a_{i,1}$ ни разу не было выполнено, то просматриваются элементы *m*-ой строки.

Общее количество сравнений в варианте 1 не превышает $m+n$.

```
Program Task209a;
Label  10;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  m,n,                { размер матрицы }
     is,js,              { позиция эл-та, равного b }
     i,j  : byte;
     b : integer;
     Cond : boolean;
     A : Matrix;         { исходная матрица }
Begin
  В в о д   m, n, A, b
  is:=0; js:=0;
  If (b<a[1,1]) or (b>a[m,n]) then
```

```
    Goto 10;
  For i:=2 to m do
    If b<a[i,1] then
      Begin
        For j:=1 to n do
          If b=a[i-1,j] then
            Begin
              is:=i-1; js:=j;  { найден элемент, равный b }
              Goto 10
            End;
        Goto 10;          { в строке нет элемента, равного b }
      End;
  For j:=1 to n do        { просмотр эл-тов последней строки }
    If b=a[m,j] then
      Begin
        is:=i-1; js:=j; Goto 10
      End;
  10:
  Writeln('is = ',is,'  js = ',js);
End.
```

Вариант 2. Более эффективным является двоичный поиск, для которого среднее количество сравнений равно $\ln(n)/\ln(2)$ (см.Task118). Однако алгоритм двоичного поиска, использованный в задаче 1.18, определен лишь по отношению к одномерному массиву.

Алгоритмы, разработанные для одномерного массива (двоичный поиск, сортировка и др.), могут быть применены по отношению к матрице, если ее неявно представить в виде одномерного массива.

По индексам i и j элемента матрицы $a_{i,j}$ можно определить его порядковый номер $k = (i-1)n + j$, где n - количество элементов в строке матрицы (количество столбцов). Тогда появляется возможность обработки матрицы как одномерного массива.

В программе требуется решать также обратную задачу: по порядковому номеру k определить индексы соответствующего элемента $a_{i,j}$. Для этого можно использовать формулы

$$i = (k\text{-}1)\ \textbf{\textit{div}}\ n + 1; \qquad j = k - (i-1)n\ .$$

```
Program Task209b;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  m,n,                 { размер матрицы }
     is,js,               { позиция элемента, равного b }
     i,j  : byte;
     k,k1,k2 : word;
     b : integer;
     A : Matrix;          { исходная матрица }
Begin
  В в о д   m, n, A, b
  k1:=1; k2:=m*n; is:=0; js:=0;
  While k1<=k2 do
    Begin
      k:=(k1+k2) div 2;
      i:=(k-1) div n + 1; j:=k-(i-1)*n;
      If b=a[i,j] then
        Begin
          is:=i; js:=j; Break
```

```
          End
        Else
          If b<a[i,j] then
            k2:=k-1
          Else
            k1:=k+1;
      End;
    Writeln('is = ',is,'    js = ',js);
End.
```

2.10. Элементы каждой строки прямоугольной матрицы сдвинуть циклически вправо, не затрагивая при этом положения максимального элемента.

Например, для строки

6 12 -4 17 1 0 9 11 14

получим

14 6 12 17 -4 1 0 9 11

```
Program Task210;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
Var  i,j,jmax,m,n : byte;
     Amax,R : real;
     A : Matrix;
Begin
  В в о д   m, n, A
  For i:=1 to m do              { Перебор строк матрицы }
    Begin
      Amax:=a[i,1]; jmax:=1;    { Определение положения }
      For j:=2 to n do          { макс.элемента в i-ой строке }
        If a[i,j]>Amax then
          Begin
            Amax:=a[i,j]; jmax:=j
          End;
        If jmax=1 then          { Максимальный элемент - }
          Begin                 { первый в строке }
            R:=a[i,n];
            For j:=n downto 3 do
              a[i,j]:=a[i,j-1];
            a[i,2]:=R;
          End
        Else
          If jmax=n then        { Максимальный элемент - }
            Begin               { последний в строке }
              R:=a[i,n-1];
              For j:=n-1 downto 2 do
                a[i,j]:=a[i,j-1];
              a[i,1]:=R;
            End
          Else                 { Максимальный элемент занимает }
            Begin              { промежуточное положение }
              R:=a[i,n];       { в строке }
              For j:=n downto jmax+2 do
                a[i,j]:=a[i,j-1];
              a[i,jmax+1]:=a[i,jmax-1];
              For j:=jmax-1 downto 2 do
```

```
                a[i,j]:=a[i,j-1];
              a[i,1]:=R;
            End;
    End;
  П е ч а т ь   A
End.
```

2.11. Нулевые элементы каждого столбца прямоугольной матрицы переместить в начало этого же столбца, сохранив без изменения последовательность расположения остальных его элементов.

```
Program Task211;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  i,j,iz,k,m,n : byte;
     A : Matrix;
Begin
  В в о д   m, n, A
  For j:=1 to n do
    Begin
      k:=0;
      For i:=1 to m do
        If a[i,j]=0 then
          Begin
            Inc(k);                    { переменная k указывает }
            For iz:=i downto k+1 do    { позицию, в которую дол-}
              a[iz,j]:=a[iz-1,j];      { жен быть перенесен  ну-}
            a[k,j]:=0;                 { левой элемент }
          End;
    End;
  П е ч а т ь   A
End.
```

2.12. Элементы $a_{i,j}$ каждого столбца прямоугольной матрицы A сгруппировать в порядке возрастания их расстояния d_i от среднего арифметического значения S элементов этого столбца, где $d_i = \left| S - a_{i,j} \right|$.

```
Program Task212;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
      Vector = array[1..Mmax] of real;
Var  i,j,k,m,n : byte;
     S : real;
     Cond : boolean;
     A : Matrix;
     D : Vector;
Begin
  В в о д   m, n, A
  For j:=1 to n do
    Begin
      S:=0;                    { Определение среднего }
      For i:=1 to m do         { арифметического значения S }
```

```
        S:=S+a[i,j];          { элементов j-го столбца }
      S:=S/m;
      For i:=1 to m do        { Определение расстояний d }
        d[i]:=abs(S-a[i,j]); { для элементов столбца }
      Cond:=true; k:=m-1;
      While Cond do           { Группировка элементов }
        Begin                 { столбца по возрастанию }
          Cond:=false; k:=m-1;{ параметра d }
          For i:=1 to k do
            If d[i]>d[i+1] then
              Begin
                S:=a[i,j]; a[i,j]:=a[i+1,j]; a[i+1,j]:=S;
                S:=d[i]; d[i]:=d[i+1]; d[i+1]:=S;
                Cond:=true;
              End;
          Dec(k);
        End;
    End;
  П е ч а т ь  A
End.
```

2.13. Элементы, расположенные на периметре прямоугольной матрицы, сдвинуть по часовой стрелке на один шаг.

```
Program Task213;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
Var  i,j,m,n : byte;
     R : real;
     A : Matrix;
Begin
  В в о д  m, n, A
  R:=a[1,1];            { Сохранение левого верхнего эл-та }
  For i:=1 to m-1 do    { Сдвиг левого столбца вверх }
    a[i,1]:=a[i+1,1];
  For j:=1 to n-1 do    { Сдвиг нижней строки влево }
    a[m,j]:=a[m,j+1];
  For i:=m downto 2 do { Сдвиг правого столбца вниз }
    a[i,n]:=a[i-1,n];
  For j:=n downto 3 do { Сдвиг первой строки вправо }
    a[1,j]:=a[1,j-1];
  a[1,2]:=R;            { Запись сохраненного элемента }
  П е ч а т ь  A
End.
```

2.14. Допустимым преобразованием матрицы называют перестановку двух строк или двух столбцов. Для заданной прямоугольной матрицы с помощью допустимых преобразований добиться того, чтобы один из элементов матрицы, обладающий наибольшим по модулю значением, располагался в левом верхнем углу матрицы.

```
Program Task214;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  i,j,imax,jmax,m,n : byte;
```

```
     R,Amax : integer;
     A : Matrix;
Begin
  В в о д   m, n, A
  Amax:=abs(a[1,1]);        { Определение положения }
  imax:=1; jmax:=1;         { максимального элемента }
  For i:=1 to m do
    For j:=1 to n do
      If abs(a[i,j])>Amax then
        Begin
          Amax:=abs(a[i,j]); imax:=i; jmax:=j;
        End;
  If imax>1 then            { Обмен элементов строк }
    For j:=1 to n do        { с номерами 1 и imax }
      Begin
        R:=a[1,j]; a[1,j]:=a[imax,j]; a[imax,j]:=R;
      End;
  If jmax>1 then            { Обмен элементов столбцов }
    For i:=1 to m do        { с номерами 1 и jmax }
      Begin
        R:=a[i,1]; a[i,1]:=a[i,jmax]; a[i,jmax]:=R;
      End;
  П е ч а т ь   imax, jmax, Amax, A
End.
```

2.15. Найти максимальный среди всех элементов тех строк матрицы, которые упорядочены (либо по возрастанию, либо по убыванию).

Числовая последовательность $x_1, x_2, ..., x_n$ считается упорядоченной по возрастанию, если

$$x_1 \le x_2 \le ... \le x_n$$

(в отличие от строгой упорядоченности, при которой должны соблюдаться отношения $x_1 < x_2 < ... < x_n$).

Функция *YesNo* в программе Task215 для k-ой строки матрицы вначале производит поиск первой пары неравных друг другу элементов. При этом булевской переменной b присваивается значение *true*, если второй элемент этой пары больше первого элемента, и значение *false* в противном случае. После этого проверяется, нет ли нарушения установленного порядка между остальными парами элементов k-ой строки. Если в строке все элементы одинаковы, то такая строка считается упорядоченной.

```
Program Task215;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var i,j,imax,jmax,m,n : byte;
    Amax : integer;
    A : Matrix;
{ ---------------------------------- }
Function YesNo(p:byte): boolean;
{ Определение упорядоченности элементов p-ой строки }
Var  j,k : byte;
     Cond : boolean;
Begin
  Cond:=true;
```

```
    For j:=1 to n-1 do
      If a[p,j]<>a[p,j+1] then
        Begin
          Cond:=a[p,j]<a[p,j+1]; Break
        End;
    YesNo:=true;
    If Cond then
      For j:=k+1 to n-1 do
        If a[p,j]>=a[p,j+1] then
          Begin
            YesNo:=false; Exit
          End;
    If not Cond then
      For j:=k+1 to n-1 do
        If a[p,j]<=a[p,j+1] then
          Begin
            YesNo:=false; Exit
          End;
  End { YesNo };
  { ---------------------------------- }
  Begin
    В в о д   m, n, A
    imax:=0; jmax:=0; Amax:=0;
    For i:=1 to m do
      If YesNo(i) then
        If imax=0 then              { встретилась первая }
          Begin                     { упорядоченная строка }
            Amax:=a[i,1]; imax:=i; jmax:=1;
            For j:=2 to n do
              If a[i,j]>Amax then
                Begin
                  Amax:=a[i,j]; jmax:=j;
                End;
          End
        Else
          For j:=1 to n do          { рассматривается очередная }
            If a[i,j]>Amax then     { упорядоченная строка }
              Begin
                Amax:=a[i,j]; imax:=i; jmax:=j;
              End;
    Writeln('imax = ',imax,'  jmax = ',jmax,'  Amax = ',Amax);
  End.
```

2.16. В прямоугольной матрице рассмотреть квадратные подматрицы размерностью 1, 2, 3, ... , причем для всех подматриц левым верхним элементом является элемент исходной матрицы с индексами (1,1). Определить номер подматрицы, среднее арифметическое значение элементов которой является максимальным.

```
Program Task216;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  i,j,k,kmax,m,n,p : byte;
     S,Smax : real;
     A : Matrix;
Begin
  В в о д   m, n, A
```

```
  If m>n then                { p = min(m,n) }
    p:=n
  Else
    p:=m;
  Smax:=a[1,1]; kmax:=1;
  For k:=2 to p do
    Begin
      S:=0;
      For i:=1 to k do
        For j:=1 to k do
          S:=S+a[i,j];
      S:=S/sqr(k);
      If S>Smax then
        Begin
          Smax:=S; kmax:=k;
        End;
    End;
  П е ч а т ь   kmax, Smax
End.
```

2.17. Соседями элемента матрицы называют элементы, смежные с ним по вертикали или по горизонтали. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной прямоугольной вещественной матрицы. Учесть, что локальный минимум не может находиться на периметре матрицы.

```
Program Task217;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
Var  i,j,m,n,
     Count : byte;        { счетчик локальных минимумов }
     R : real;
     Cond : boolean;
     A : Matrix;
Begin
  В в о д   m, n, A
  Count:=0;
  For i:=2 to m-1 do
    For j:=2 to n-1 do
      Begin
        R:=a[i,j];
        Cond:=(R<a[i,j-1]) and (R<a[i,j+1]) and
              (R<a[i-1,j]) and (R<a[i+1,j]);
        If Cond then
          Inc(Count);
      End;
  Writeln('Count = ',Count);
End.
```

2.18. Известно, что в прямоугольной целочисленной матрице два и только два элемента равны между собой. Определить их индексы.

Указание. В программе не должно быть двухкратного сравнения одних и тех же элементов.

Здесь, как и в задаче 2.9, нужно рассматривать матрицу как одномерный массив, индексация которого определяется порядковыми номерами элементов матрицы.

```
Program Task218;
Label  10;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  i1,j1,          { индексы эл-та с порядковым номером k1 }
     i2,j2,          { индексы эл-та с порядковым номером k2 }
     m,n : byte;     { кол-во строк и столбцов матрицы }
     k1,k2,          { порядковые номера элементов матрицы }
     p : word;       { количество элементов матрицы }
     Cond : boolean;
     A : Matrix;     { исходная матрица }
Begin
  В в о д   m, n, A
  p:=m*n; Cond:=false;
  For k1:=1 to p-1 do
    Begin
      i1:=(k1-1) div n + 1; j1:=k1-(i1-1)*n;
      For k2:=k1+1 to p do
        Begin
          i2:=(k2-1) div n + 1; j2:=k2-(i2-1)*n;
          If a[i1,j1]=a[i2,j2] then
            Begin
              Cond:=true; Goto 10;
            End;
        End;
    End;
  10:
  If Cond then
    Writeln('i1=',i1,'  j1=',j1,'  i2=',i2,'  j2=',j2)
  Else
    Writeln('В матрице нет одинаковых элементов');
End.
```

2.19. Для заданной целочисленной квадратной матрицы A, имеющей $n \times n$ элементов, проверить, имеет ли место хотя бы однократное совпадение k-ой строки и k-го столбца ($k = 1..n$).

```
Program Task219;
Const  Nmax = 40;
Type  Matrix = array[1..Nmax,1..Nmax] of integer;
Var  j,k,n : byte;
     Cond : boolean;
     A : Matrix;
Begin
  В в о д   m, n, A
  k:=0;
  Repeat
    Cond:=true; Inc(k);
    For j:=1 to n do
      If a[k,j]<>a[j,k] then       { Выход из цикла просмотра }
        Begin                      { строки при обнаружении }
          Cond:=false; Break       { несовпадающих элементов }
```

```
        End;
    Until Cond or (k=n);
    If Cond then
      Writeln('Совпадение строки и столбца с номером ',k)
    Else
      Writeln('Нет совпадающих строки и столбца');
End.
```

Цикл ***Repeat*** заканчивает свою работу, если найдены совпадающие строка и столбец (*Cond* = *true*) или просмотрены все строки и соответствующие им столбцы (*k* = *n*).

2.20. Две строки матрицы линейно зависимы, если одну из них можно получить из другой умножением на постоянный коэффициент. Определить, имеются ли в прямоугольной вещественной матрице линейно зависимые строки и указать номера первой пары таких строк.

В программе Task220 сравниваются *i*-ая и *k*-ая строки.

Если $a_{i,1} = 0$ (первый элемент *i*-ой строки), $i = 1, \ldots, m\text{-}1$, то *i*-ая строка не сравнивается с другими строками. Если $a_{k,1} = 0$ (первый элемент *k*-ой строки), $k = i+1, \ldots, m$, то пара строк $i - j$ не проверяется. В противном случае вычисляется коэффициент $R = \dfrac{a_{i,1}}{a_{k,1}}$.

Чтобы исключить возможность деления на нуль, в дальнейшем элементы *i*-ой и *k*-ой строк проверяются по условию $a_{i,j} = R\,a_{k,j}$, $j = 2..n$ (с учетом сравнения по ε - по условию $\left| a_{i,j} - R\,a_{k,j} \right| \le \varepsilon$).

```
Program Task220;
Label  10;
Const  eps = 0.001; Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of real;
Var  i,j,k,m,n : byte;
     R : real;
     Cond : boolean;
     A : Matrix;
Begin
  В в о д   m, n, A
  For i:=1 to m-1 do
    If a[i,1]<>0 then
      For k:=i+1 to m do
        If a[k,1]<>0 then
          Begin
            R:=a[i,1]/a[k,1]; Cond:=true;
            For j:=2 to n do
              If abs(a[i,j]-R*a[k,j])>eps then
                Cond:=false;
            If Cond then
              Goto 10;
          End;
  10:
  If Cond then
    Writeln('Линейно зависимы строки ',i,' и ',k)
  Else
```

```
    Writeln('В матрице нет линейно зависимых строк');
End.
```

2.21. Подсчитать количество строк целочисленной прямоугольной матрицы $A(m \times n)$, элементы которых являются перестановкой чисел 1, 2, ..., n.

```
Program Task221a;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  i,j,m,n : byte;
     A : Matrix;
{ ------------------------------- }
Function YesNo(k:byte):boolean;
{ true, если k-ая строка является перестановкой  1 .. n }
Var  j,nk : byte;
     R : integer;
     Cond : boolean;
Begin
  Cond:=true; nk:=n-1;        { Группировка строки }
  While Cond do               { по возрастанию }
    Begin
      Cond:=false;
      For j:=1 to nk do
        If a[k,j]>a[k,j+1] then
          Begin
            R:=a[k,j]; a[k,j]:=a[k,j+1]; a[k,j+1]:=R;
            Cond:=true;
          End;
      Dec(nk);
    End;
  YesNo:=true;                { Проверка равенства между }
  For j:=1 to n do            { значением элемента и его }
    If a[k,j]<>j then         { порядковым номером       }
      Begin
        YesNo:=false; Exit;
      End;
End { YesNo };
{ ------------------------------- }
Begin
  В в о д   m, n, A
  k:=0;
  For i:=1 to m do
    If YesNo(i) then
      Inc(k);
  Writeln('k = ',k);
End.
```

Функция *YesNo* группирует каждую строку матрицы *A* по возрастанию ее элементов. Это не всегда допустимо, поскольку при этом изменяются исходные данные. Поэтому выполним еще один вариант задачи, но без группировки элементов матрицы. В этом случае для каждой строки, содержащей *n* элементов, должны быть выполнены следующие условия:

- минимальный элемент должен быть равен 1;
- максимальный элемент должен быть равным значению *n*;
- в строке не должно быть повторяющихся элементов.

```
Program Task221b;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  i,j,m,n : byte;
     A : Matrix;
{ ------------------------------- }
Function YesNo(k:byte):boolean;
{ true, если k-ая строка является перестановкой  1 .. n }
Var  i,j : byte;
     amax,amin : integer;
Begin
  YesNo:=true; amin:=a[k,1]; amax:=a[k,1];
  For j:=2 to n do
    If a[k,j]>amax then
      amax:=a[k,j]
    Else
      If a[k,j]<amin then
        amin:=a[k,j];
  If (amin<>1) or (amax<>n) then
    Begin
      YesNo:=false; Exit
    End;
  For j:=1 to n-1 do
    For i:=j+1 to n do
      If a[k,j]=a[k,i] then
        Begin
          YesNo:=false; Exit
        End;
End { YesNo };
{ ------------------------------- }
Begin
  В в о д   m, n, A
  k:=0;
  For i:=1 to m do
    If YesNo(i) then
      Inc(k);
  Writeln('k = ',k);
End.
```

2.22. Определить, имеется ли в прямоугольной целочисленной матрице хотя бы одна симметричная строка, элементы которой строго монотонно изменяются от начала до ее середины (по возрастанию или по убыванию).

```
Program Task222;
Const  Mmax = 30; Nmax = 50;
Type  Matrix = array[1..Mmax,1..Nmax] of integer;
Var  i,k,
     m,n : byte;
     Cond : boolean;
     A : Matrix;
{ ------------------------------- }
Function SignRow(k:byte):boolean;
{ Проверка симметричности и строгой упорядоченности }
{ элементов k-ой строки                              }
Var  i,j : byte;
     CondSim,CondHigh : boolean;
```

```
Begin
  CondSim:=true; SignRow:=true; {Проверка симметричности}
  i:=1; j:=n;                   { элементов строки      }
  While i<j do
    If a[k,i]<>a[k,j] then
      Begin
        SignRow:=false; Exit
      End
     Else
      Begin
        Inc(i); Dec(j);
      End;
  CondHigh:=a[k,1]<a[k,2];      { Определение направления }
  If CondHigh then              { изменения }
    For j:=2 to (n div 2)-1 do  { Проверка строгой }
      If a[k,j]>=a[k,j+1] then  { упорядоченности по }
        Begin                   { возрастанию }
          SignRow:=false; Exit
        End;
  If not CondHigh then          { Проверка строгой упорядо- }
    For j:=1 to (n div 2)-1 do  { ченности по убыванию }
      If a[k,j]<=a[k,j+1] then
        Begin
          SignRow:=false; Exit
        End;
End { SignRow };
{ -------------------------------- }
Begin
  В в о д   m, n, A
  Cond:=false;
  For i:=1 to m do
    If SignRow(i) then
      Begin
        Cond:=true; k:=i; Break
      End;
  If Cond then
    Writeln('Заданная строка имеет номер ',k)
  Else
    Writeln('В матрице нет заданной строки');
End.
```

2.23. В заданной вещественной квадратной матрице сдвинуть каждую ее строку циклически влево или вправо таким образом, чтобы максимальный элемент строки был расположен на главной диагонали. Вычислить след матрицы до и после ее преобразования.

Циклический сдвиг строки предполагает такую перестановку ее элементов, при которой не нарушается их исходное относительное расположение.

Пусть, например, третья строка матрицы имеет вид

5.8 -4.4 10.0 8.7 14.6 11.9

Здесь максимальный элемент $a_{3,5}$ расположен справа на две позиции от элемента главной диагонали $a_{3,3}$. Следовательно, необходимо дважды выполнить циклический сдвиг строки влево, после чего она примет вид

10.0 8.7 14.6 11.9 5.8 -4.4

След матрицы - это сумма элементов ее главной диагонали.

Программа Task223 в каждой *i*-ой строке матрицы ($i = 1..n$) определяет максимальный элемент $a_{\max}$ и его индекс $j_{\max}$, после чего вычисляет параметр $k = i - jmax$, указывающий положение элемента $a_{\max}$ по отношению к главной диагонали. Если $k > 0$ (максимальный элемент находится слева от элемента главной диагонали), то выполняется циклический сдвиг строки вправо до тех пор, пока параметр k не примет нулевое значение (в каждом цикле значение k уменьшается на 1). Последнее означает, что максимальный элемент переставлен на место элемента главной диагонали. При $k < 0$ циклический сдвиг производится влево.

```
Program Task223;
Const  Nmax = 30;
Type  Matrix = array[1..Nmax,1..Nmax] of real;
Var  i,j,n,
     jmax : byte;       { позиция макс.элемента в строке }
     k : shortint;      { разница между положениями Amax }
                        { и элемента a[i,i] }
     Amax,              { максимальный элемент в строке }
     Buf,               { буферная переменная }
     Trace : real;      { след матрицы }
     A : Matrix;        { обрабатываемая матрица }
{ ------------------------------------------------- }
Function TraceMatrix:real;
{ Вычисление следа матрицы }
Var  i : byte;  R : real;
Begin
  R:=0;
  For i:=1 to n do
    R:=R+a[i,i];
  TraceMatrix:=R;
End { TraceMatrix };
{ ------------------------------------------------- }
Begin
  В в о д  и  п е ч а т ь   n, A
  Trace:=TraceMatrix;
  Writeln('След исходной матрицы Trace = ',Trace:8:2);
  For i:=1 to n do
    Begin
      Amax:=a[i,1]; jmax:=1;     { Определение положения }
      For j:=2 to n do           { максимального элемента }
        If a[i,j]>Amax then      { i-ой строки }
          Begin
            Amax:=a[i,j]; jmax:=j
          End;
      k:=i-jmax;
      If k>0 then
        While k>0 do             { Циклический сдвиг }
          Begin                  { строки вправо }
            Buf:=a[i,n];
            For j:=n downto 2 do
              a[i,j]:=a[i,j-1];
            a[i,1]:=Buf; Dec(k);
          End
      Else
        If k<0 then
          While k<0 do           { Циклический сдвиг }
            Begin                { строки влево }
```

```
                Buf:=a[i,1];
                For j:=1 to n-1 do
                  a[i,j]:=a[i,j+1];
                a[i,n]:=Buf;  Inc(k);
              End;
    End;
  П е ч а т ь   n, A
  Trace:=TraceMatrix;
  Writeln('След преобразованной матрицы Trace = ',Trace:8:2);
End.
```

Задания для самостоятельной работы

2.24. В прямоугольной матрице каждый нулевой элемент заменить средним арифметическим значением ненулевых элементов той строки, в которой расположен данный нулевой элемент. Если в строке несколько нулевых элементов, то они должны быть заменены одним и тем же значением.

2.25. В прямоугольной матрице определить количество столбцов, содержащих только числа одного знака (положительные или отрицательные) и не содержащих нулевых элементов.

2.26. Для прямоугольной матрицы найти минимальный из положительных и максимальный из отрицательных элементов, после чего обменять их местами. Нулевые элементы не учитывать.

2.27. В прямоугольной целочисленной матрице обменять местами максимальный по модулю и минимальный по модулю четные элементы.

2.28. Для каждого столбца прямоугольной матрицы подсчитать сумму входящих в него элементов и определить, имеются ли столбцы с одинаковой суммой.

2.29. В прямоугольной матрице часть элементов имеют нулевое значение. Заменить каждый нулевой элемент суммой смежных ему элементов (по горизонтали и вертикали). Формирование новой матрицы выполнять в буферном массиве.

2.30. Если максимальный элемент i-ой строки прямоугольной матрицы ($i = 1..m$) больше суммы остальных элементов данной строки, а модуль минимального элемента меньше суммы остальных элементов строки, то заменить максимальный и минимальный элементы полусуммой их значений.

2.31. При однократном просмотре прямоугольной матрицы определить три минимальных элемента, после чего переставить их в обратном порядке.

2.32. Дана квадратная вещественная матрица. Определить отдельно сумму S_1 элементов, расположенных выше побочной диагонали, и сумму S_2 элементов, расположенных ниже этой диагонали. Если эти суммы не равны, то ко всем элементам, образующим меньшую сумму, добавить такое значение, чтобы суммы S_1 и S_2 оказались равными.

2.33. Дана квадратная вещественная матрица. Определить отдельно количество k_1 отрицательных элементов, расположенных выше главной диагонали, и количество k_2 отрицательных элементов, расположенных ниже этой диагонали. Если $k_1 \neq k_2$, то изменить знаки определенного количества отрицательных элементов таким образом, чтобы выполнялось равенство $k_1 = k_2$.

2.34. Для каждого столбца прямоугольной целочисленной матрицы определить сумму модулей его элементов, а затем сгруппировать столбцы в порядке возрастания этих сумм.

2.35. В каждом столбце прямоугольной матрицы перенести максимальный по модулю элемент в последнюю позицию столбца, сдвинув при этом вверх расположенные после него элементы.

2.36. Среди диагоналей квадратной матрицы, параллельных главной и расположенных ниже нее, найти такую, сумма модулей элементов которой минимальна по сравнению с другими диагоналями.

2.37. Элементы, расположенные на периметре прямоугольной матрицы, сгруппировать в порядке возрастания, начиная с элемента, расположенного в левом верхнем углу матрицы. Обход элементов производить по часовой стрелке. Выполнить два варианта задачи: с использованием и без использования буферного массива.

2.38. Для заданной квадратной матрицы сформировать одномерный массив, элементы которого - максимумы элементов диагоналей, параллельных главной и расположенных ниже нее.

2.39. Для заданной квадратной матрицы найти минимум среди максимальных по модулю элементов диагоналей, параллельных побочной диагонали матрицы и расположенных выше нее.

2.40. В прямоугольной матрице, имеющей четное количество строк и четное количество столбцов, слева направо и сверху вниз пронумерованы квадраты из четырех элементов. Определить номер квадрата, для которого сумма входящих в него элементов максимальна.

Пример нумерации квадратов для матрицы 6×8 элементов:

1 2 3 4
5 6 7 8
9 10 11 12

2.41. В прямоугольной матрице, имеющей четное количество строк и четное количество столбцов, слева направо и сверху вниз пронумерованы квадраты из четырех элементов. Переставить квадраты в обратном порядке.

2.42. Найти минимальный среди модулей всех элементов тех столбцов матрицы, которые строго упорядочены (либо по возрастанию, либо по убыванию).

2.43. Определить, имеются ли среди столбцов прямоугольной целочисленной матрицы такие, которые составлены из попарно различных чисел.

2.44. Среди строк прямоугольной целочисленной матрицы, содержащих только нечетные по значению элементы, найти строку с минимальной суммой входящих в нее элементов.

2.45. Допустимым преобразованием матрицы называют перестановку двух строк или двух столбцов. Для заданной квадратной матрицы с помощью допустимых преобразований добиться того, чтобы один из элементов матрицы, обладающий наименьшим по модулю значением, располагался в правом верхнем углу матрицы.

2.46. В квадратной матрице $A(n \times n)$ найти наибольший элемент среди элементов, расположенных на главной и побочной диагоналях, после чего обменять его местами с наименьшим элементом, смежным с пересечением этих диагоналей.

Примечание. Если n - нечетное, то на пересечении указанных диагоналей находится один элемент, при четном n с точкой пересечения смежны 4 элемента.

2.47. Сформировать одномерный массив, получающийся при чтении квадратной матрицы по спирали, начиная с заданного углового элемента (против часовой стрелки).

2.48. Задана вещественная квадратная матрица A. Определить значение и положение двух элементов $a_{i,j}$ и $a_{k,l}$, для которых параметр $P = \sin\left| a_{i,j} - a_{k,l} \right|$ имеет максимальное значение. Для каждой пары элементов параметр P должен вычисляться только один раз.

2.49. Для заданной квадратной матрицы сформировать одномерный массив из элементов диагоналей, параллельных главной. Найти след матрицы, суммируя элементы одномерного массива.

2.50. Матрица, симметричная относительно главной диагонали, задана верхним треугольником, включающим главную диагональ, в виде одномерного массива по строкам. Восстановить исходную квадратную матрицу.

2.51. Квадратную матрицу повернуть на 180 градусов по часовой стрелке. При этом дополнительную матрицу не использовать.

2.52. В квадратной матрице поменять местами элементы, расположенные в верхней и нижней четвертях, ограниченных главной и побочной диагоналями (за исключением элементов этих диагоналей).

2.53. Квадратную матрицу называют ленточной шириной d, если все ее элементы, расположенные ниже d-ой поддиагонали и выше d-ой наддиагонали, равны нулю. Требуется заданную ленточную матрицу (значение d нужно определить в программе) представить в виде одномерного массива (наддиагонали d, d-1, ..., главная диагональ, поддиагонали 1, 2, ..., d) и, суммируя элементы этого массива, вычислить ее след.

2.54. По целочисленной прямоугольной матрице $A(m \times n)$ сформировать одномерный массив $B = (b_1,...,b_p)$, включив в него по убыванию k несовпадающих между собой наибольших элементов матрицы $A\,(1 \le k \le m \cdot n)$. Рекомендуется предварительно сгруппировать матрицу A по убыванию ее элементов, неявно представив ее как одномерный массив. В частном случае может иметь место $p < k$, если в матрице A много одинаковых элементов.

2.55. Элементами квадратной матрицы являются цифры 0, 1, ..., 9. Определить, имеются ли в матрице четыре смежных элемента по горизонтали (W - O) или вертикали (N - S), составляющие цифры текущего года. Здесь буквами N, S, W, O обозначены стороны света, определяющие направление просмотра. Если такие элементы обнаружены, отпечатать их индексы.

2.56. То же, но рассматривать диагонали SW – NO и NW – SO.

2.57. В каждой строке квадратной матрицы определить сумму S_1 элементов, расположенных слева от главной диагонали, и сумму S_2 элементов, расположенных справа от этой диагонали, после чего сгруппировать строки в порядке возрастания разности $dS = S_1 - S_2$.

2.58. В каждой строке прямоугольной матрицы, кроме первой и последней, определить сумму элементов, не принадлежащих периметру матрицы, после чего сгруппировать эти строки в порядке уменьшения указанных сумм, не затрагивая при этом элементы, расположенные на периметре матрицы.

2.59. В прямоугольной целочисленной матрице A расположены последовательно по строкам элементы одномерного массива X. Определить местоположение в матрице наиболее длинной монотонной последовательности этих элементов.

Примечание. Для монотонной серии выполняется условие

$$x_i \le x_{i+1} \le x_{i+2} \le \ldots \quad \text{или} \quad x_i \ge x_{i+1} \ge x_{i+2} \ge \ldots$$

2.60. Рассматривая элементы прямоугольной матрицы как правостороннюю спираль с началом в одном из угловых элементов, сгруппировать эти элементы в порядке возрастания. Выполнить два варианта задачи: с использованием и без использования буферного массива. Индексы углового элемента ввести с клавиатуры.

2.61. В квадратной матрице сгруппировать в порядке возрастания элементы каждой диагонали, параллельной главной, рассматривая эти элементы в направлении NW - SO. Реализовать два варианта программы: с использованием буферного массива и без него.

2.62. Элементами квадратной матрицы являются числа 0 и 1, причем все элементы первой строки равны 1. Определить, имеется ли хотя бы один путь, проходящий через единичные элементы, начинающийся в верхнем левом углу и заканчивающийся в нижней строке матрицы. При этом движение разрешается лишь вправо и вниз. Если путь найден, отпечатать индексы входящих в него элементов.

2.63. Среди квадратных подматриц $B(m\times m)$, содержащихся в заданной целочисленной матрице $A(n\times n)$, найти такую, для которой среднее арифметическое значение ее положительных элементов наибольшее. При этом верхний левый угловой элемент $b[1,1]$ должен находиться на главной диагонали исходной матрицы A.

2.64. Определить, является ли заданная целочисленная квадратная матрица ортонормированной, т.е. такой, в которой скалярное произведение каждой пары различных строк равно 0, а скалярное произведение каждой строки на себя равно 1.

Примечание. Например, матрица, у которой все элементы главной диагонали равны 1, а остальные элементы равны 0, является ортонормированной. При тестировании программы используйте другие виды ортонормированных матриц.

2.65. Определить, является ли заданная целочисленная квадратная матрица магическим квадратом, т.е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.

2.66. Задана целочисленная прямоугольная матрица. Удалить из нее нулевые строки и столбцы, если такие имеются.

3. Числа и системы счисления

3.1. Два натуральных числа называются дружественными, если каждое из них равно сумме всех делителей другого, кроме самого этого числа. Найти хотя бы одну пару дружественных чисел, лежащих в диапазоне $[m,n]$ ($m,n < 2^{30}$, $m \le n$). Например, в диапазоне 100 .. 300 такими числами являются 220 и 284 (220 :1+2+4+5+10+11+20+22+44+55+110 = 284; 284: 1+2+4+71+142 = 220).

Поиск дружественных чисел в программе Task301 производится по методу полного перебора. В диапазоне $m..n$ рассматриваются все пары чисел i и j, где i изменяется от m до $n-1$, а j - от $i+1$ до n.

```
Program Task301;
Var  i,j,k,m,n,
     s1,                { сумма делителей первого числа }
     s2 : longint;      { сумма делителей второго числа }
     Cond : boolean;
Begin
  Read(m,n); Cond:=false;
  For i:=m to n-1 do
    Begin
      s1:=1;
      For k:=2 to (i div 2) do
        If i mod k = 0 then     { k - делители числа i }
          s1:=s1+k;
      For j:=i+1 to n do
        Begin
          s2:=1;
          For k:=2 to (j div 2) do
            If j mod k = 0 then { k - делители числа j }
              s2:=s2+k;
            If (i=s2) and (j=s1) then
              Begin
                Writeln('i=',i,'  j=',j,'  s1=',s1,
                        ' s2 = ',s2);
                Cond:=true;
              End;
        End;
```

```
    End;
  If not Cond then
    Writeln('В диапазоне ',m,'..',n,' нет дружественных '
            'чисел');
End.
```

3.2. Подсчитать количество различных представлений заданного натурального числа N в виде суммы трех различных неотрицательных слагаемых. Представления, отличающиеся лишь порядком слагаемых, различными не считаются.

В программе формируются тройки чисел i, j, k, удовлетворяющие отношениям

$$i < j < k \quad \text{и} \quad i + j + k = N,$$

что обеспечивает их неповторяемость.

```
Program Task302;
Var i,j,k,N,Count : integer;
Begin
  Read(N); Count:=0;
  For i:=0 to N div 3 do
    For j:=i+1 to N-i do
      Begin
        k:=N-i-j;
        If k>j then
          Inc(Count);
      End;
  Writeln('N = ',N,'  Count = ',Count);
End.
```

Например, для N = 100 получим *Count* = 833 (0,1,99; 0,2,98; 0,3,97 и т.д,), для N = 1000 *Count* = 17797.

3.3. Переставить восьмеричные цифры целого числа M в обратном порядке и отпечатать полученное десятичное значение.

Будем считать число M положительным. Для решения задачи необходимо разработать два алгоритма: перевод целого числа из десятичной системы счисления в систему с основанием q (q = 8) и наоборот.

а) Перевод $10 \to q$.

Нам требуется сформировать массив восьмеричных цифр числа M . Произведем вначале оценку размера такого массива.

Максимальное значение целого числа в Паскаль-программе определяется типом *longint: MaxLongInt* = 2 147 483 647. Тогда количество цифр этого числа

$$n = \log_q(M) + 1 = \frac{\ln(M)}{\ln(q)} + 1 .$$

Для M = *MaxLongInt* получим:

$$q = 10 \quad n = 10$$
$$q = 8 \quad n = 11$$
$$q = 4 \quad n = 16$$

$$q = 2 \qquad n = 32$$

Будем записывать цифры числа M в системе q в виде

$$a_{n-1}a_{n-2}...a_1a_0,$$

что соответствует общепринятому представлению числа с n цифрами в произвольной системе счисления:

$$a_{n-1}\,q^{n-1} + a_{n-2}\,q^{n-2} + ... + a_1\,q + a_0\ .$$

Тогда программа преобразования целого числа ($10 \to q$) имеет вид

```
n:=0;
Repeat
  a[n]:=M mod q;  M:=M div q;
  Inc(n);
Until M = 0;
```

При первом выполнении цикла ***Repeat*** мы получаем младшую цифру a_0, затем - цифры a_1, a_2, ..., a_{n-1}. Значение n - общее количество цифр числа M в системе счисления q.

б) Перевод $q \to 10$.

Запись числа в форме приведенного выше выражения - это полином. Производим его вычисление по схеме Горнера:

$$P = (...(a_{n-1}q + a_{n-2})\,q + ... + a_1)\,q + a_0$$

```
P := 0
P := P*q + a[n-1]
P := P*q + a[n-2]
.................
P := P*q + a[0]
```

```
Program Task303;
Const  Nmax = 10;
Type  Ar = array[0..Nmax] of byte;
Var  M,M1,M2 : longint;
     i,j,n,q,r : byte;
     A : Ar;
Begin
  Read(M); q:=8;
  M1:=abs(M); n:=0;        { Формирование цифр числа }
  Repeat                   { M1 в системе счисления  }
    a[n]:=M1 mod q;  M1:=M1 div q; { с основанием  q = 8 }
    Inc(n);
  Until M1 = 0;
  Writeln('M = ',M,'  n = ',n);
  For i:=n-1 downto 0 do   { Печать восьмеричных цифр }
    Write(a[i],' ');       { числа M                   }
  Writeln;
  i:=0; j:=n-1;            { Перестановка цифр }
  While i<j do             { восьмеричного числа }
    Begin                  { в обратном порядке }
      r:=a[i]; a[i]:=a[j]; a[j]:=r;
      Inc(i); Dec(j);
    End;
  For i:=n-1 downto 0 do   { Печать восьмеричных }
    Write(a[i],' ');       { цифр числа M }
  Writeln;
  M2:=0;                   { Преобразование }
```

```
  For i:=n-1 downto 0 do  { восьмеричного числа }
    M2:=M2*q+a[i];        { в десятичную с/с }
  If M<0 then
    M2:=-M2;
  Writeln('M2 = ',M2);
End.
```

В результате работы программы будет напечатано:

```
M = 1000000000    n = 10
7 3 4 6 5 4 5 0 0 0
0 0 0 5 4 5 6 4 3 7
M2 = 1465631
```

3.4. Даны натуральные числа M и N, являющиеся числителем и знаменателем простой дроби. Сформировать первые 10 цифр дроби в системе счисления с основанием $2 \le q \le 10$ (с округлением) и отпечатать полученное десятичное значение.

Пусть нам дана десятичная дробь R. При переводе ее в систему счисления с основанием q цифры новой дроби $a_1 a_2 ... a_{m-1} a_m$ формируются путем последовательного умножения дробной части числа R на значение q до тех пор, пока не будет получено требуемое количество k цифр или же значение R не станет равным нулю (см. раздел «Системы счислени»):

```
i:=0;
While (i<=k) and (R>0) do
  Begin
    R:=R*q; Inc(i);
    a[i]:=Trunc(R);
    R:=R-a[i];
  End;
```

Чтобы округлить дробь до s цифр, необходимо к ее $(s+1)$-му разряду добавить половину цены разряда в системе с основанием q, т.е. значение $q/2$, после чего отбросить разряды $s+1,\ s+2$, ...

Для перевода дроби из системы с основанием q в десятичную систему представим ее в виде полинома:

$$R = a_1 q^{-1} + a_2 q^{-2} + a_3 q^{-3} + ... + a_{m-1} q^{-m+1} + a_m q^{-m} \ .$$

После этого вычисление значения R производим по схеме Горнера:

$$R = (...(a_m / q + a_{m-1}) / q + a_{m-2}) / q + ... + a_2) / q + a_1) / q \ .$$

```
Program Task304;
Type  Ar = array[1..11] of byte;
Var  M,N,p : word;
     i,q : byte;
     R : real;
     A : Ar;
Begin
  Read(M,N,q);
  If M=N then
    Writeln('Введены равные значения M = ',M,' и N = ',N)
  Else
    Begin
```

```
      If M>N then
        Begin
          p:=M; M:=N; N:=p
        End;
      Writeln('M = ',M,'  N = ',N,'  q = ',q);
      R:=M/N;
      Writeln('Исходное значение  R = ',R);
      For i:=1 to 11 do                { Формирование цифр }
        Begin                          { дробного числа в  }
          R:=R*q;                      { системе счисления }
          a[i]:=Trunc(R);              { с основанием q    }
          R:=R-a[i];
        End;
      Writeln('Цифры числа до округления :');
      For i:=1 to 11 do
        Write(a[i],' ');
      Writeln;
      a[11]:=a[11]+(q div 2);    { Округление дроби }
      If a[11]>=q then           { в системе q до }
        Begin                    { десяти цифр }
          a[11]:=a[11]-q; p:=1;
        End
      Else
        p:=0;
      i:=10;
      While (i>0) and (p=1) do   { Учет единицы переноса }
        Begin                    { при округлении дроби }
          a[i]:=a[i]+p;
          If a[i]>=q then
            a[i]:=a[i]-q
          Else
            p:=0;
          Dec(i);
        End;
      Writeln('Цифры числа после округления :');
      For i:=1 to 10 do
        Write(a[i],' ');
      Writeln;
      R:=0;                      { Перевод дробного }
      For i:=10 downto 1 do      { числа из системы q }
        R:=(R+a[i])/q;           { в десятичную систему }
      Writeln('Результат  R = ',R);
    End;
End.
```

В результате работы программы будет напечатано:

```
   M = 3   N = 7   q = 10
Исходное значение  R = 4.2857142857E-01
Цифры дробной части до округления
   4 2 8 5 7 1 4 2 8 5 7
Цифры дробной части после округления
   4 2 8 5 7 1 4 2 8 6
Результат  R = 4.2857142860E-01
```

3.5. Определить, имеет ли заданное натуральное число M строго возрастающую последовательность цифр в системе счисления с основанием $2 \le q \le 10$.

Числовая последовательность $x_1, x_2, x_3 ..., x_n$ является возрастающей, если выполняются условия $x_1 \le x_2 \le x_3 \le ... \le x_n$. Эта же последовательность будет строго возрастающей при условии $x_1 < x_2 < x_3 < ... < x_n$.

По отношению к записи цифр целого числа проверке должно подвергаться отношение $a_n < a_{n-1} < ... < a_1 < a_0$.

В программе Task305 при просмотре массива A проверяется обратное отношение $a_i \ge a_{i-1}$, $i = 1 .. n$, и если оно имеет место, то дальнейший просмотр массива A прекращается.

```
Program Task305;
Const  Nmax = 32;
Type  Ar = array[0..Nmax] of byte;
Var  M,M1 : longint;
     i,n,q : byte;
     Cond : boolean;
     A : Ar;
Begin
  Read(M,q);
  M1:=M; n:=0;              { Формирование цифр числа }
  Repeat                    { M в системе счисления   }
    a[n]:=M1 mod q;  M1:=M1 div q; { с основанием  q }
    Inc(n);
  Until  M1 = 0;
  Writeln('M = ',M,'  q = ',q,'  n = ',n);
  Writeln('Цифры числа M в системе счисления q:');
  For i:=n-1 downto 0 do
    Write(a[i],' ');
  Writeln;
  Cond:=true;               { Проверка строгой упорядоченнос- }
  For i:=1 to n-1 do        { ти по возрастанию цифр числа M  }
    If a[i]>=a[i-1] then
      Begin
        Cond:=false; Break
      End;
  If Cond then
    Writeln('Последовательность цифр строго возрастающая')
  Else
    Writeln('Последовательность цифр не является строго ',
            'возрастающей');
End.
```

В результате работы программы будет напечатано:

```
   M = 1245789    q = 7    n = 8
Цифры числа M в системе счисления q:
   1 3 4 0 6 0 1 6
Последовательность цифр не является строго возрастающей
```

3.6. Заданы две простые дроби $\frac{m}{n}$ и $\frac{p}{q}$, где m,n,p,q - натуральные числа. Определить дробную часть их суммы в виде несократимой простой дроби $\frac{a}{b}$.

Сумму двух дробей можно определить по общеизвестной формуле

$$R=\frac{m}{n}+\frac{p}{q}=\frac{mq+pn}{nq},$$

однако более надежно предварительно найти наименьшее общее кратное их знаменателей :

$$k(n,q)=\frac{nq}{d(n,q)},$$

где d - наибольший общий делитель чисел n и q.

Тогда

$$R=\frac{\frac{mq}{d}+\frac{pn}{d}}{\frac{nq}{d}}=\frac{m\frac{q}{d}+p\frac{n}{d}}{k}$$

При использовании этой формулы увеличивается возможный диапазон представления чисел m,n,p,q.

Для сокращения дробной части $\frac{a}{b}$ числа R используется наибольший общий делитель $d(a,b)$.

```
Program Task306;
Var  m,n,p,q,a,b,d,k : word;
     R : real;
{ ----------------------------------- }
Function Evklid(m,n:word):word;
{ Нахождение наибольшего общего делителя }
Var  r : word;
Begin
  If m<n then
    Begin
      r:=m; m:=n; n:=r
    End;
  While n>0 do
    Begin
      r:=m mod n; m:=n; n:=r;
    End;
  Evklid:=m;
End { Evklid };
{ ----------------------------------- }
Begin
  Read(m,n,p,q);
  Writeln('m = ',m,'   n = ',n,'   p = ',p,'   q = ',q);
  d:=Evklid(n,q); k:=n*(q div d);
  a:=m*(q div d)+p*(n div d);
  b:=k;
  If a>=b then
    a:=a-b;
  d:=Evklid(a,b);
  a:=a div d; b:=b div d;
  Writeln('a = ',a,'   b = ',b);
```

```
End.
```

В результате работы программы будет напечатано:

```
m = 58    n = 60    p = 475    q = 500
a = 11    b = 12
```

3.7. Для заданного вещественного числа $0 < R < 1$ найти период его дроби в системе счисления с основанием $2 \le q \le 10$, если такой существует.

Вначале рассмотрим алгоритм и программу решения при $q = 2$.

Например, для числа $0{,}3 = 0{,}010011001100110011001100... = 0{,}0(1001)$ период равен 1001.

Вариант 1.

В варианте 1 сначала формируем k цифр дробной части числа, а затем производим поиск повторяющейся части в полученной последовательности цифр.

Предположим, что нам заданы числитель m и знаменатель n простой дроби. Тогда $R = \frac{m}{n}$. Если теперь формировать двоичные цифры числа R таким же образом, как это сделано в программе Task304, то можно получить неверный результат вследствие ошибки представления вещественного числа.

Вещественные числа в ПЭВМ заданы в форме двоичных мантиссы и порядка. В частности, в формате *real* мантисса занимает 5 байт (39 бит + 1 бит для знака числа), порядок - 1 байт (8 бит). Следовательно, если дробь *R* не является конечной, то мантисса автоматически округляется до 40 двоичных разрядов.

При формировании двоичных цифр по алгоритму, использованному в программе Task304, в каждом цикле мантисса умножается на 2. Это эквивалентно ее сдвигу влево на один разряд, после чего отбрасывается целая часть числа, а его дробная часть снова увеличивается в 2 раза и т.д. Тогда после 39 сдвигов мантисса будет нулевая, а следовательно, остаточная дробь в этом случае равна нулю. Равенство остаточной дроби $R = 0$ означает, что все остальные цифры двоичной дроби равны нулю. Тогда программа должна выдать сообщение, что исходная двоичная дробь конечная, что не соответствует действительности.

В программе Task307a для получения двоичных цифр простой дроби не производится предварительное деление числителя m на знаменатель n. Преобразование выполняется непосредственно по отношению к исходным значениям m и n. Это обеспечивает получение любого количества точных цифр простой дроби в произвольной системе счисления q. Сформированные дробные цифры числа m/n записываются в массив A.

Индикация потенциального периода дроби производится следующим образом.

Переменная *i*, изменяемая во внешнем цикле обработки массива *A*, определяет начальный разряд отбора цифр периода, переменная *l* - количество цифр периода. Эти цифры переписываются в массив *B*. Если все *l* цифр в массиве *B* нулевые, то проверка такого периода не производится. В противном случае *l* цифр массива *B* последовательно сравниваются с соответствующими отрезками массива *A*. При этом формируются две переменные: *Count,* определяющая количество совпадений проверяемого периода с отрезками массива *A*, и *p*, указывающая количество этих отрезков. Период дроби считается найденным, если выполнены два условия: *Count* = *p* и *Count* > 2. Если же отмечено лишь два совпадения достаточно длинного периода с отрезками массива *A* (*Count* = 2), то это ставит под сомнение достоверность результата (в этом случае желательно увеличить количество формируемых цифр $N_{\max}$).

Примечание. Число с типом *real* при выводе его в 10 с/с имеет мантиссу, содержащую 11 десятичных цифр. Методику генерации массива цифр дробной части числа по образцу примера 3.04 демонстрирует следующий фрагмент программы:

```
P:=m/n; q:=10;
For i:=1 to 50 do
  Begin
    P:=P*q;
    a[i]:=Trunc(P);
    P:=P-a[i];
  End;
```

Для $m = 1$ и $n = 7$ будет получен следующий массив цифр:

A = 14275714285710128024220466613769531250000000000000

Это свидетельствует о том, что 1/7 в 10 с/с – это конечная дробь, что в данном случае неверно. Результат, полученный по программе Task307a для тех же исходных данных, указывает, что данная дробь имеет период 142857.

```
Program Task307a;
Label  10;
Const  Nmax = 50;
Type  Ar = array[1..Nmax] of byte;
Var  i,j,k,l,p,q,Count : byte;
     m,n : word;
     Cond,CondZero,CondEqual : boolean;
     A,B : Ar;
Begin
  Read(m,n); q:=2;
  Writeln('m = ',m,'  n = ',n);
  For i:=1 to Nmax do          { Формирование цифр }
    Begin                      { дробного числа m/n }
      m:=m*q;                  { в системе счисления }
      If m>=n then             { с основанием q }
        Begin
          a[i]:=m div n; m:=m mod n
        End
      Else
        a[i]:=0;
    End;
  Writeln('Дробные цифры числа m/n :');
  For i:=1 to Nmax do
    Write(a[i]);
  Writeln;
  If m=0 then
    Begin
      Writeln('Дробь конечная'); Exit
    End;
  i:=1; Cond:=false;
  While i<Nmax do          { i - нач.разряд периода дроби }
    Begin
      l:=1;                { l - длина периода }
      While l<= (Nmax-i) div 2 do
        Begin
          k:=0;
          For j:=i to i+l-1 do       { перепись цифр периода }
            Begin                    { в массив B }
```

```
            Inc(k); b[k]:=a[j];
          End;
        CondZero:=false;          { проверка наличия в }
        For j:=1 to l do          { периоде ненулевых цифр }
          If b[j]>0 then
            CondZero:=true;
        If CondZero then
          Begin
            j:=i+1; p:=0; Count:=0;
            While j<=Nmax-l+1 do       { последовательное }
              Begin                    { сравнение цифр }
                CondEqual:=true; Inc(p); { периода с от-}
                For k:=1 to l do         { резками мас- }
                  If b[k]<>a[j+k-1] then { сива A }
                    CondEqual:=false;
                  If CondEqual then Inc(Count);
                  Inc(j,l);
              End;
            If (Count>2) and (Count=p) then
              Begin
                Cond:=true; Goto 10    { период найден }
              End;
          End;
        Inc(l);
      End;
    Inc(i);
  End;
10:
If Cond then
  Begin
    Write('Дробь имеет период ');
    For j:=1 to l do
      Write(b[j]);
    Writeln(', начиная с разряда ',i);
  End
Else
  Writeln('Дробь непериодическая');
End.
```

В результате работы программы будет напечатано:

```
m = 1   n = 8    R = 1.2500000000E-01
Дробные цифры числа R :
00100000000000000000000000000000
Дробь конечная

m = 3   n = 10    R = 3.0000000000E-01
Дробные цифры числа R :
01001100110011001100110011001100
Дробь имеет период 1001, начиная с разряда 2

m = 3   n = 100    R = 3.0000000000E-02
Дробные цифры числа R :
00000111101011100001010001111010111
Дробь непериодическая
```

Нетрудно заметить, что программа Task307a работоспособна при любом значении q, заданном в диапазоне 2 .. 10.

Вариант 2.

Анализируя остатки, получаемые при последовательном делении m/n, легко определить момент окончания периода: период заканчивается, как только получится остаток, который встречался ранее. При $m = 1$ можно реализовать эффективную программу, в которой не требуется в явном виде запоминать последовательный массив остатков.

В программе Task307b выводятся на печать все дробные числа 1/*i* для *i*, изменяющегося от 2 до $N_{\max}$. Массив *D* - это массив цифр дроби, массив *X* - индексы остатков. Индексы обозначают то место, откуда начинается период.

При каждом повторении цикла по *i* индексам присваивается нулевое значение. После этого заполняются те элементы массива *X*, индексы которых равны очередному остатку от деления. Цикл ***Repeat*** формирования цифр дроби заканчивается, когда $x_{ост} \neq 0$, что определяет появление ранее полученного остатка.

Например, для числа 1/7 = 0,(142857) будут по остаткам формироваться следующие элементы массива *X*: $x_1, x_3, x_2, x_6, x_4, x_5, x_1$.

```
Program Task307b;
Const  Nmax = 100;
Type   Ar = array[0..Nmax] of byte;
Var     i,j,k,q : byte;
        ost : word;                { остаток от деления }
        D,X : Ar;
Begin
  Read(q);
  For i:=2 to Nmax do
    Begin
      For j:=0 to i-1 do
        x[j]:=0;
      k:=0; Ost:=1;
      Repeat
        Inc(k); x[ost]:=k;
        ost:=q*ost; d[k]:=ost div i;
        ost:=ost mod i;
      Until x[ost]<>0;
      Write(i:6,'  0.');
      For j:=1 to x[ost]-1 do
        Write(d[j]);
      Write('(');
      For j:=x[ost] to k do
        Write(d[j]);
      Write(')');
      Writeln;
    End;
End.
```

Программа Task307b при $q = 10$ печатает результаты в следующем виде (приводятся выборочные результаты):

```
     2   0.5(0)
     3   0.(3)
    12   0.08(3)
```

```
17   0.(0588235294117647)
24   0.041(6)
```

Вариант 3.

При $m > 1$ и использовании идеи варианта 2 необходимо формировать не массив индексов, а массив значений остатков от деления, и при получении очередного остатка проверять, был ли он ранее записан в этот массив.

```
Program Task307c;
Const  Nmax = 100;
Type  Ard = array[1..Nmax] of byte;
     Aros = array[1..Nmax] of word;
Var    i,j,k,m,n,q : byte;
       ost : word;          { остаток от деления }
      Cond : boolean;
         D : Ard;          { массив цифр числа }
        Os : Aros;         { массив остатков от деления }
Begin
  В в о д  q, m, n
  i:=0; Ost:=m; Cond:=true;
  While Cond do
    Begin
      Inc(i); Os[i]:=Ost;
      ost:=q*ost; d[i]:=ost div n;
      ost:=ost mod n;
      k:=0;
      For j:=1 to i do
        If ost=Os[j] then
          Begin
            k:=j; Cond:=false;
          End;
      If (i=Nmax) or (ost=0) then
        Cond:=false;
    End;
  Write('0.');
  If ost=0 then
    Begin
      For j:=1 to i do
        Write(d[j]);
      Write('(0)');
    End
  Else
    If k>0 then
      Begin
        For j:=1 to k-1 do
          Write(d[j]);
        Write('(');
        For j:=k to i do
          Write(d[j]);
        Write(')');
      End
    Else
      For j:=1 to i do
        Write(d[j]);
End.
```

Форма печати результата в программе Task307c такая же, как и в программе Task307b.

3.8. Натуральное число M задано массивом своих двоичных цифр $a_n, a_{n-1}, ..., a_1, a_0$, причем старшая цифра $a_n = 0$. Напечатать массив двоичных цифр чисел $M+1$ и $M-1$.

Переменная *p* в программе Task308 - это единица переноса, используемая при формировании числа $M + 1$. Работа программы организована по правилам сложения и вычитания двоичных чисел.

```
Program Task308;
Const  Nmax = 50;
Type  Ar = array[0..Nmax] of byte;
Var  i,j,n,p : byte;
     A : Ar;
Begin
  В в о д   n, A
{ Формирование числа  M + 1 }
  p:=1; i:=0;
  While p=1 do
    Begin
      a[i]:=a[i]+p;
      If a[i]>1 then
        a[i]:=0
      Else
        p:=0;
      Inc(i);
    End;
  П е ч а т ь   A
{ Формирование числа  M - 1 }
  i:=0;                  { Поиск первого }
  While a[i]=0 do        { ненулевого разряда }
    Inc(i);
  a[i]:=0;
  If i>0 then
    For j:=0 to i-1 do
      a[j]:=1;
  П е ч а т ь   A
End.
```

В результате работы программы будет напечатано:

```
  0 1 0 0 1 0 1 0 1 1 1 1
  0 1 0 0 1 0 1 1 0 0 0 0
  0 1 0 0 1 0 1 0 1 1 1 1
```

3.9. В массивах $X = (x_0, x_1, ..., x_n)$ и $Y = (y_0, y_1, ..., y_m)$ заданы цифры двух чисел в системе счисления $2 \le q \le 10$ (x_0, y_0 - младшие цифры чисел X и Y). Сформировать цифры числа $Z = X + Y$ в той же системе счисления q и отпечатать десятичные значения X, Y, Z.

Комментарии к программе Task309.

1) При сложении чисел *X* и *Y*, имеющих *n* разрядов, может иметь место формирование старшего (*n*+1)-го разряда. Поэтому в объявлении массивов указан тип индекса 0 .. $N_{\max}$ +1.

При этом предполагается, что для исходных массивов *nx, ny* ≤ $N_{\max}$, для суммы *nz* ≤ $N_{\max}$ +1.

2) Цифры исходных массивов записываются в файле в общепринятом виде, т.е. начиная со старшего разряда. При вводе, поскольку заранее неизвестны значения *nx* и *ny*, заполнение массивов *X* и *Y* начинается с нулевого разряда. Поэтому в процедуре *ReadArray* после ввода массива цифр из файла производится их перестановка в обратном порядке.

3) Поскольку длины чисел *X* и *Y* в общем случае могут быть различными, то незаполненные старшие разряды обнуляются.

4) Сложение чисел *X* и *Y* производится по обычной схеме "в столбик" с формированием единицы переноса *p*.

5) Преобразование цифрового массива в десятичное значение выполняется функцией *TransNumber* по описанной ранее схеме Горнера.

```
Program Task309;
Const  Nmax = 20;
Type  Ar = array[0..Nmax+1] of byte;
Var  i,j,nx,ny,nz,m,p,q : byte;
     Xl0,Yl0,Zl0 : longint;  { значения X,Y,Z в 10 с/с }
     X,Y,Z : Ar;
     Fx,Fy : text;
{ ---------------------------------- }
Procedure ReadArray(Var F:text; Var A:Ar; Var n:byte);
{ Ввод массива цифр числа в системе счисления q }
Var  i,j,k : byte;
Begin
  Reset(F);
  n:=0;
  While not SeekEof(F) do
    Begin
      Read(F,a[n]); Inc(n);
    End;
  Dec(n);
  Close(F);
  i:=0; j:=n;
  While i<j do
    Begin
      k:=a[i]; a[i]:=a[j]; a[j]:=k;
      Inc(i); Dec(j);
    End;
End { ReadArray };
{ ---------------------------------- }
Function TransNumber(Var A:Ar; n:byte):longint;
{ Формирование десятичного значения числа }
Var  i : byte;
     P : longint;
Begin
  P:=0;
  For i:=n downto 0 do
    P:=P*q+a[i];
  TransNumber:=P;
End { TransNumber };
{ ---------------------------------- }
Begin
  Read(q); Writeln('q = ',q);
  Assign(Fx,'Fx.dat');  Assign(Fy,'Fy.dat');
```

```
  ReadArray(Fx,X,nx);   ReadArray(Fy,Y,ny);
  Печать массивов X и Y
  For i:=nx+1 to Nmax do
    x[i]:=0;
  For i:=ny+1 to Nmax do
    y[i]:=0;
  For i:=0 to Nmax+1 do
    z[i]:=0;
  If nx>ny then
    m:=nx
  Else
    m:=ny;
  p:=0;
  For i:=0 to m+1 do
    Begin
      z[i]:=x[i]+y[i]+p;
      If z[i]>=q then
        Begin
          p:=1; z[i]:=z[i]-q;
        End
      Else
        p:=0;
    End;
  nz:=m+1;
  Печать массива Z
  Xl0:=TransNumber(X,nx); Yl0:=TransNumber(Y,ny);
  Zl0:=TransNumber(Z,nz);
  Writeln('Xl0=',Xl0,'  Yl0=',Yl0,'  Zl0=',Zl0);
End.
```

Пример решения для десятичной системы счисления:

```
   q = 10
   Массив X   n = 8
   3 5 8 6 4 1 4 6 5
   Массив Y   n = 7
     1 2 9 5 3 9 8 7
   Массив Z   n = 9
 0 3 7 1 5 9 5 4 5 2
   Xl0 = 358641465  Yl0 = 12953987  Zl0 = 371595452
```

Пример решения для восьмеричной системы счисления:

```
   q = 8
   Массив X   n = 9
   1 3 5 4 6 4 1 4 6 5
   Массив Y   n = 8
     1 1 2 6 5 3 5 4 7
   Массив Z   n = 10
 0 1 4 6 7 5 1 5 2 3 4
   Xl0 = 196297525  Yl0 = 19617639  Zl0 = 215915164
```

3.10. В массивах $X=(x_0,x_1,...,x_n)$ и $Y=(y_0,y_1,...,y_m)$ заданы цифры двух чисел в системе счисления $2\le q\le 10$ (x_0,y_0 - младшие цифры чисел X и Y), причем $X\ge Y$.

Сформировать цифры числа $Z = X - Y$ в той же системе счисления q. Преобразование в десятичную систему и обратно не использовать.

Просмотр разрядов вычитаемых чисел *X* и *Y* производится справа налево, т.е. от младшего разряда к старшему. Если $x_i < y_i$, то с помощью процедуры *Trans* определяется ближайший ненулевой старший разряд *it* числа *X*. Содержимое разряда *it* уменьшается на 1. Если между *it* и *i* имеются нулевые разряды, им присваивается значение *q*-1. Разряд *i* увеличивается на значение *q*.

Для определения номера старшего разряда *nz* полученной разницы *Z* в массиве *Z* определяется ближайший слева ненулевой разряд. Если *X* = *Y* и, следовательно, *Z* = 0, то *nz*= 0. Тогда значением *Z* является $z_0 = 0$.

```
Program Task310;
Const  Nmax = 20;
Type  Ar = array[0..Nmax] of byte;
Var  i,j,nx,ny,nz,q : byte;
     X,Y,Z : Ar;
{ -------------------------------- }
Procedure Trans(k:byte);
Var  it,jt : byte;
Begin
  For it:=k+1 to nx do
    If x[it]<>0 then
      Begin
        x[it]:=x[it]-1;
        For jt:=it-1 downto k+1 do
          x[jt]:=q-1;
        x[k]:=x[k]+q; Exit;
      End;
End { Trans };
{ -------------------------------- }
Begin
  Ввод  q, nx, X, ny, Y
  For i:=ny+1 to Nmax do y[i]:=0;
  For i:=0 to Nmax do z[i]:=0;
  For i:=0 to nx do
    Begin
      If x[i]<y[i] then
        Trans(i);
      z[i]:=x[i]-y[i];
    End;
  nz:=0;
  For i:=Nmax downto 1 do
    If z[i]<>0 then
      Begin
        nz:=i; Break
      End;
  Печать  nz, Z
End.
```

Пример решения для десятичной системы счисления:

```
q = 10
Массив X   n = 8
3 5 8 6 4 0 0 0 5
Массив Y   n = 7
```

```
  1 2 9 5 3 9 8 7
Массив Z   n = 9
3 4 5 6 8 6 0 1 8
```

Пример решения для восьмеричной системы счисления:

```
q = 8
Массив X   n = 9
3 5 7 6 4 0 0 0 5
Массив Y   n = 8
  1 2 7 5 3 7 6 7
Массив Z   n = 9
3 4 4 6 6 4 0 1 6
```

3.11. Найти сумму и количество всех простых чисел, не превышающих значения $N \le 10000$. Операции умножения и деления не использовать.

Для формирования списка простых чисел наиболее эффективным является алгоритм, называемый "решето Эратосфена". Преимуществом этого алгоритма является то, что в нем для нахождения простых чисел не нужно использовать операции умножения и деления.

Алгоритм Эратосфена рассмотрен в разделе "Формирование списка простых чисел".

По условию задачи не требуется в явном виде формировать список простых чисел. Поэтому в программе Task311 после получения очередного простого числа изменяются лишь счетчик *Count* и сумма *Sum* простых чисел. В качестве "решета" в программе используется массив *Sieve*, в который предварительно записываются натуральные числа от 2 до 10000.

```
Program Task311;
Const  Nmax = 10000;
Type  SieveAr = array[2..Nmax] of word;
Var  i,j,
     n,                  { анализируемый диапазон  1..n }
     Count,              { кол-во простых чисел }
     NextPrime,          { очередное простое число }
     Multiple : word;    { множитель, кратный NextPrime }
     Sum : longint;      { сумма простых чисел }
     Sieve : SieveAr;    { "решето" натуральных чисел }
     Cond : boolean;
Begin
  Read(n); Writeln('n = ',n);
  For i:=2 to n do       { размещение чисел в "решете" }
    Sieve[i]:=i;
  Sum:=1; Count:=1; Cond:=true;
  While Cond do
    Begin
      NextPrime:=0;
      For i:=2 to N do              { нахождение минималь- }
        If Sieve[i]<>0 then         { ного числа в массиве }
          Begin                     { Sieve                }
            NextPrime:=i; Break
          End;
      If NextPrime=0 then           { если NextPrime=0, то }
        Cond:=false                 { "решето" - пустое    }
      Else
```

```
        Begin
          Inc(Count); Sum:=Sum+NextPrime;
          Multiple:=NextPrime;
          While Multiple<=n do     { удаление из решета чи-}
            Begin                  { сел, кратных значению }
              Sieve[Multiple]:=0;            { NextPrime }
              Inc(Multiple,NextPrime);
            End;
         End;
    End;
  Writeln('Count = ',Count,'   Sum = ',Sum);
End.
```

В результате работы программы будет напечатано:

```
n = 1230   Count = 202   Sum = 112817
n = 10000  Count = 1230  Sum = 5736397
```

3.12. Для заданного натурального числа $N < 2^{30}$ найти количество и сумму всех его простых делителей (разложение натурального числа, большего 1, на простые множители называется факторизацией).

При решении задачи требуется формировать список простых чисел в диапазоне от 2 до $2^{29} = 536\ 870\ 912$. Массив с таким количеством элементов создать невозможно, поэтому алгоритм Эратосфена здесь неприменим.

В программе Task312 не проверяется явно, является ли очередной делитель простым числом. Индикация простых делителей начинается со значения $i = 2$. Если число *M*, равное первоначально *N*, делится на *i*, то в цикле ***While*** выполняется уменьшение числа *M* в *i* раз до тех пор, пока не прекратится делимость на *i*. Тогда в дальнейшем в списке делителей будут отсутствовать числа, кратные *i*, а очередной делитель наверняка будет простым числом.

```
Program Task312;
Var  k,M,N,Sum : longint;
     Count : word;
Begin
  Read(N); Writeln('N = ',N);
  Count:=0; Sum:=0;
  k:=2; M:=N;
  While M > 1 do
    If M mod k = 0 then
      Begin
        Inc(Count); Inc(Sum,i);
        M:=M div k;
      End
    Else
      Inc(k);
  Writeln('Count = ',Count,'   Sum = ',Sum);
End.
```

В результате работы программы будет напечатано:

```
N = 123456788
Count = 6    Sum = 391
```

3.13. Заданное натуральное число $N < 2^{30}$ разложить на простые множители.

Программа Task313, реализующая поставленную задачу, является модификацией программы Task312.

```
Program Task313;
Const  Nmax = 500;
Type  Ar1 = array[1..Nmax] of longint;
      Ar2 = array[1..Nmax] of byte;
Var  M,N,
     k,                 { множитель }
     kp : longint;      { предыдущее значение множителя }
     Count : word;      { кол-во простых множителей }
     Multiple : Ar1;    { список простых множителей }
     NumMult  : Ar2;    { кол-во повторений множителей }
Begin
  Read(N); Writeln('N = ',N);
  Count:=0;
  k:=2; kp:=1; M:=N;
  While M > 1 do
    If (M mod k)=0 then
      Begin
        M:=M div k;
        If k <> kp then
          Begin
            Inc(Count); kp:=k;
            Multiple[Count]:=k; NumMult[Count]:=1;
          End
        Else
          Inc(NumMult[Count]);
      End
    Else
      Inc(k);
  Writeln('Count = ',Count);
  Writeln(' Множитель   Кол-во повторений');
  For i:=1 to Count do
    Writeln(Multiple[i]:6,'          ',NumMult[i]:3);
End.
```

В результате работы программы будет напечатано:

```
N = 123456788
Count = 6
Множитель  Кол-во повторений
     2            2
     7            1
    13            1
    17            1
    71            1
   281            1
```

3.14. Задан массив цифр $X=(x_1,x_2,...,x_n)$ натурального числа в системе счисления с основанием $2 \le q \le 10$. Число может содержать незначащие нули. Определить, является ли это число степенью основания q. Операции умножения и деления не использовать.

Число является степенью своего основания q, если среди его цифр содержится лишь одна единица, а остальные цифры - нули.

```
Program Task314;
Const  Nmax = 50;
Type  Ar = array[0..Nmax] of byte;
Var  i,k1,k2,n : byte;
     A : Ar;
Begin
  В в о д   n, X
  k1:=0; k2:=0;
  For i:=0 to n do
    Begin
      If x[i]=1 then     { кол-во единиц в числе }
        Inc(k1);
      If x[i]<>0 then    { кол-во ненулевых цифр в числе }
        Inc(k2);
    End;
  If (k1=1) and (k1=k2) then
    Writeln('Число является степенью основания q')
  Else
    Writeln('Число не является степенью основания q');
End.
```

3.15. Определить, является ли натуральное число $N < 2^{30}$ степенью основания 2. Арифметические операции не использовать.

В отличие от предыдущей задачи, здесь задано число, а не массив его цифр.

Целое число в машинном представлении записано в двоичной системе счисления. Это число является степенью основания 2, если в его состав входит лишь одна двоичная единица. Значению 2^{30} соответствует тип *longint*. Запрет использования арифметических операций означает, что мы не можем с помощью операций ***div*** и ***mod*** представить число *N* в виде массива его двоичных цифр. В этом случае могут быть предложены два способа решения.

1) Выполнить логическое умножение числа *N* типа *longint* на константу $00000001. После этого операцией ***shr*** сдвинуть число *N* вправо на один разряд. Логическое умножение и сдвиг повторить 31 раз. Если результат логического умножения равен 1, то добавить единицу к счетчику.

Если после окончания цикла содержимое счетчика равно 1, то это означает, что число *N* является степенью основания 2.

2) Заменить логическое умножение проверкой четности числа *N* с помощью функции *odd* (работа этой функции на машинном уровне также сводится к логическому умножению на константу $00000001).

```
Program Task315;
Const  Log = $00000001;
Var  N,M : longint;
     i,Count : byte;
Begin
  В в о д  N
  N:=abs(N); Count:=0;
  For i:=1 to 31 do
    Begin
      M:=N and Log;
```

```
      If M>0 then
        Inc(Count);
       N:=N shr 1;
    End;
  If Count=1 then
    Writeln('Число является степенью основания 2')
  Else
    Writeln('Число не является степенью основания 2');
End.
```

3.16. Определить количество нулей в двоичном представлении заданного натурального числа $N < 2^{30}$. Арифметические операции не использовать.

Задача в основном аналогична предыдущей. Однако при ее реализации необходимо учесть, что в искомое количество не должны быть включены незначащие нули, расположенные перед первым единичным разрядом. В связи с этим на начальном этапе производится сдвиг числа *N* влево до тех пор, пока не будет обнаружена первая единица. Такой анализ выполняется путем логического умножения на константу $40000000 (первая двоичная цифра числа типа *longint* находится во втором разряде, первый разряд - это знак числа). Подсчет количества незначащих нулей начинается после обнаружения первой единицы числа.

Если $N = 0$, то в составе такого числа имеется лишь один значащий нуль.

```
Program Task316;
Const  Log = $40000000;
Var  N,M : longint;
     i,k,Count : byte;
     Cond : boolean;
Begin
  В в о д  N
  N:=abs(N);
  If N=0 then
    Count:=1
  Else
    Begin
      Cond:=true; k:=0;
      While Cond do
        Begin
          Inc(k);
          M:=N and Log;
          If M>0 then
            Cond:=false
          Else
            N:=N shl 1;
        End;
      Count:=0;
      For i:=2 to 31-k do
        Begin
          M:=N and Log;
          If M=0 then
            Inc(Count);
          N:=N shl 1;
        End;
    End;
  Печать Count
End.
```

3.17. Задан массив цифр $X = (x_1, x_2, ..., x_n)$ натурального числа в системе счисления с основанием $2 \le q \le 10$, где x_1 младшая цифра числа. Число может содержать незначащие нули. Определить, является ли это число кратным своему основанию q. Операции умножения и деления не использовать.

Число кратно своему основанию q, если его младший разряд равен нулю.

```
Program Task315;
Const  Nmax = 50;
Type  Ar = array[0..Nmax] of byte;
Var i,k1,k2,n : byte;
    X : Ar;
Begin
  В в о д   n, X
  If x[1]=0 then
    Writeln('Число кратно основанию q')
  Else
    Writeln('Число не кратно основанию q');
End.
```

Задания для самостоятельной работы

3.18. Отпечатать первые n чисел, делителями которых являются только числа 2, 3 и 5. Пример таких чисел: 20, 25, 27, 30, 64.

3.19. Определить, сколько единиц содержится в двоичном представлении целого положительного числа M.

3.20. Заданы n цифр целой части и m цифр дробной части двоичного числа. Удалить из состава числа незначащие нули, если они имеются.

3.21. Задано нечетное натуральное число M. Определить, является ли двоичная запись этого числа симметричной.

3.22. Определить, делится ли заданное натуральное число M на каждую из своих десятичных цифр. Нулевые цифры не учитывать.

3.23. Определить, имеет ли заданное натуральное число M одинаковые цифры в системе счисления с основанием $2 \le q \le 10$.

3.24. Задано произвольное целое число M. Требуется сгруппировать его восьмеричные цифры в порядке убывания и отпечатать полученное десятичное значение.

3.25. Целое $2n$-значное число называют счастливым, если сумма его первых n цифр совпадает с суммой остальных n цифр. Определить, является ли счастливым заданное натуральное число M.

3.26. Определить количество таких натуральных чисел, не превосходящих заданного значения N, двоичная запись которых представляет последовательность нулей и единиц, симметричную относительно ее середины и начинающуюся с единицы.

3.27. Заданы n цифр числа в системе счисления с основанием p ($2 \le p \le 10$). Сформировать цифры этого числа в системе счисления с основанием q ($2 \le q \le 10$, $q \ne p$).

3.28. Указать то число, не превышающее заданного натурального значения N, в двоичном представлении которого больше всего единиц.

3.29. В целочисленном массиве цифрами 0 и 1 записано двоичное число, целая и дробная части которого разделены цифрой 2. Отпечатать восьмеричное представление этого числа, разделив точкой его целую и дробную части. Перевод в 10 с/с не использовать.

Указание. Использовать тот факт, что каждая триада двоичного числа – это одна восьмеричная цифра.

3.30. Отпечатать все натуральные числа, не превосходящие заданного значения N, в двоичном представлении которых номера ненулевых разрядов образуют арифметическую прогрессию.

3.31. Отпечатать все пары соседних простых чисел, не превосходящих заданного значения N, троичные представления которых получаются друг из друга записью цифр в обратном порядке (первая такая пара - это 5 и 7, в троичной записи - 12 и 21).

3.32. Напечатать в порядке возрастания все простые несократимые дроби, знаменатели которых не превышают заданное значение $2 \le N \le 10$. Печать производить в виде тройки чисел, определяющих числитель дроби, ее знаменатель и десятичное значение.

3.33. Правильными делителями натурального числа N являются все делители этого числа, за исключением его самого. Число называется совершенным, если оно равно сумме своих правильных делителей (например, $6 = 1 + 2 + 3$); его называют неполным, если оно меньше суммы своих правильных делителей ($12 < 1 + 2 + 3 + 4 + 6$), и избыточным, если оно больше этой суммы ($9 > 1 + 3$). Определить количества совершенных, неполных и избыточных чисел, не превышающих заданного значения N.

3.34. Среди натуральных чисел, не превышающих заданного значения $N < 2^{30}$, найти такое, для которого сумма его делителей максимальна. В состав делителей проверяемого числа M не включать значения 1 и M.

3.35. Для заданного натуральное числа N отпечатать список чисел Мерсена, не превышающих значения N (число Мерсена - простое число, представимое в виде 2^p -1, где p - также простое число).

3.36. Дано натуральное число $N \le 10000$. Среди чисел 1 .. N найти все автоморфные числа, т.е. такие, запись которых совпадает с последними цифрами записи их квадрата ($6^2 = 36$, $25^2 = 625$, $76^2 = 5776$).

3.37. Найти все меньшие заданного значения $N \le 40000$ натуральные числа-палиндромы, которые при возведении в квадрат также дают палиндром.

3.38. Найти все меньшие заданного значения $N < 1000000$ натуральные числа, которые являются палиндромами как в десятичной, так и в двоичной системах счисления.

3.39. Задано натуральное число N и массив его цифр $a_1, a_2, ..., a_m$ в некоторой системе счисления с основанием $2 \le q \le 10$. Определить значение q.

3.40. Даны натуральные числа m и n. Получить все их натуральные общие кратные, не превышающие значения $m \cdot n$.

3.41. Числами-близнецами называют такие простые числа, разница значений которых равна 2. Найти и отпечатать все пары чисел-близнецов, не превышающих 1000 (например, 5 и 7, 17 и 19, 461 и 463).

3.42. Натуральное число из n цифр является числом Армстронга, если сумма его n цифр, возведенных в n-ю степень, равна самому числу (например, $153 = 1^3 + 5^3 + 3^3$). Отпечатать все числа Армстронга, состоящие из двух, трех и четырех цифр.

3.43. Для натурального числа N задан массив, содержащий k его двоичных цифр, где $k \le 100$. Сформировать массив цифр числа $M = 3N$.

Указание. Вычисление вести по схеме $M = 2N + N$, используя операции сложения и сдвига двоичных чисел.

3.44. Определить, сколько чисел в диапазоне от M до N ($M \le N$) состоят только из нечетных цифр. Незначащие нули в числе не учитывать.

3.45. Для целого числа $N < 2^{31}$ определить его цифровой корень в системе счисления с основанием $3 \le q \le 10$.

Примечание. Цифровым корнем называют такую сумму цифр целого числа, которая после некоторого количества их сложений изображается одной цифрой. Пример для $q = 10$:

$853977 \to 8+5+3+9+7+7 = 39 \to 3+9 = 12 \to 1+2 = 3$

3.46. Заданы три натуральных числа $A < B < N$. Определить количество натуральных чисел, не превосходящих значения N, которые можно представить в виде суммы произвольного количества слагаемых, каждое из которых равно A или B.

3.47. Определить максимальную длину серии несчастливых билетов. Счастливым билетом считать тот билет, у которого сумма левых трех цифр равна сумме правых трех цифр. Несчастливые билеты образуют серию, если они идут один за другим.

3.48. Определить в целочисленном массиве X максимальное положительное число, двоичное представление которого содержит три единицы в старших разрядах и две единицы - в младших. Арифметические операции при анализе разрядов числа не использовать. Это означает, что запрещается преобразование числа в массив двоичных цифр.

3.49. Определить в целочисленном массиве X количество положительных чисел, двоичные представления которых состоят из чередования цифр 1 и 0 (например, 101010 или 10101). Арифметические операции при анализе разрядов числа не использовать.

3.50. Используя логические операции и операции сдвига, определить, является ли симметричной двоичная запись натурального числа N (например, 11011011, 10101 и т.п.).

3.51. Используя логические операции и операции сдвига, определить, имеются ли в двоичной записи натурального числа N две подряд стоящие единицы, окруженные слева и справа по крайней мере одним нулем (например, 101100111011).

4. Итерационные циклы

4.0. В общем случае итерационным называется такой вычислительный процесс, в котором нельзя заранее определить количество повторных вычислений (количество циклов). Это количество зависит от значений исходных данных и в большинстве случаев от требуемой точности вычислений.

Критерием окончания вычислений является достижение некоторого заранее заданного условия.

4.1. Определить порядковый номер и значение наибольшего члена ряда Фибоначчи, для которого сумма предыдущих элементов этого ряда не превышает заданной величины S.

Ряд Фибоначчи формируется по правилу

$$f_{n+2} = f_{n+1} + f_n; \quad f_0 = 0; \quad f_1 = 1; \quad n = 0, 1, 2, \ldots$$

В программе Task401 цикл ***While***, реализующий поиск требуемого значения, прекращает работу, когда будет достигнуто $\sum_{i=1}^{n} f_i > S$. Поэтому результатом решения является значение f_1.

```
Program Task401;
Var  f0,f1,f2,     { члены ряда Фибоначчи }
     n : word;     { количество выполнений цикла }
```

```
     S,Sum : longint;
Begin
  Read(S); Writeln('S = ',S);
  f0:=0; f1:=1; f2:=1;
  n:=1; Sum:=2;
  While Sum<=S do
    Begin
      f0:=f1; f1:=f2;
      f2:=f0+f1;
      Inc(n); Sum:=Sum+f2;
    End;
  Writeln('f1 = ',f1,'   n = ',n);
End.
```

При $b = 700$ получим $f1 = 610$, $n = 15$.

4.2. Золотое сечение, используемое в задачах оптимизации, определяется как предел отношения

$$V = \frac{f_n}{f_{n-1}},$$

где f_{n-1}, f_n - два последовательных члена ряда Фибоначчи. Требуется вычислить значение V с погрешностью, не превышающей заданного значения ε. Сравнить полученный результат с точным значением

$$V = \frac{1+\sqrt{5}}{2}$$

```
Program Task402;
Const  eps = 0.001;
Var  f0,f1,f2,     { члены ряда Фибоначчи }
     n : word;     { номер члена ряда Фибоначчи }
     V,            { текущее значение параметра V }
     Vold,         { предыдущее значение параметра V }
     V1 : real;    { точное значение золотого сечения }
Begin
  f0:=0; f1:=1; f2:=1;
  n:=1;  V:=0;
  Repeat
    Vold:=V;
    f0:=f1; f1:=f2; f2:=f0+f1;
    Inc(n); V:=f2/f1;
  Until abs(V-Vold)<=eps;
  V1:=0.5*(1+sqrt(5));
  Writeln('V = ',V:10:8,'   V1 = ',V1:10:8,'   n = ',n);
End.
```

Полученные результаты:

eps	n	V	V1
0.001	10	1.61818182	1.61803399
0.0001	12	1.61805556	1.61803399
0.00001	15	1.61803279	1.61803399
0.000001	17	1.61803381	1.61803399
0.0000001	19	1.61803396	1.61803399
0.00000001	22	1.61803399	1.61803399

4.3. Определить длину кривой $y = f(x)$ на интервале $[a,b]$ с погрешностью, не превышающей заданного значения ε.

Решение задачи выполнено в двух вариантах: в виде программы для конкретной функции $f(x)$ и в виде подпрограммы для произвольной функции $f(x)$.

Вариант 1.

Решение выполняется по отношению к функции $y = \sin^2\sqrt{x}$. Длина кривой приближенно определяется как длина ломаной, количество отрезков n которой равно количеству делений заданного интервала $[a,b]$. Начальное значение n принято равным 10. При каждом повторении цикла ***Repeat*** это количество увеличивается вдвое. Цикл работает до тех пор, пока разность между значением *LOld* длины кривой в предыдущем цикле и значением *L* этой длины в текущем цикле превышает допустимую погрешность ε (разность между "старым" и "новым" значениями длины кривой).

```
Program Task403a;
Const  eps = 0.001;
Var  i,
     n : word;          { кол-во отрезков разделения интервала }
     a,b,               { границы интервала }
     x1,y1,x2,y2,       { конечные точки отрезка }
     h,                 { шаг по оси абсцисс }
     d,                 { длина отрезка }
     L,LOld : real;     { длина кривой }
Begin
  Read(a,b);
  n:=5; L:=0;
  Repeat
    n:=2*n; LOld:=L;
    h:=(a+b)/n; L:=0;
    x1:=a; y1:=sqr(sin(sqrt(x1)));
    For i:=1 to n do
      Begin
        x2:=x1+h; y2:=sqr(sin(sqrt(x2)));
        d:=sqrt(sqr(x2-x1)+sqr(y2-y1));
        L:=L+d; x1:=x2; y1:=y2;
      End;
  Until abs(L-LOld)<=eps;
  Writeln('n=',n,'  L=',L:12);
End.
```

Для интервала [1,4] получены следующие результаты:

```
          eps = 0.001      n =  40      L = 5.09598
          eps = 0.0001     n =  80      L = 5.09606
          eps = 0.00001    n = 320      L = 5.09609
```

Вариант 2.

Вычисление длины кривой оформлено в виде функции *LengthCurve*, одним из аргументов которой является имя функции *F*, для которой определяется длина кривой. Формальный аргумент *F* является функциональным параметром типа *Fun*, описанным в разделе ***Type.*** При обращении к функции *LengthCurve* аргументу *F* соответствуют имена конкретных функций *F1* и *F2*, являющихся функциональными константами. Поскольку имена *F1* и *F2* восприни-

маются как константы, аргумент *F* в списке формальных параметров является параметром-значением (перед именем *F* не ставится слово ***Var***).

С точки зрения машинного представления параметр *F* - это указатель, определяющий адрес точки входа соответствующей функции. Для указателя всегда выделяется 4 байта памяти, в которых записывается полный адрес поля памяти (адрес сегмента и смещение).

Для процедур и функций обычно генерируется краткий адрес точки входа размером 2 байта, в которых записывается только смещение. Чтобы было обеспечено соответствие между функциональным параметром *F* и функциональными константами *F1* и *F2*, перед описанием функций *F1* и *F2* установлена директива F+, предписывающая компилятору генерировать полные адреса процедур и функций.

Функция F1 вычисляет значение $y = \sin^2\sqrt{x}$, функция F2 - $y = e^{\sqrt{x+\sin(x)}}$.

```
Program Task403b;
Type  Fun = function(x:real):real;
Var  L1,L2 : real;       { длины кривых }
{ ------------------------------------ }
{$F+}
Function F1(x:real):real;
Begin
  F1:=sqr(sin(sqrt(x)));
End { F1 };
{ ------------------------------------ }
Function F2(x:real):real;
Begin
  F2:=exp(sqrt(x+sin(x)));
End { F2 };
{$F-}
{ ------------------------------------ }
Function LengthCurve(F:Fun; a,b,eps:real):real;
{ F - имя функции; a,b - границы интервала; }
{ eps - допустимая погрешность }
Var  i,
     n : word;          { кол-во отрезков разделения интервала }
     x1,y1,x2,y2,       { конечные точки отрезка }
     h,                 { шаг по оси абсцисс }
     d,                 { длина отрезка }
     L,LOld : real;     { длина кривой }
Begin
  n:=5; L:=0;
  Repeat
    n:=2*n; LOld:=L;
    h:=(a+b)/n; L:=0;
    x1:=a; y1:=F(x1);
    For i:=1 to n do
      Begin
        x2:=x1+h; y2:=F(x2);
        d:=sqrt(sqr(x2-x1)+sqr(y2-y1));
        L:=L+d; x1:=x2; y1:=y2;
      End;
  Until abs(L-LOld)<=eps;
  LengthCurve:=L;
End { LengthCurve };
{ ------------------------------------ }
Begin
  L1:=LengthCurve(F1,1,4,0.001);
```

```
  L2:=LengthCurve(F2,1,4,0.001);
  Writeln('L1 = ',L1:12,'   L2 = ',L2:12);
End.
```

В результате работы программы Task403b получены следующие значения:

```
L1 = 5.09598   L2 = 9.41317
```

Задания для самостоятельной работы

4.4. Заданный вещественный массив $X(n)$ осреднить следующим образом: максимальный и минимальный элементы заменить их полусуммой, то же сделать по отношению к максимальному и минимальному элементам преобразованного массива X и так далее до тех пор, пока разница между максимальным и минимальным элементами не станет меньше заданного малого значения ε. Отпечатать количество произведенных усреднений и среднее арифметическое значение элементов массива X.

4.5. Найти частное и остаток двух целых чисел, пользуясь только операциями сложения и вычитания.

4.6. Найти произведение двух целых двоичных чисел, используя лишь операции сложения и сдвига.

4.7. Рассмотрим последовательность $d_1, d_2, ..., d_n$ периметров правильных многоугольников с удваивающимся числом сторон, вписанных в окружность с радиусом R. Определить значение n, при котором периметры двух последовательно образуемых многоугольников отличаются не более чем на ε, где ε - малое число (например, 0.000001). Начальное значение n принять равным 6.

Указание. Сторона a правильного вписанного n-угольника вычисляется по формуле $a = 2R\sin\frac{\pi}{n}$.

4.8. Заданы вещественные массивы X, Y, Z и M, определяющие координаты и массы материальных точек в трехмерном пространстве. Каждая из точек с максимальной массой исчезает, теряя десятую часть своей массы и раздавая оставшуюся всем остальным точкам обратно пропорционально их массам. Определить суммарную массу точек в тот момент, когда максимальная относительная разница между средним арифметическим масс и массами точек не превышает заданного значения ε (например, ε = 0.1).

4.9. Определить, содержится ли в заданной целочисленной последовательности хотя бы одно число Фибоначчи.

4.10. Так называемый эллиптический интеграл $I(a,b)$ равен пределу любой из двух сходящихся последовательностей

$$S_0, S_1, S_2, \ldots \text{ или } t_0, t_1, t_2, \ldots,$$

которые определяются следующими рекуррентными отношениями:

$$S_i = \frac{S_{i-1} + t_{i-1}}{2};\ t_i = \sqrt{S_{i-1}\, t_{i-1}}$$

и начальными значениями $S_0 = a,\ t_0 = b$.

Для заданных значений a и b вычислить $I(a,b)$ с погрешностью, не превышающей ε.

4.11. Определить значение S, вычислив первые n членов последовательности

$$S = \frac{2}{3}\sin(3x) - \frac{3}{8}\sin(4x) + \frac{4}{15}\sin(5x) - \ldots$$

Вычисления прекратить, если коэффициент перед синусом станет меньше ε = 0.001.

4.12. Определить значение S, вычислив первые n членов последовательности

$$S = \frac{b-2}{b} - \frac{b-4}{b-1} + \frac{b-8}{b-4} - \frac{b-16}{b-5} + \frac{b-32}{b-8} - \frac{b-64}{b-9} + \frac{b-128}{b-12} - \frac{b-256}{b-13} + \ldots$$

Вычисления прекратить, если разность абсолютных значений двух смежных членов последовательности не превышает $\varepsilon = 0.001$. Обратить внимание на проверку деления на нуль.

4.13. Определить минимальное количество *m* членов ряда Фибоначчи, среднее арифметическое значение которых превышает заданную величину *b*. Вычисление текущего среднего арифметического значения S_m производить методом накопления:

$$a_m = a_{m-1} + f_m; \quad S_m = \frac{1}{m} a_m;$$

$$m = 1, 2, 3, \ldots \quad ; \quad a_0 = 1$$

4.14. Определить количество и сумму чисел Фибоначчи, принадлежащих заданному интервалу $[a, b]$.

4.15. Заданное натуральное число n представить в виде суммы минимального количества различных чисел Фибоначчи.

4.16. Элементы монотонно убывающей числовой последовательности имеют следующие значения:

$$1 + \frac{1}{2}; \; 1 + \frac{1}{2} \cdot \frac{1}{3}; \; 1 + \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{1}{4}; \; \ldots$$

Определить номер ближайшего элемента этой последовательности, значение которого не превышает заданного числа b ($1 < b < 1.5$).

5. Строки

5.0. Общие замечания.

В рассматриваемых ниже задачах, как правило, предполагается следующее:

- обрабатываемый текст содержится в одной строке;
- слова текста разделяются между собой одним или несколькими пробелами;
- в начале и в конце строки могут быть пробелы;
- слово - это произвольная последовательность символов, отличных от символов-разделителей.

Если условие задачи в чем-либо отличается от вышеприведенного, об этом будет записано в тексте конкретной задачи.

В ряде программ используются функции *Space, NotSpace* и *Number,* описанные в разделе "Примеры обработки строк". В связи с этим повторное описание указанных функций в программах не производится.

5.1. Переставить буквы каждого слова в обратном порядке, определив одновременно, имеются ли в тексте слова-палиндромы, т.е. слова, которые читаются одинаково слева направо и справа налево.

```
Program Task501;
Var  S,                  { исходная строка }
     Sb1,Sb2 : string;   { буф.строки для анализируемого слова }
     i,j,
     k1,k2 : byte;       { левая и правая границы слова }
     SignPal,            { признак наличия палиндрома }
     Cond : boolean;     { условие окончания обработки текста }
     ch : char;          { символ строки }
{ ------------------------------------------------- }
```

```
Begin
  Readln(S); Writeln('S=',S);
  SignPal:=false;
  k2:=0; Cond:=true;
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(S,k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false;
            End;
          Sb1:=Copy(S,k1,k2-k1); Sb2:=Sb1;
          i:=1; j:=length(Sb2);
          While i<j do
            Begin
              ch:=Sb2[i]; Sb2[i]:=Sb2[j]; Sb2[j]:=ch;
              Inc(i); Dec(j);
            End;
          If Sb1=Sb2 then
            SignPal:=true;
          Delete(S,k1,k2-k1);
          Insert(Sb2,S,k1);
        End;
    End;
  Writeln('S=',S);
  If SignPal then
    Writeln('В строке есть слова-палиндромы')
  Else
    Writeln('В строке нет слов-палиндромов');
End.
```

В программе производится последовательное выделение слов текста путем определения границ *k1* и *k2*, где *k1* - первая буква слова (первый непробел), *k2* - первый пробел после конца слова.

Найденное слово, ограниченное значениями переменных *k1* и *k2,* копируется в буферные строки *Sb1* и *Sb2*. Буквы в слове *Sb2* переставляются в обратном порядке. Если теперь *Sb1 = Sb2*, то это означает, что данное слово является палиндромом.

Перестановка букв выполнена в строке *Sb2*, но не в строке *S*. Поэтому символы строки *Sb2* переписываются обратно в строку *S*.

5.2. Обменять местами первое и второе слова текста.

В программе Task502 с помощью функций *Space* и *NotSpace* определяются начало и конец первого (*k1, k2*) и второго (*k3, k4*) слов. Обработка строки выполняется, если в ней имеется не менее двух слов.

Решение задачи представлено в двух вариантах.

Последовательность обработки для варианта 1:

- удаление из строки *S* подстроки от начала первого до конца второго слова;

- вставка, начиная с позиции *k1*, второго слова;
- вставка пробелов между первым и вторым словами;
- вставка второго слова.

Вставляемые подстроки копируются из буферной строки *SBuf*, в которую предварительно была переписана исходная строка *S*.

Последовательность обработки для варианта 2:
- присваивание буферной строке *SBuf* пустого значения;
- добавление к строке *SBuf* пробелов перед первым словом строки *S;*
- добавление второго слова;
- добавление пробелов между первым и вторым словами;
- добавление первого слова;
- добавление символов строки *S*, расположенных после ее второго слова;
- присваивание строке *S* значения строки *SBuf*.

Вариант 1.

```
Program Task502;
Var  i,j,
     k1,k2,          { начало и конец первого слова }
     k3,k4 : byte;   { начало и конец второго слова }
     S,SBuf : string;
Begin
  Readln(S); Writeln('S=',S);
  If length(S)=0 then
    Writeln('Строка пустая ')
  Else
    Begin
      k1:=NotSpace(S,1);
      If k1=0 then
        Writeln('В строке только пробелы ')
      Else
        Begin
          k2:=Space(S,k1+1);
          If k2=0 then
            Writeln('В строке одно слово ')
          Else
            Begin
              k3:=NotSpace(S,k2+1);
              If k3=0 then
                Writeln('В строке одно слово  l = ',l)
              Else
                Begin
                  k4:=Space(S,k3+1);
                  If k4=0 then
                    k4:=length(S)+1;
                  SBuf:=S; Delete(S,k1,k4-k1);
                  Insert(Copy(SBuf,k3,k4-k3),S,k1);
                  Insert(Copy(SBuf,k2,k3-k2),S,k1+k4-k3);
                  Insert(Copy(SBuf,k1,k2-k1),S,k1+k4-k2);
                End;
            End;
        End;
    End;
  Writeln('S=',S);
End.
```

Вариант 2.

```
Определение  k1, k2, k3, k4
SBuf:='';
SBuf:=SBuf+Copy(S,1,k1-1);
SBuf:=SBuf+Copy(S,k3,k4-k3);
SBuf:=SBuf+Copy(S,k2,k3-k2);
SBuf:=SBuf+Copy(S,k1,k2-k1);
SBuf:=SBuf+Copy(S,k4,length(S)-k4+1);
S:=SBuf;
```

5.3. В строке содержится выражение, состоящее из букв, цифр, знаков операций и круглых скобок. Если количество открывающих скобок не равно количеству закрывающих, то избыточные внешние скобки удалить из текста.

Внешние открывающие скобки отсчитываются в выражении слева направо, закрывающие - справа налево.

В программе Task503 вначале определяются количество открывающих *k1* и количество закрывающих *k2* скобок. После этого в зависимости от их соотношения в цикле ***While*** производится удаление избыточных открывающих (при *k1 > k2*) или закрывающих (при *k1<k2*) скобок. Цикл ***While*** работает до тех пор, пока разность $dk = |k1-k2|$ не станет нулевой.

```
Program Task503;
Var   i,
      k1,             { кол-во открывающих скобок }
      k2,             { кол-во закрывающих скобок }
      dk : byte;      { dk = abs(k2-k1) }
      S  : string;
Begin
  Readln(S); Writeln('S=',S);
  k1:=0; k2:=0;
  For i:=1 to length(S) do
    If S[i]='(' then
      Inc(k1)
    Else
      If S[i]=')' then
        Inc(k2);
  dk:=abs(k2-k1);
  If k1>k2 then
    Begin
      i:=1;                       { Удаление избыточных }
      While dk>0 do               { открывающих скобок }
        If S[i]='(' then
          Begin
            Delete(S,i,1); Dec(dk)
          End
        Else
          Inc(i);
    End
  Else
    If k1<k2 then
      Begin
        i:=length(S);             { Удаление избыточных }
        While dk>0 do             { закрывающих скобок }
          Begin
```

```
            If S[i]=')' then
              Begin
                Delete(S,i,1); Dec(dk)
              End;
            Dec(i);
          End;
      End;
  Writeln('k1 = ',k1,'  k2 = ',k2,'  S=',S);
End.
```

5.4. Для записи слов используются только большие буквы латинского алфавита. Преобразовать строку, заменив каждую букву ее двухзначным номером в алфавите. Для пробела использовать значение '00'.

Например, для строки 'HTREK ESAB UYTR' получим
'0820180511000005190102000021252018' .

В строке-константе *Alf* записаны пробел и большие буквы латинского алфавита. В цикле ***For*** для каждого очередного символа строки *S* с помощью функции *Pos* определяется его позиция *k* в строке *Alf*. Тогда численное значение символа, записываемое в строку *S1*, равно *k*-1. Если это значение имеет лишь одну цифру, то перед нею дописывается символ '0'.

```
Program Task504;
Const  Alf = ' ABCDEFGHIJKLMNOPQRSTUVWXYZ';
Var   i,k  : byte;
      S,S1 : string;
      Sb : string[2];
Begin
  Readln(S); Writeln('S=',S);
  S1:='';
  For i:=1 to length(S) do
    Begin
      k:=Pos(S[i],Alf)-1;
      Str(k,Sb);
      If k<10 then
        Sb:='0'+Sb;
      S1:=S1+Sb;
    End;
  Writeln('S1=',S1);
End.
```

5.5. В строке записан список целых шестнадцатеричных чисел, разделенных между собой запятыми. Некоторые из этих чисел имеют знаки "+" или "-". Удалить из текста незначащие нули. Например, для текста

"5A787,+00DF890,0000672,-98,-003B,000,+89ADF00,+0,+011"

получим

"5A787,+DF890,672,-98,-3B,0,+89ADF00,+0,+11".

Для определения положения запятой в программе Task505 используется функция *Comma*. Ее выходное значение, присваиваемое переменной *k2*, - это конечная граница числа. Тогда начало следующего числа определяется значением *k2*+1 .

Исходный список шестнадцатеричных чисел содержится в строке *S*, в строку *S1* выделяется из *S* очередное число, в строке *S2* "складируются" обработанные шестнадцатеричные числа.

При обработке числа, записанного в строку *S1*, вначале определяется позиция *k* его первой цифры: $k = 2$, если число имеет знак, в противном случае $k = 1$. После этого в цикле **While** удаляются нулевые цифры числа до тех пор, пока в позицию *k* не будет установлена ненулевая цифра. Дополнительное условие *length(S1) > k,* проверяемое в цикле While, означает, что в числе должна оставаться хотя бы одна цифра (или знак и цифра). Это используется для того, чтобы в нулевом значении (например, в форме '0000') не были удалены все нули.

```
Program Task505;
Var  i,k,k1,k2  : byte;
     Cond : boolean;
     S,S1,S2 : string;
{ ------------------------------------------------- }
Function Comma(k:byte):byte;
{ Поиск ближайшей запятой, начиная с позиции k }
Var  i : byte;
Begin
  Comma:=0;
  For i:=k to length(S) do
    If S[i]=',' then
      Begin
        Comma:=i; Exit
      End;
End { Comma };
{ ------------------------------------------------- }
Begin
  Readln(S); Writeln('S=',S);
  Cond:=true; k2:=0; S2:='';
  While Cond do
    Begin
      k1:=k2+1; k2:=Comma(k1+1);
      If k2=0 then
        Begin
          k2:=length(S)+1; Cond:=false;
        End;
      S1:=Copy(S,k1,k2-k1); k:=1;
      If (S1[1]='+') or (S1[1]='-') then
        k:=2;
      While (S1[k]='0') and (length(S1)>k) do
        Delete(S1,k,1);
      S2:=S2+S1+',';
    End;
  Delete(S2,length(S2),1);
  S:=S2; Writeln('S=',S);
End.
```

5.6. Разделителями между словами текста являются: пробел, запятая, точка, двоеточие, точка с запятой, вопросительный и восклицательный знаки. Каждый из разделителей может быть окружен слева и справа одним или несколькими пробелами. Отредактировать текст следующим образом:

- в начале строки установить три пробела;
- удалить пробелы в конце строки;

- удалить пробелы, имеющиеся перед любым из разделителей (в том числе и пробелом);

- после каждого разделителя, кроме пробела, установить лишь один пробел.

Строка-константа *Separs* содержит список разделителей, причем первым из них является пробел.

Последовательность обработки исходной строки *S* :

- если после пробела следует любой другой разделитель, в том числе пробел, то пробел удаляется;

- если после разделителя, не являющегося пробелом (*Pos(S[i], Separs*) > 1), нет пробела, то производится вставка пробела после этого разделителя.

После выполнения первого этапа ни в начале, ни в конце строки не может быть более одного пробела. Дальнейшие действия:

- если в конце строки имеется пробел, то он удаляется;

- если в начале строки нет пробела, то в первую позицию строки вставляется пробел;

- в начало строки дописываются два пробела.

```
Program Task506;
Const  Separs = ' ,.:;?!';
Var  i : byte;
     S : string;
Begin
  Readln(S); Writeln('S=',S);
  i:=1;                          { Удаление избыточных }
  While i<length(S) do           { пробелов }
    If (S[i]=' ') and (Pos(S[i+1],Separs)>0) then
      Delete(S,i,1)
    Else
      Inc(i);
  i:=length(S)-1;
  While i>=1 do                   { Вставка пробела }
    Begin                         { после разделителя }
      If (Pos(S[i],Separs)>1) and (S[i+1]<>' ') then
        Insert(' ',S,i+1);
      Dec(i);
    End;
  If S[length(S)]=' ' then        { Удаление пробела }
    Delete(S,length(S),1);        { в конце строки }
  If S[1]<>' ' then               { Установка трех }
    S:=' '+S;                     { пробелов в начале }
  S:='  '+S; Writeln('S=',S);     { строки }
End.
```

5.7. Определить, имеются ли в тексте слова с повторяющимися символами.

Проверка наличия повторяющихся символов в очередном слове, скопированном в строку *S1*, производится функцией *RepeatCharacter*. Если повторение символов в каком-либо слове обнаружено, то дальнейшая обработка строки *S* не выполняется.

```
Program Task507;
Label  10;
Var  k1,k2 : byte;
     Cond,Result : boolean;
     S,S1 : string;
```

```
{ ------------------------------ }
Function RepeatCharacter:boolean;
{ Проверка наличия повторяющихся символов в слове }
Var  i,j : byte;
Begin
  RepeatCharacter:=false;
  For i:=1 to length(S1)-1 do
    For j:=i+1 to length(S1) do
      If S1[i]=S1[j] then
        Begin
          RepeatCharacter:=true; Exit
        End;
End { RepeatCharacter };
{ ------------------------------ }
Begin
  Readln(S); Writeln('S=',S);
  k2:=0; Result:=false; Cond:=true;
  While Cond do
    Begin
      k1:=NotSpace(k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false
            End;
          S1:=Copy(S,k1,k2-k1);
          Result:=RepeatCharacter;
          If Result then
            Goto 10;
        End;
    End;
  10:
  If Result then
    Writeln('В тексте имеются слова с повт.символами')
  Else
    Writeln('В тексте нет слов с повт.символами');
End.
```

5.8. В строке записан строчными буквами текст на русском языке. Определить количество слов, содержащих удвоенные гласные.

Список гласных содержится в строке-константе *Vocal*. Проверку наличия двух подряд стоящих одинаковых гласных выполняет функция *DoubleVocal*. Если выходное значение этой функции равно *true*, то производится добавление единицы в счетчик *Count*.

```
Program Task508;
Const  Vocal = 'аеиоуыэюя';
Var  k1,k2,Count : byte;
     Cond : boolean;
     S,S1 : string;
{ ------------------------------ }
```

```
Function DoubleVocal:boolean;
{ Проверка удвоения гласных в слове }
Var  i,p : byte;
Begin
  DoubleVocal:=false;
  For i:=1 to length(S1)-1 do
    Begin
      p:=Pos(S1[i],Vocal);
      If (p>0) and (S1[i]=S1[i+1]) then
        Begin
          DoubleVocal:=true; Exit
        End;
    End;
End { DoubleVocal };
{ ------------------------------ }
Begin
  Read(S); Writeln('S=',S);
  Count:=0; k2:=0; Cond:=true;
  While Cond do
    Begin
      k1:=NotSpace(k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false
            End;
          S1:=Copy(S,k1,k2-k1);
          If DoubleVocal then
            Inc(Count);
        End;
    End;
  Writeln('Count=',Count);
End.
```

5.9. В строке записан некоторый текст, в состав которого входят слова и целые десятичные числа без знака. Заменить каждое число суммой его цифр. В результирующей строке оставить между словами по одному пробелу, в начале и в конце строки пробелы удалить.

Список десятичных цифр записан в строке-константе *Digit*. Если в очередном слове первый символ *S*[*k1*] - цифра, то производится обработка этого слова как числа. Для этого после переписи его в строку *S1* осуществляется с помощью процедуры *DigitsSumma* суммирование цифр и формирование символьной записи нового числа в строке *S2*. Если длина строки *S2* меньше, чем строки *S1*, то строка *S2* дополняется соответствующим количеством пробелов. После этого исходное число удаляется из строки *S*, а новое число вставляется в эту строку. На последнем этапе работы программы производится удаление избыточных пробелов в соответствии с условием задачи.

Если очередное число состоит из одной цифры, то замена такого числа в строке *S* не производится.

```
Program Task509;
Const  Digit = '0123456789';
```

```
Var  i,k1,k2,p : byte;
     Cond : boolean;
     S,S1,S2 : string;
{ ------------------------------------------------- }
Procedure DigitsSumma;
{ Определение суммы цифр числа }
Var  i,Dig : byte;
     Code,Sum : integer;
Begin
  Sum:=0;
  For i:=1 to length(S1) do
    Begin
      Val(S1[i],Dig,Code); Sum:=Sum+Dig;
    End;
  Str(Sum,S2);
  If length(S2)<length(S1) then
    For i:=1 to length(S1)-length(S2) do
      S2:=S2+' ';
End { DigitsSumma };
{ ------------------------------------------------- }
Begin
  Readln(S); Writeln('S=',S);
  k2:=0; Cond:=true;
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(S,k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false
            End;
          p:=Pos(S[k1],Digit);
          If p>0 then
            Begin
              S1:=Copy(S,k1,k2-k1);
              If length(S1)>1 then
                Begin
                  DigitsSumma;
                  Delete(S,k1,k2-k1); Insert(S2,S,k1);
                End;
            End;
        End;
    End;
  i:=1;
  While i<length(S) do
    If (S[i]=' ') and (S[i+1]=' ') then
      Delete(S,i,1)
    Else
      Inc(i);
  If S[1]=' ' then
    Delete(S,1,1);
  If S[length(S)]=' ' then
    Delete(S,length(S),1);
```

```
  Writeln('S=',S);
End.
```

5.10. В строке записан текст на русском языке. В составе текста могут быть целые десятичные числа без знака. Для записи слов используются как прописные, так и строчные буквы. Определить количество слов, в которых все буквы различные.

После выделения слова (не числа) производится преобразование малых букв в большие. Поскольку в слове используются не латинские буквы, то такое преобразование не может быть выполнено с помощью функции *UpCase*. Для этого в программе Task510 используется строка-константа с перечнем малых букв алфавита и символьный массив с перечнем больших букв (строку-константу нельзя индексировать).

Если в слове нет повторяющихся букв, то выходное значение функции *DifLetters* будет равным *true*; в этом случае производится добавление единицы в счетчик слов *Count*.

```
Program Task510;
Const  Digits = '0123456789';
Var  k1,k2,Count : byte;
     Cond : boolean;
     S,S1 : string;
{ ------------------------------------------------- }
Function DifLetters:boolean;
{ true, если в слове S1 нет повторяющихся букв }
Const  Alf = 'абвгдежзийклмнопрстуфхцчшщэыъьюя';
    AlfAr : array[1..32] of char =('А','Б','В','Г','Д','Е',
      'Ж','З','И','Й','К','Л','М','Н','О','П','Р','С','Т','У',
      'Ф','Х','Ц','Ч','Ш','Щ','Э','Ы','Ъ','Ь','Ю','Я');
Var  i,j,k,ls : byte;
Begin
  DifLetters:=true; ls:=length(S1);
  For i:=1 to ls do              { преобразование строчных }
    Begin                        { букв в прописные        }
      k:=Pos(S[i],Alf);
      If k>0 then
        S[i]:=AlfAr[k];
    End;
  For i:=1 to ls-1 do            { проверка наличия }
    For j:=i+1 to ls do          { повторяющихся букв }
      If S1[i]=S1[j] then
        Begin
          DifLetters:=false; Exit
        End;
End { DifLetters };
{ ------------------------------------------------- }
Begin
  Readln(S); Writeln('S=',S);
  k2:=0; Cond:=true; Count:=0;
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(S,k1+1);
```

```
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false
            End;
          S1:=Copy(S,k1,k2-k1);
          If (Pos(S1[1],Digits)=0) and DifLetters then
            Inc(Count);
        End;
    End;
  Writeln('Count=',Count);
End.
```

5.11. В строке записан строчными буквами текст на русском языке. Определить значение и порядковый номер слова, содержащего максимальное количество согласных. Учесть, что в частном случае в состав слова могут входить цифры.

В программе Task511 используется список согласных, записанный в строке-константе *Consonants*. Определение количества согласных в очередном слове, переписанном из строки *S* в строку *S1*, производит функция *NumCons*. Результатом работы программы является печать порядкового номера слова *pmax* и количества содержащихся в нем согласных *ConsMax*.

```
Program Task511;
Const  Consonants = 'бвгджзклмнпрстфхцчшщ';
Var  k1,k2,
     p,           { порядковый номер слова }
     pmax,        { пор.номер слова с макс.кол-вом согласных }
     Cons,        { кол-во согласных в слове }
     ConsMax : byte; { макс.кол-во согласных в слове }
     Cond : boolean;
     S,S1 : string;
{ ------------------------------ }
Function NumCons:byte;
{ Определение кол-ва согласных в слове S1 }
Var  i,k,p : byte;
Begin
  k:=0;
  For i:=1 to length(S1) do
    Begin
      p:=Pos(S1[i],Consonants);
      If p>0 then
        Inc(k);
    End;
  NumCons:=k;
End { NumCons };
{ ------------------------------ }
Begin
  Read(S); Writeln('S=',S);
  k2:=0; p:=0; pmax:=0;
  Cond:=true; ConsMax:=0;
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
```

```
        Begin
          k2:=Space(S,k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false
            End;
          Inc(p); S1:=Copy(S,k1,k2-k1);
          Cons:=NumCons;
          If Cons>ConsMax then
            Begin
              pmax:=p; ConsMax:=Cons;
            End;
        End;
    End;
  Writeln('ConsMax=',ConsMax,'  pmax=',pmax);
End.
```

5.12. В строке записан список идентификаторов и целых десятичных чисел. Элементы списка разделены между собой запятыми. Определить порядковый номер элемента, являющегося идентификатором с максимальным количеством цифр.

Как известно, идентификатор - это последовательность букв и цифр, начинающаяся с буквы. Поэтому для его индикации в программе Task512 производится проверка первого символа очередного слова *S1,* выделенного из исходной строки *S.*

В задаче не требуется определять, какая именно цифра содержится в числе или в идентификаторе, здесь достаточно произвести проверку, является ли заданный символ цифрой или не цифрой. Поэтому для такой проверки вместо строки-константы используется множество-константа *Digits.* Непосредственный подсчет количества цифр в идентификаторе выполняет функция *DigitsNumber.*

```
Program Task512;
Const  Digits = ['0'..'9'];
Var  k1,k2,
     Dig,               { кол-во цифр в идентификаторе }
     DigMax,            { макс.кол-во цифр }
     OrdNumber,         { порядковый номер ид-ра }
     OrdMax : byte;     { номер ид-ра с макс.кол-вом цифр }
     Cond : boolean;
     S,S1,Smax : string;
{ ------------------------------------------------- }
Function DigitsNumber:byte;
{ Определение количества цифр в идентификаторе }
Var  i,k,ls : byte;
Begin
  k:=0; ls:=length(S1);
  For i:=1 to ls do
    If S1[i] in Digits then
      Inc(k);
  DigitsNumber:=k;
End { DigitsNumber };
{ ------------------------------------------------- }
Begin
  Readln(S); Writeln('S=',S);
  Digmax:=0; OrdNumber:=0;
  Cond:=true; k2:=0;
```

```
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(S,k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false
            End;
          Inc(OrdNumber); S1:=Copy(S,k1,k2-k1);
          If not(S1[1] in Digits) then
            Begin
              Dig:=DigitsNumber;
              If Dig>DigMax then
                Begin
                  DigMax:=Dig; OrdMax:=OrdNumber;
                End;
            End;
        End;
    End;
  Writeln('DigMax=',DigMax,'  OrdMax=',OrdMax);
End.
```

5.13. В строке записан строчными буквами текст на русском языке. Определить количество слов, содержащих свыше двух подряд идущих согласных букв.

Программа Task513 во многом аналогична программе Task511, но вместо подсчета количества согласных проверяется, имеются ли в слове три подряд идущие согласные, для чего используется функция *Cons3*.

```
Program Task513;
Const  Consonants = 'бвгджзклмнпрстфхцчшщ';
Var  k1,k2,NumWords : byte;
     Cond : boolean;
     S,S1 : string;
{ ------------------------------------------------- }
Function Cons3:boolean;
{ Проверка наличия в слове трех последовательных согласных }
Var  i,p1,p2,p3 : byte;
Begin
  Cons3:=false;
  For i:=1 to length(S1)-2 do
    Begin
      p1:=Pos(S1[i],Consonants); p2:=Pos(S1[i+1],Consonants);
      p3:=Pos(S1[i+2],Consonants);
      If (p1>0) and (p2>0) and (p3>0) then
        Begin
          Cons3:=true; Exit
        End;
    End;
End { Cons3 };
{ ------------------------------------------------- }
```

```
Begin
  Readln(S); Writeln('S=',S);
  k2:=0; NumWords:=0; Cond:=true;
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(S,k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false
            End;
          S1:=Copy(S,k1,k2-k1);
          If Cons3 then
            Inc(NumWords);
        End;
    End;
  Writeln('NumWords=',NumWords);
End.
```

5.14. Задан произвольный массив неповторяющихся натуральных чисел. Записать эти числа в строке, используя диапазоны. В диапазон включать группу, содержащую не менее трех чисел. Числа в строке должны быть упорядочены по возрастанию. Элементы строки разделять одним пробелом. Например, для массива

X = (15,88,36,12,14,89,37,13,8,87,11,90,91,35,77,44,10,7,86)

получим:

S = '7 8 10..15 35..37 44 77 86..91' .

В программе Task514 обрабатывается массив, сгруппированный по возрастанию. В связи с этим на начальном этапе работы программы с помощью процедуры *Buble*, реализующей метод "пузырька", производится группировка исходного массива *X*. Накапливание диапазонного списка чисел в символьном виде осуществляется в строке *S*, которой первоначально присваивается пустое значение.

Для выделения из массива *X* отрезка натурального ряда чисел, т.е. группы чисел, в которой разница между смежными элементами равна 1, используются переменные *k1* и *k2*: *k1* указывает индекс первого числа очередной группы (начало группы), *k2* определяет конец группы (индекс ближайшего числа, не принадлежащего группе).

Для определения конца группы (значения переменной *k2*) используется функция *SignEnd*. Тогда начало следующей группы *k1* принимается равным *k2* (для первой группы k1 = 1).

Если для очередной группы имеет место *k2* - *k1* > 2, то это означает, что числа *x*[*k1*] и *x*[*k2-1*] являются границами диапазона. Тогда после преобразования их к символьному виду при записи в строку *S* между указанными числами включается подстрока '..' , в противном случае они разделяются в строке *S* пробелом.

```
Program Task514;
Const  Nmax = 200;
Type   Ar = array[1..Nmax] of word;
Var    k1,k2,n : byte;
       Cond : boolean;
```

```
        X : Ar;
     S,S1 : string;
{ ------------------------------------------------- }
Procedure Buble;
{ Группировка массива по возрастанию }
Var  i,m : byte;
       R : word;
    Cond : boolean;
Begin
  Cond:=true; m:=n-1;
  While Cond do
    Begin
      Cond:=false;
      For i:=1 to m do
        If x[i]>x[i+1] then
          Begin
            R:=x[i]; x[i]:=x[i+1]; x[i+1]:=R;
            Cond:=true;
          End;
      Dec(m);
    End;
End { Buble };
{ ------------------------------------------------- }
Function SignEnd(k:byte):byte;
{ Поиск конца диапазона, начиная с элемента x[k] }
Var  i : byte;
Begin
  SignEnd:=0;
  For i:=k+1 to n do
    If x[i]-x[i-1]>1 then
      Begin
        SignEnd:=i; Exit
      End;
End { SignEnd };
{ ------------------------------------------------- }
Begin
  В в о д   и   п е ч а т ь   n, X
  Buble;
  S:=''; k2:=1; Cond:=true;
  While Cond do
    Begin
      k1:=k2; k2:=SignEnd(k1);
      If k2=0 then
        Begin
          k2:=n+1; Cond:=false;
        End;
      If k2-k1>2 then
        Begin
          Str(x[k1],S1); S:=S+S1+'..';
          Str(x[k2-1],S1); S:=S+S1+' ';
        End
      Else
        Begin
          Str(x[k1],S1); S:=S+S1+' ';
        End;
    End;
  Writeln('S = ',S);
```

```
End.
```

5.15. В строке записан текст телеграммы. В качестве разделителей слов используются пробел, запятая и точка. Запятая обозначается символами '*зпт*', точка - символами '*тчк*'. С клавиатуры вводится цена одного слова в копейках. Определить стоимость телеграммы (запятая "*зпт*" и точка "*тчк*" словами не считаются). Ответ отпечатать в виде "*т грн k коп*".

Выделение слов в строке *S*, как и ранее, производится с помощью функций *Space* и *NotSpace*. Если значение очередного слова не равно '*тчк*' или '*зпт*', то к счетчику слов *n* добавляется единица. Стоимость телеграммы *Cost* - это произведение значения *n* на цену одного слова *Price*. Целочисленная часть от деления *Cost* на 100 - это количество гривней, остаток от деления на 100 - количество копеек.

```
Program Task515;
Var  k1,k2,
     n,                { количество слов }
     Price : byte;     { цена одного слова, коп. }
     Cost : word;      { стоимость телеграммы, коп. }
     Cond : boolean;
     S,S1,S2 : string;
Begin
  Readln(S); Read(Price);
  k2:=0; Cond:=true; n:=0;
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(S,k1+1);
          If k2=0 then
            Begin
              k2:=length(S)+1; Cond:=false;
            End;
          S1:=Copy(S,k1,k2-k1);
        End;
      If (S1<>'тчк') and (S1<>'зпт') then
        Inc(n);
    End;
  Cost:=n*Price; S1:='Стоимость телеграммы  ';
  Str(Cost div 100,S2); S1:=S1+S2+' грн ';
  Str(Cost mod 100,S2); S1:=S1+S2+' коп ';
  Writeln('S1 = ',S1);
End.
```

5.16. Расстояние между двумя словами равной длины - это количество позиций, в которых различаются эти слова. Для заданной строки текста указать номера и значения слов с максимальным расстоянием между ними.

По условию задачи требуется сравнивать все пары слов:

```
1 - 2    1 - 3    1 - 4    1 - 5    1 - n
         2 - 3    2 - 4    2 - 5    2 - n
```

```
                    3 - 4    3 - 5    3 - n
                    ......................
                                   (n - 1) - n
```

Здесь *n* – количество слов в строке (значение *n* заранее неизвестно).

В общем случае для пары (*i* - *j*), где *i, j* - номера слов в строке, переменные *i* и *j* должны изменяться в следующих диапазонах: $i = 1 .. (n-1)$, $j = i+1 .. n$.

В программе Task516 для перебора слов используются два цикла: во внешнем цикле, управляемом булевской переменной *Cond1*, выбирается первое слово, во внутреннем, управляемом переменной *Cond2,* - второе слово сравниваемой пары слов. Вычисление расстояния между словами, скопированными в строки *Sb1* и *Sb2,* производится, если количество символов в сравниваемых словах одинаково, т.е. одинаковы длины строк *Sb1* и *Sb2.* В программе учтено, что количество слов *n* заранее неизвестно.

```
Program Task516;
Var  S,              { исходная строка }
     Sb1max,Sb2max,  { слова с макс.расстоянием между ними }
     Sb1,Sb2 : string; { буф.строки для анализируемых слов }
     i,              { параметр цикла }
     k1,k2,          { левая и правая границы первого слова }
     k3,k4,          { левая и правая границы второго слова }
     n1,n2,          { текущие номера слов }
     np1,np2,        { номера слов с макс.расстоянием }
     d,dmax : byte;  { текущее и макс.расст-я между словами }
     Res,            { признак наличия слов одинак.длины }
     Cond1,          { переменная для упр-я внешним циклом }
     Cond2 : boolean;{ то же для внутреннего цикла }
Begin
  Readln(S);
  dmax:=0; Res:=false; n1:=0;
  k2:=0; Cond1:=true;
  While Cond1 do              { Выделение первого }
    Begin                     { из пары сравниваемых }
      k1:=NotSpace(k2+1);     { слов }
      If k1=0 then
        Cond1:=false
      Else
        Begin
          k2:=Space(k1+1);
          If k2=0 then
            Cond1:=false
          Else
            Begin
              Inc(n1); Sb1:=Copy(S,k1,k2-k1);
              For i:=1 to length(Sb1) do
                Sb1[i]:=UpCase(Sb1[i]);
              k4:=k2+1; Cond2:=true; n2:=n1;
              While Cond2 do      { Выделение второго }
                Begin             { из пары сравниваемых }
                  k3:=NotSpace(k4+1); { слов }
                  If k3=0 then
                    Cond2:=false
                  Else
                    Begin
                      k4:=Space(k3+1);
                      If k4=0 then
                        Begin
```

```
                        k4:=length(S)+1; Cond2:=false;
                      End;
                    Inc(n2); Sb2:=Copy(S,k3,k4-k3);
                    If length(Sb1)=length(Sb2) then
                      Begin
                        Res:=true; d:=0;
                        For i:=1 to length(Sb2) do
                          Sb2[i]:=UpCase(Sb2[i]);
                        For i:=1 to length(Sb1) do
                          If Sb1[i]<>Sb2[i] then
                            Inc(d);
                        If d>dmax then
                          Begin
                            dmax:=d; np1:=n1; np2:=n2;
                            Sb1max:=Sb1; Sb2max:=Sb2;
                          End;
                      End;
                  End;
              End;
          End;
      End;
    If not Res then
      Writeln('В строке нет слов одинаковой длины')
    Else
      Begin
        Writeln('dmax = ',dmax);
        Writeln('np1 = ',np1,'   Sb1max = ',Sb1max);
        Writeln('np2 = ',np2,'   Sb2max = ',Sb2max);
      End;
  End.
```

5.17. Сгруппировать слова в тексте в порядке уменьшения их длин, разделив между собой одним пробелом. Например, для исходного текста

aa bbbbbb uuu rrrrrrr ttttttttt e vvvv nnnnnnnnnn ss

получим

nnnnnnnnnn ttttttttt rrrrrrr bbbbbb vvvv uuu aa ss e .

Слово минимальной длины содержит один символ. С учетом пробелов между словами максимальное количество слов в строке может быть 128.

В программе Task517 при последовательном просмотре строки *S* формируется массив записей *Params*, *i*-ая компонента которого содержит два элемента: начальную позицию *k1* *i*-го слова и его длину *l*.

После формирования массива *Params* с *n* компонентами производится его группировка по убыванию параметра *l*. После этого в соответствии с условием задачи создается новая строка.

```
Program Task517;
Const  Nmax = 128;
Type   ParamType = record
           k1,l : byte;
       end;
       ParamAr = array[1..Nmax] of ParamType;
Var   S,S1 : string;
      i,k1,k2,l,n : byte;
```

```
      Cond : boolean;
      ch : char;
      Param : ParamType;
      Params : ParamAr;
{ ------------------------------------------------- }
Procedure Buble;
{ Группировка массива Params по убыванию длины слова l }
Var  i,m : byte;
     Cond : boolean;
Begin
  Cond:=true; m:=n-1;
  While Cond do
    Begin
      Cond:=false;
      For i:=1 to m do
        If Params[i].l<Params[i+1].l then
          Begin
            Param:=Params[i]; Params[i]:=Params[i+1];
            Params[i+1]:=Param; Cond:=true;
          End;
      Dec(m);
    End;
End { Buble };
{ ------------------------------------------------- }
Begin
  Readln(S); Writeln('S=',S);
  n:=0; k2:=0; Cond:=true;
  While Cond do
    Begin
      k1:=NotSpace(S,k2+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=Space(S,k1+1); { Запись в массив }
          If k2=0 then       { Params нач.позиции }
            Begin            { и длины каждого слова }
              k2:=length(S)+1; Cond:=false;
            End;
          l:=k2-k1; Inc(n);
          Param.k1:=k1; Param.l:=l;
          Params[n]:=Param;
        End;
    End;
  Buble;
  S1:='';                    { Перепись в строку S1 }
  For i:=1 to n do           { слов исходного текста }
    Begin                    { в порядке уменьшения }
      k1:=Params[i].k1; l:=Params[i].l;   { их длин }
      S1:=S1+Copy(S,k1,l)+' ';
    End;
  Delete(S1,length(S1),1);
  S:=S1; Writeln('S=',S);
End.
```

5.18. Даны две строки, содержащие цифры двух длинных целых десятичных чисел. Найти сумму этих чисел.

Алгоритм решения задачи 5.18 аналогичен алгоритму, реализованному в программе 3.09 по отношению к числовым массивам цифр, но его реализация в программе Task518 трансформирована для символьного представления цифр.

```
Program Task518;
Const  Digits = '0123456789';
Var  i,l1,l2,ls,dl,Dig1,Dig2,
     DigSum,p : byte;
     Cond : boolean;
     S1,S2,Sum : string;
     Sb : string[1];
Begin
  Readln(S1); Readln(S2);
  l1:=length(S1); l2:=length(S2);
  dl:=abs(l2-l1);
  If l1>l2 then
    For i:=1 to dl do
      S2:='0'+S2;
  If l1<l2 then
    For i:=1 to dl do
      S1:='0'+S1;
  If l1>l2 then
    ls:=l1
  Else
    ls:=l2;
  Sum:=''; p:=0;
  For i:=ls downto 1 do
    Begin
      Dig1:=Pos(S1[i],Digits)-1; Dig2:=Pos(S2[i],Digits)-1;
      DigSum:=Dig1+Dig2+p;
      If DigSum>=10 then
        Begin
          DigSum:=DigSum-10; p:=1;
        End
      Else
        p:=0;
      Str(DigSum,Sb); Sum:=Sb+Sum;
    End;
  If p=1 then
    Sum:='1'+Sum;
  Writeln('Sum = ',Sum);
End.
```

5.19. Даны две строки, содержащие цифры двух длинных целых десятичных чисел. Найти разность этих чисел.

Алгоритм решения задачи 5.19 аналогичен алгоритму, реализованному в программе 3.10 по отношению к числовым массивам цифр, но его реализация трансформирована для символьного представления цифр.

```
Program Task519;
Const  Digits = '0123456789';
```

```
       DigAr : array[1..10] of char =
             ('0','1','2','3','4','5','6','7','8','9');
Var  i,j,l1,l2,ls,dl,k,k1,
     Dig1,Dig2,DigSub : byte;
     CondMinus,Cond : boolean;
     S1,S2,Sub : string;
     Sb : string[1];
Begin
  Readln(S1); Readln(S2);
  l1:=length(S1); l2:=length(S2); dl:=abs(l2-l1);
  If l1>l2 then
    For i:=1 to dl do
      S2:='0'+S2;
  If l1<l2 then
    For i:=1 to dl do
      S1:='0'+S1;
  If l1>l2 then
    ls:=l1
  Else
    ls:=l2;
  CondMinus:=false;
  If S1[1]<S2[1] then
    Begin
      Sub:=S1; S1:=S2; S2:=Sub;
      CondMinus:=true;
    End;
  Sub:='';
  For i:=ls downto 1 do
    Begin
      Dig1:=Pos(S1[i],Digits)-1; Dig2:=Pos(S2[i],Digits)-1;
      If Dig1<Dig2 then
        Begin
          For j:=i-1 downto 1 do
            If S1[j]>'0' then
              Begin
                k1:=Pos(S1[j],Digits); S1[j]:=DigAr[k1-1];
                For k:=j+1 to i-1 do
                  S1[k]:='9';
                Dig1:=Dig1+10; Goto 10;
              End;
          10:
        End;
      DigSub:=Dig1-Dig2;
      Str(DigSub,Sb); Sub:=Sb+Sub;
    End;
  If CondMinus then
    Sub:='-'+Sub;
  Writeln('Sub = ',Sub);
End.
```

Задания для самостоятельной работы

5.20. По отношению к строке с объявленной длиной n символов требуется:
- в начале строки установить три пробела;

- если текущая длина строки l меньше значения n, то добавить в конце строки $n-l$ пробелов;

- сдвинуть последнее слово к концу строки;

- выравнять промежутки между словами так, чтобы количества пробелов между ними были равными или отличались между собой не более чем на единицу.

5.21. В строке записаны слова и целые десятичные числа. Для записи слов используются прописные и строчные русские буквы. Длина слова не превышает $m \le 16$ букв. Сформировать словарь слов и отпечатать его в алфавитном порядке. В словаре каждое слово должно начинаться с прописной буквы.

5.22. Удалить из текста слова с четными порядковыми номерами, а в остальных словах переставить буквы в обратном порядке.

5.23. Слова, состоящие не более чем из двух букв, слить вместе в последовательности, противоположной их расположению в тексте, одновременно удалив их из текста. Сформированное таким образом новое слово записать в начале строки, отделив от остальной ее части одним пробелом, если в начале строки не было пробела.

5.24. В каждом слове текста удалить минимальное количество символов так, чтобы в преобразованном слове не было повторяющихся символов.

5.25. В строке записана произвольная последовательность символов $s_1, s_2, ..., s_n$. Добавить к строке минимальное количество $m < n$ символов так, чтобы последовательность $s_1 ... s_n ... s_{m+n}$ стала палиндромом. Прни этом допускается перестановка символов исходной строки.

5.26. В строке S записаны слова исходного текста, в отдельной строке S_1 - одно из слов, которое может быть в строке S (количество букв в S_1 не превышает 10). В словах исходного текста могут быть ошибки, в частности, могут быть переставлены две соседние буквы. Проверить текст в строке S и, если в нем имеется слово S_1 с наличием указанной ошибки, то необходимо скорректировать это слово.

5.27. То же, что 5.26, но в качестве возможной ошибки рассматривать пропуск одной буквы.

5.28. То же, что 5.26, но в качестве возможной ошибки рассматривать замену одной буквы.

5.29. Символами строки являются большие и малые латинские буквы. Удалить из состава строки последовательности '*abcd*', в составе которых могут быть как большие, так и малые буквы.

5.30. Строка содержит произвольные символы таблицы ASCII. Удалить из строки все цифры и повторить дважды каждый символ, кроме пробела, не являющийся цифрой. Просмотр строки осуществлять один раз.

5.31. Строка имеет объявленную длину n символов. Заменить в тексте строки слова "*child*" словами "*children*". Если при этом длина строки превысит n символов, избыточные слова разместить в новой строке, не используя знак переноса части слова. Буферную строку не использовать.

5.32. В строке записано арифметическое выражение, состоящее из идентификаторов, целых десятичных чисел, знаков операций и круглых скобок. Внутренними скобками считается такая пара '(' и ')', внутри которой нет других скобок. Требуется удалить из строки содержимое внутренних скобок (вместе со скобками). Учесть, что в строке может быть несколько пар внутренних скобок.

5.33. В строке содержится список десятичных чисел, содержащих целую и дробную части, разделенные точкой. Элементы списка разделены запятыми. Если в дробной части числа свыше двух цифр, округлить ее до двух цифр, сократив соответственно длину строки.

5.34. В строке содержится список десятичных чисел, содержащих целую и дробную части, разделенные точкой. Элементы списка разделены запятыми. Требуется удалить незна-

чащие нули в целой и дробной частях, если они имеются. Если при этом полностью удаляется дробная часть, то удалению подлежит также разделяющая их точка. В частности, число '19.000' после этого должно принять вид '19'.

5.35. Если в строке имеются последовательности, состоящие из более чем трех одинаковых символов, то заменить каждую такую последовательность подстрокой mka, где m - признак повторения, k - двухзначное число, определяющее количество повторений, a - повторяющийся символ. В качестве признака повторений взять неиспользуемый в тексте символ #14. Составить две процедуры, выполняющие соответственно сжатие и восстановление текста.

5.36. Заданы две строки S_1 и S_2. Удалить из строки S_1 символы, содержащиеся в строке S_2.

5.37. Строка содержит произвольные символы таблицы ASCII. Найти наиболее длинную последовательность цифр и удалить ее из строки.

5.38. Определить, имеется ли в тексте хотя бы одно слово, которое является перестановкой букв другого слова этого же текста (например, "рост" и "сорт", "огород" и "дорого"). Отпечатать номера и значения такой пары слов.

5.39. В словах текста используются лишь большие и малые латинские буквы. Определить количества буквосочетаний из двух букв (aa, ab, ac и т.д.). Отпечатать 10 буквосочетаний с максимальной частотой (по убыванию частоты).

Указание. Рекомендуется объявить матрицу *A* типа

```
array['a'..'z','a'..'z'] of byte,
```

обнулив ее элементы на начальном этапе работы программы.

Тогда, выделив из слова очередную пару букв, например '*ps*', необходимо лишь добавить единицу к элементу *a*['*p*','*s*'].

5.40. В строке записаны слова и целые положительные числа, не превышающие значения 3000. Заменить каждое число его представлением в римской системе счисления.

5.41. Кроме слов, в строке содержатся числа, записанные в римской системе счисления. Признаком такого числа является то, что все его символы могут состоять только из больших латинских букв I, V, X, L, C, D, M. Заменить каждое такое число его десятичным представлением.

5.42. В строке малыми латинскими буквами записаны слова, разделяющиеся запятыми. Первые буквы некоторых слов могут совпадать. Указать минимальное количество первых букв, по которым можно различить слова из заданного списка.

5.43. Ученики зашифровывают свои записки, записывая все слова наоборот и расставляя их в предложении в обратном порядке. В строке содержится несколько предложений, при этом точка установлена справа от первого слова. Составить программу, позволяющую адресату прочесть записку без ручной дешифрации.

5.44. В строке записано выражение, состоящее из идентификаторов, знаков арифметических операций и круглых скобок. Между элементами выражения могут быть пробелы. При записи выражения возможны следующие ошибки:

- несоответствие количества открывающих и закрывающих скобок;
- закрывающая скобка расположена до открывающей;
- отсутствуют содержательные символы между скобками, т.е. символы, отличные от пробела и знаков операций.

Определить наличие каждого типа ошибки и, если это возможно, ее местоположение. При обнаружении любой ошибки дальнейшая проверка строки прекращается.

5.45. Список фамилий, разделенных запятыми, записан в строке в произвольном порядке. Упорядочить его по алфавиту и записать в ту же строку.

5.46. Определить, является ли заданный текст палиндромом (пробелы во внимание не принимать).

5.47. Произвести циклический сдвиг текста вправо на одно слово.

5.48. Для записи текста использованы большие и малые латинские буквы, цифры и разделители. Заменить каждую букву новым значением, циклически сдвинутым по алфавиту вправо на k позиций (k задано).

Например, при $k = 5$ буква 'a' будет заменена буквой 'f', а буква 'w' - буквой 'b'.

5.49. В строке записан список идентификаторов и целых десятичных чисел, разделенных запятыми. Определить, имеются ли в этом списке одинаковые идентификаторы. Если длины сравниваемых идентификаторов превышают 5 символов, то считать их одинаковыми, если совпадают первые 5 символов.

5.50. Определить, имеются ли в тексте слова, составленные из одних и тех же букв, при этом количество повторений каждой буквы во внимание не принимается (например, "*малина"* и *"налим").*

5.51. Для каждого из слов указать, сколько раз оно встречается в тексте.

5.52. Отредактировать текст, удалив из него слова, которые уже встречались в нем раньше.

5.53. Каждое из двух предложений текста записано в отдельной строке. Определить:

- самое длинное общее слово двух предложений;
- самое короткое слово первого предложения, отсутствующее во втором предложении.

5.54. Удалить из текста слова, состоящие из вхождений не более чем двух букв, при этом прописные и строчные буквы считаются эквивалентными (например, '*Akka*', *'rTTrt*' и т.п.).

5.55. Строка содержит список идентификаторов и целых десятичных чисел без знака. Элементы строки разделены запятыми. Требуется заменить каждое число его цифровым корнем. Например, для строки

abcd,56743,55,erfghh,45632,ghd45g,3052675981,ASj7,N4,7

получим

abcd,7,1,erfghh,2,ghd45g,4,ASj7,N4,7 .

Примечание. Цифровым корнем называют такую сумму цифр целого числа, которая после некоторого количества их сложений изображается одной цифрой. Пример:

$$30526375981 \to 3+0+5+2+6+3+7+5+9+8+1=49 \to 4+9=13 \to 1+3=4$$

5.56. В строке записан список натуральных чисел, превышающих значение 10^{11}, т.е. максимально возможное значение для типа longint. Элементы списка разделены между собой запятыми. Определить, сколько чисел в этом списке делится без остатка на 3, 4 или 5.

5.57. В строке записан список натуральных чисел, превышающих значение 10^{11}, т.е. максимально возможное значение для типа longint. Элементы списка разделены между собой запятыми. Определить, сколько чисел в этом списке делится без остатка на 7.

Указание. Признак делимости на 7 формулируется следующим образом. Цифры исходного числа необходимо разделить справа налево на триады и объединить полученные числа знаками операций “-”, “+”, ”-”, “+” и т.д. Последняя триада может быть неполной. Если полученное при этом число делится без остатка на 7, то это и является признаком делимости исходного числа на 7.

Пример. Для числа 6999994557112662 получим

$$662 - 112 + 557 - 994 + 999 - 6 = 1106 \Rightarrow 1106 : 7 = 158.$$

5.58. Заданы две строки $S1$ и $S2$, в каждой из которых слова разделены между собой одним или несколькими пробелами. Требуется сжать эти строки, удалив из них избыточные пробелы, после чего определить, можно ли из строки $S1$ путем перестановки ее литер получить строку $S2$.

6. Ф а й л ы

6.0. Формирование типизированного файла.

В задачах 6.1 - 6.10 рассматривается обработка типизированных файлов, элементами которых являются целые числа, вещественные числа, строки или записи. При этом предполагается, что обрабатываемые файлы уже существуют, т.е. они были предварительно сформированы в другой программе.

Формирование типизированного файла можно произвести или путем последовательного ввода его элементов с клавиатуры, или путем ввода из существующего текстового файла. С точки зрения удобства отладки программы и оперативности внесения изменений в типизированный файл второй вариант более предпочтителен.

Ниже приведены программы формирования типизированных файлов, элементами которых соответственно являются целые числа и записи.

```
Program FileInt;
Var  x : integer;
     F : file of integer;
     Ft : text;
Begin
  Assign(Ft,'F1.dat');
  Assign(F,'F2.dat');
  Reset(Ft); Rewrite(F);
  While not SeekEof(Ft) do
    Begin
      Read(Ft,x); Write(F,x);
    End;
  Close(Ft); Close(F);
End.
```

```
Program FileRec;
Type  RecType = record
          Ind : boolean;
          k : integer;
          r : real;
       end;
Var  b : byte;   { вводится 0 или 1 – машинный эквивалент }
     x : RecType;              { значений false или true }
     F : file of RecType;
     Ft : text;
Begin
  Assign(Ft,'D:\Example\f1.dat');
  Assign(F,'D:\Example\f.dat');
  Reset(Ft); Rewrite(F);
  While not SeekEof(Ft) do
    Begin
      Read(Ft,b,x.k,x.r);
      x.Ind:=boolean(b);  { преобразование числ.значения b }
      Write(F,x);         { в булевское значение x.Ind     }
    End;
  Close(Ft); Close(F);
End.
```

6.1. Компонентами типизированного файла являются целые числа, упорядоченные по возрастанию. Включить в состав файла произвольное число b, не нарушая при этом упорядоченности компонент файла.

Практически любая задача может быть реализована в программе различными способами. По отношению к обработке типизированных файлов, при которой выполняется добавление или удаление его элементов, существуют два основных способа:

- передвижение по файлу с помощью процедуры *Seek* и соответствующее перемещение его элементов; при этом буферные файлы не используются;

- перепись в буферный файл необходимых элементов исходного файла и переименование буферного файла после окончания его формирования.

С точки зрения затрат машинного времени оба варианта примерно одинаковы. Дополнительные затраты дисковой памяти во втором варианте не являются решающими, поскольку ее объем значительно превышает объем оперативной памяти компьютера. В то же время читабельность и возможность понимания программы, реализованной по второму способу, заметно выше. В связи с этим рассматриваемые ниже задачи реализуются, как правило, с использованием буферных файлов.

В принципе можно указать еще на один способ обработки типизированного файла: чтение из файла, формирование массива, обработка массива, запись в файл. Однако такой способ, как правило, неприменим по следующим причинам:

- размер массива, объявляемого в программе, всегда фиксированный, что делает проблематичным его соответствие размеру файла;

- размер файла может превышать 64 Кбайта, что делает невозможным его размещение в массиве.

Комментарии к программе Task601.

1. Если исходный файл *F* пустой, то в него записывается значение переменной *b* и файл *F* закрывается; в противном случае выполняются следующие части программы.

2. Если файл *F* не пустой, то формирование новых элементов производится в буферном файле *F1.*

3. Для индикации однократной записи в файл переменной *b* используется булевская переменная *Cond: true* - переменная *b* еще не записана в файл, *false* - запись этой переменной уже произведена.

4. Поскольку файл *F* не пустой, то в нем имеется по крайней мере один элемент. Поэтому до начала цикла обработки в переменную *x* читается значение первого компонента файла *F.*

5. В цикле ***While*** выполняются следующие действия:

- если $b \le x$ и *Cond = true*, значение *b* записывается в буферный файл *F1*; при этом переменной *Cond* присваивается значение *false*, что исключает в дальнейшем повторную запись *b* в *F1;*

- в *F1* записывается значение *x*;

- из файла *F* читается очередной компонент *x* .

Если в файле *F* изначально находился лишь один элемент, то цикл ***While*** ни разу не выполняется.

6. Если после окончания цикла имеет место *Cond = true*, то это означает, что переменная *b* еще не записана в файл *F1.* Тогда сравниваются значения *b* и последнего компонента *x,* прочитанного из файла *F*, и, в зависимости от результата сравнения, производится запись в *F1* пары *b* и *x* или пары *x* и *b*. Если же *Cond = false*, то в файл *F1* записывается лишь последний компонент *x.*

7. В программе имеются две файловые переменные: *F* и *F1*. Первой из них соответствует дисковый файл *F.dat*, второй - файл *F1.dat*. Процедура *Erase(F)* уничтожает на диске файл *F.dat*. При отработке процедуры *Rename(F1,'F.dat')* файл *F1.dat,* связанный процедурой *Assign* с файловой переменной *F1*, получает новое имя *F.dat*. Поскольку действия процедур *Assign* не изменились, то файловой переменной *F* по-прежнему соответствует дисковый файл *F.dat* (но уже с измененными в соответствии с условием задачи компонентами), а переменной *F1* – уже не существующий файл *F1.dat.*

Рабочим файлом в программе является *F.dat*, он по-прежнему под этим именем находится на диске и может дополнительно обрабатываться в этой или в другой программе. Файл *F1.dat* требовался лишь на время обработки файла *F.dat:* он был создан программой на диске, а затем фактически удален из диска (вернее, переименован в файл *F.dat* после уничтожения исходного файла с этим именем).

```
Program Task601;
Type  FileInt = file of integer;
Var  x,b : integer;
     Cond : boolean;
     F,F1 : FileInt;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Read(b); Reset(F);
  If FileSize(F)=0 then
    Begin
      Write(F,b); Close(F);
    End
  Else
    Begin
      Rewrite(F1); Read(F,x);
      Cond:=true;
      While not eof(F) do
        Begin
          If Cond and (b<=x) then
            Begin
              Write(F1,b); Cond:=false
            End;
          Write(F1,x); Read(F,x);
        End;
      If Cond then
        If b<=x then
          Begin
            Write(F1,b); Write(F1,x);
          End
        Else
          Begin
            Write(F1,x); Write(F1,b);
          End
      Else
        Write(F1,x);
      Close(F); Close(F1);
      Erase(F); Rename(F1,'F.dat');
    End;
End.
```

6.2. Компонентами типизированного файла являются вещественные числа. Требуется обменять местами первый и второй отрицательные элементы. Учесть, что в частном случае в файле может быть менее двух таких элементов.

Переменная *p* определяет в программе Task602 порядковый номер компонента файла, нумерация которых, как известно, производится в последовательности 0, 1, 2, ...,*FileSize(F)-1*. Переменные *Pos1Neg* и *Pos2Neg* используются для запоминания позиций первого и второго отрицательных элементов файла *F*. Этим переменным присваивается начальное значение -1, находящееся за пределами возможных номеров компонентов файла. Если после окончания

цикла просмотра файла *F* будет обнаружено, что переменная *Pos2Neg* не превышает нулевое значение, то это означает, что в файле *F* имеется менее двух отрицательных элементов. При *Pos2Neg > 0* производится обмен двух элементов файла. Для этого указатель файла перемещается на позицию *Pos1Neg* и в файл записывается значение второго компонента *Neg2*; после этого в позиции *Pos2Neg* выполняется запись значения *Neg1*. При этом, естественно, "старые" значения компонентов с номерами *Pos1Neg* и *Pos2Neg* будут заменены новыми значениями, равными соответственно *Neg2* и *Neg1*.

```
Program Task602;
Label  10;
Type  FileReal = file of real;
Var  p,Pos1Neg,Pos2Neg : longint;
     x,Neg1,Neg2 : real;
     F : FileReal;
Begin
  Assign(F,'F.dat'); Reset(F);
  Pos1Neg:=-1; Pos2Neg:=-1; p:=0;
  While not eof(F) do
    Begin
      Read(F,x);
      If x<0 then
        Begin
          If Pos1Neg=-1 then
            Begin
              Pos1Neg:=p; Neg1:=x
            End
          Else
            Begin
              Pos2Neg:=p; Neg2:=x; Goto 10
            End;
        End;
      Inc(p);
    End;
  10:
  If (Pos2Neg>0) and (Neg1<>Neg2) then
    Begin
      Seek(F,Pos1Neg); Write(F,Neg2);
      Seek(F,Pos2Neg); Write(F,Neg1);
    End;
  Close(F);
End.
```

6.3. Компонентами типизированного файла являются целые числа. Требуется обменять местами минимальный элемент с последним нулевым элементом файла.

Если файл не пустой, то в нем обязательно имеется минимальный элемент (если все элементы файла одинаковы, то в качестве минимального принимается первый элемент). Если в файле нет нулевого элемента, то этот элемент не с чем обменивать; если минимальный элемент равен нулю, то его незачем обменивать; эти обстоятельства учтены в программе Task603.

В отличие от предыдущей программы, для определения позиции искомого элемента вместо счетчика компонент *p* используется предописанная функция *FilePos(F),* выходным значением которой является текущее положение указателя файла. При этом учитывается, что после выполнения процедур *Read* или *Write* указатель файла сдвигается на следующий ком-

понент; поэтому переменным *PosMin* и *PosZero* присваивается значение *FilePos(F)-1,* а не *FilePos(F).*

В программе Task603 следует обратить внимание еще на одну деталь. После установки указателя файла в позицию *PosMin* производится запись в файл *F* переменной *x,* которой предварительно было присвоено нулевое значение. В этом месте неприменима процедура *Write*(*F*,0), так как в типизированный файл нельзя записывать константу. Это связано с тем, что при обмене данными между оперативной памятью и типизированным файлом преобразование формы представления информации не производится. Другими словами, если компонентом файла по объявлению в разделе ***Var*** является переменная типа *integer*, то в этот файл можно записывать только значения переменных типа *integer*. В то же время представление константы 0 в Паскаль-программе многозначно: это может быть число любого вещественного или целочисленного типа.

```
Program Task603;
Type  FileInt = file of integer;
Var  PosMin,PosZero : longint;
     x,xmin : integer;
     F : FileInt;
Begin
  Assign(F,'F.dat'); Reset(F);
  PosZero:=0; PosMin:=0;
  xmin:=MaxInt;
  While not eof(F) do
    Begin
      Read(F,x);
      If x<xmin then
        Begin
          xmin:=x; PosMin:=FilePos(F)-1;
        End;
      If x=0 then
        PosZero:=FilePos(F)-1;
    End;
  If (PosZero>0) and (xmin<>0) then
    Begin
      Seek(F,PosZero); Write(F,xmin);
      x:=0;
      Seek(F,PosMin); Write(F,x);
    End;
  Close(F);
End.
```

6.4. Компонент типизированного файла - это запись, одним из элементов которой является переменная *Ind* типа *boolean*, предназначенная для индикации фиктивно удаленных компонентов (в этом случае переменной *Ind* присваивается значение *true*). Сжать исходный файл, удалив из него все компоненты, содержащие *Ind = true.*

Компонентом файла *F* является запись, состоящая в данном случае из трех элементов. В отличие от текстового файла, при обращении к типизированному файлу производится передача всей записи, а не отдельных ее элементов.

В программе Task604 в буферный файл *F1* переписываются из файла *F* лишь те записи, в которых составляющая *Ind* имеет значение *false.* В остальном обработка файла выполняется так же, как и в программе Task601.

```
Program Task604;
```

```
Type  RecType = record
        Ind : boolean;
        k : integer;
        r : real
      end;
    FileRec = file of RecType;
Var  x : RecType;
     F,F1 : FileRec;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Reset(F);
  If FileSize(F)>0 then
    Begin
      Rewrite(F1);
      While not eof(F) do
        Begin
          Read(F,x);
          If not x.Ind then
            Write(F1,x);
        End;
      Close(F); Close(F1);
      Erase(F); Rename(F1,'F.dat');
    End
  Else
    Close(F);
End.
```

6.5. Компонентами типизированного файла являются целые числа, причем среди них может быть лишь один нулевой элемент. Обменять местами части файла, расположенные до и после нулевого элемента. Учесть, что в частном случае нулевой элемент может быть первым или последним в файле.

В программе Task605 применяются два буферных файла *F1* и *F2*.

Для определения позиции нулевого элемента используется переменная *p*, являющаяся фактически счетчиком компонент файла *F*; для запоминания этой позиции - переменная *PosZero*. Переменная *Cond* свидетельствует о наличии в файле *F* нулевого элемента.

В фрагменте 1 определяется положение нулевого элемента. Если он обнаружен (*Cond=true*), то в дальнейшем из файла *F* в файл *F1* переписываются все компоненты, расположенные до нулевого элемента (фрагмент 2), а в файл *F2* - компоненты, расположенные после этого элемента (фрагмент 3). Если нулевой элемент является первым в файле *F*, то файл *F1* будет пустым; если он последний, то пустым будет файл *F2*. После этого содержимое файла *F1* переписывается в файл *F* (фрагмент 3), записывается нулевое значение, а затем в файл *F* переписывается содержимое файла *F2* (фрагмент 4). Файлы *F1* и *F2* уничтожаются.

```
Program Task605;
Type  FileInt = file of integer;
Var  PosZero : longint;
     x : integer;
     F,F1,F2 : FileInt;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Assign(F2,'F2.dat'); Reset(F);
  PosZero:=-1;
  While not eof(F) do                { Фрагмент 1 }
```

```
      Begin
        Read(F,x);
        If x=0 then
          Begin
            PosZero:=File(F)-1; Break
          End;
      End;
    If PosZero>=0 then
      Begin
        Rewrite(F1); Rewrite(F2);
        Seek(F,0); p:=0;
        While p<PosZero do              { Фрагмент 2 }
          Begin
            Read(F,x); Write(F1,x); Inc(p);
          End;
        Seek(F,PosZero+1);
        While not eof(F) do             { Фрагмент 3 }
          Begin
            Read(F,x); Write(F2,x);
          End;
        Close(F); Rewrite(F); Seek(F2,0);
        While not eof(F2) do            { Фрагмент 4 }
          Begin
            Read(F2,x); Write(F,x);
          End;
        x:=0; Write(F,x); Seek(F1,0);
        While not eof(F1) do            { Фрагмент 5 }
          Begin
            Read(F1,x); Write(F,x);
          End;
        Close(F1); Close(F2);
        Erase(F1); Erase(F2);
      End;
    Close(F);
End.
```

Примечание. В программе Task605 можно ограничиться одним буферным файлом *F1*. Порядок действий:

- определение положения нулевого элемента *PosZero*;
- установка указателя файла *F* в позицию *PosZero+1*;
- перепись в файл *F1* всех компонент файла *F*, начиная с установленной позиции (в цикле "***While not** eof(F) **do**...");
- запись в *F1* нулевого значения;
- установка указателя файла *F* в нулевую позицию;
- запись в файл *F1* всех компонент файла *F* от начальной до компоненты с номером *PosZero-1* (в цикле "***For** i:=0 **to** PosZero-1 **do**...");
- закрытие файлов *F* и *F1*;
- уничтожение дискового файла, связанного с файлом *F*;
- переименование дискового файла, связанного с файлом *F1*.

6.6. Из типизированного файла, компонентами которого являются целые числа, удалить все отрицательные элементы, кроме первого такого элемента.

Индикатором первого отрицательного элемента в файле *F* является переменная *Cond*. Если *Cond = true*, то отрицательный элемент записывается в буферный файл *F1*, в противном случае из файла *F* переписываются только неотрицательные элементы.

```
Program Task606;
Type  FileInt = file of integer;
Var  x : integer;
     Cond : boolean;
     F,F1 : FileInt;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Reset(F); Rewrite(F1);
  Cond:=true;
  While not eof(F) do
    Begin
      Read(F,x);
      If (x<0) and Cond then
        Begin
          Write(F1,x); Cond:=false
        End
      Else
        If x>=0 then
          Write(F1,x);
    End;
  Close(F); Close(F1);
  Erase(F); Rename(F1,'F.dat');
End.
```

6.7. Из типизированного файла с вещественными компонентами удалить элемент, в минимальной степени отличающийся от среднего арифметического значения элементов, содержащихся в файле.

Здесь речь идет об удалении элемента, для которого модуль разности между его значением и средним арифметическим значением элементов файла минимален. В программе Task607, если файл *F* не пустой, определяется среднее арифметическое значение *S*, а затем номер *PosMin* искомого элемента (при нумерации 1,2,3,...). После этого в буферный файл *F1* из файла *F* переписываются все элементы, кроме элемента с номером *PosMin*.

```
Program Task607;
Type  FileReal = file of real;
Var  n,PosMin : word;
     x,xmin,S : real;
     F,F1 : FileReal;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Reset(F);
  If FileSize(F)>0 then
    Begin
      Rewrite(F1); S:=0; n:=0;
      While not eof(F) do
        Begin
          Read(F,x); Inc(n); S:=S+x;
        End;
      S:=S/n;
      Seek(F,0); n:=1;
```

```
      Read(F,x); xmin:=abs(S-x);
      While not eof(F) do
        Begin
          Read(F,x); Inc(n);
          If abs(S-x)<xmin then
            Begin
              xmin:=abs(S-x); PosMin:=n;
            End;
        End;
      Seek(F,0); n:=0;
      While not eof(F) do
        Begin
          Read(F,x); Inc(n);
          If n<>PosMin then
            Write(F1,x);
        End;
      Close(F); Close(F1);
      Erase(F); Rename(F1,'F.dat');
    End
  Else
    Close(F);
  Writeln('Среднее арифметическое  S = ',S:6:1);
End.
```

6.8. Компонентами типизированного файла являются целые числа, упорядоченные по убыванию. Заданы также две целые переменные *k1* и *k2*, причем $k1 \le k2$. Удалить из файла все компоненты, которые по своему значению расположены в диапазоне *k1* .. *k2*, после чего записать в файл значение выражения *(k1+k2)/2*, не нарушая при этом упорядоченности элементов файла. Учесть, что в частном случае файл может быть пустым.

В программе Task608, если файл *F* не пустой, в буферный файл переписываются все компоненты, расположенные по значению вне диапазона [*k1,k2*] . После этого при обратной переписи из *F1* в *F* по аналогии с программой Task601 в файл *F* включается значение *k = (k1 + k2)* ***div*** *2*. Если файл F пустой, то производится лишь запись в этот файл значения *k*.

```
Program Task608;
Type  FileInt = file of integer;
Var  k,k1,k2,x : integer;
     Cond : boolean;
     F,F1 : FileInt;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Read(k1,k2); Reset(F);
  k:=(k1+k2) div 2;
  If FileSize(F)>0 then
    Begin
      Rewrite(F1);
      While not eof(F) do
        Begin
          Read(F,x);
          If (x<k1) or (x>k2) then
            Write(F1,x);
        End;
      Cond:=true;
      Seek(F1,0); Rewrite(F);
```

```
      While not eof(F1) do
        Begin
          Read(F1,x);
          If (x<=k) and Cond then
            Begin
              Write(F,k); Cond:=false
            End;
          Write(F,x);
        End;
      If Cond then
        Write(F,k);
      Close(F1); Erase(F1);
    End
  Else
    Write(F,k);
  Close(F);
End.
```

6.9. Компонентами типизированного файла являются записи. Каждая запись содержит координаты центра и радиус окружности. Удалить из файла записи, которые определяют окружности, полностью или частично выходящие за пределы первого квадранта.

В описании типа записи *CircleType* компоненты *cx* и *cy* - это абсцисса и ордината центра окружности, *R* - ее радиус. Окружность полностью находится в первом квадранте, если *cx - R > 0* и *cy - R > 0*. В программе Task609 этот признак используется для принятия решения о том, какие компоненты переписывать из файла *F* в файл *F1*.

```
Program Task609;
Type  CircleType = record
        cx,cy,R : real;
      end;
      FileCircles = file of CircleType;
Var  Circle : CircleType;
     Cond : boolean;
     F,F1 : FileCircles;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Reset(F);
  If FileSize(F)>0 then
    Begin
      Rewrite(F1);
      While not eof(F) do
        Begin
          Read(F,Circle);
          Cond:=((Circle.cx-Circle.R)>0) and
                ((Circle.cy-Circle.R)>0);
          If Cond then
            Write(F1,Circle);
        End;
      Close(F); Close(F1);
      Erase(F); Rename(F1,'F.dat');
    End
  Else
    Close(F);
End.
```

6.10. Компонентом типизированного файла является строка с объявленной длиной 80 символов. В частном случае строка может быть пустой. Удалить из файла все пустые строки, кроме последней такой строки.

Способ организации программы Task610 в основном аналогичен программе Task606, но вместо переменной *Cond* для индикации наличия удаляемого элемента непосредственно используется его стандартный порядковый номер *k* : значение *k* = -1 свидетельствует об отсутствии такого элемента.

```
Program Task610;
Type  string80 = string[80];
Var  k : longint;
     St : string80;
     F,F1 : file of string80;
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Reset(F);
  If FileSize(F)>0 then
    Begin
      k:=-1;                        { индикация положения }
      While not eof(F) do           { последней пустой строки }
        Begin
          Read(F,St);
          If length(St)=0 then
            k:=FilePos(F);
        End;
      If k>=0 then
        Begin
          Rewrite(F1); Seek(F,0);
          While not eof(F) do
            Begin
              Read(F,St);
              If length(St)>0 then
                Write(F1,St)
              Else
                If k=FilePos(F) then
                  Write(F1,St);
            End;
          Close(F); Close(F1);
          Erase(F); Rename(F1,'F.dat');
        End
      Else
        Close(F);
    End
  Else
    Close(F);
End.
```

Примечание. Переменной *k* фактически присвоен номер компонента, расположенного после последней пустой строки. Это учитывается в операторе ***If*** при принятии решения о записи этой строки в файл *F1*.

6.11. Задан символьный файл (***file of*** *char*), элементами которого являются цифры и пробелы. Числа, формируемые последовательностью цифр, могут быть целые и вещественные. В последнем случае целая и дробная части числа разделяются точкой. Перед первой цифрой числа может стоять знак "+" или "-". В том случае, когда дробная часть числа имеет свыше двух разрядов, необходимо округлить ее до двух разрядов. Если перед числом стоит знак "+", то его удалить. Разделителем между числами должен быть только один пробел.

Из символьного файла возможно лишь последовательное чтение содержащихся в нем символов (или запись символов).

Чтение файла *F* производится в основной части программы Task611. Формирование символов числа выполняется в строке *S*, которой в стартовой части программы присвоено пустое значение. Переменная *Cond* управляет формированием числа; этой переменной первоначально присвоено значение ***true.***

Пробелы, расположенные перед первым символом числа, игнорируются. Формирование числа начинается при поступлении первого значащего символа. При этом переменной *Cond* присваивается значение *false*. Если после значащих символов будет прочитан пробел, то это означает, что формирование числа закончено. Тогда переменной *Cond* присваивается значение *true,* что приводит к игнорированию пробелов, расположенных перед первым символом следующего числа. Дальнейшая обработка строки *S* производится в процедуре *Rounding.*

В процедуре *Rounding* вначале проверяется, имеется ли точка в составе числа. Положение точки определяет переменная *k*. Если точка обнаружена, то проверяется дробная часть числа. Если в этой части свыше двух разрядов, то начинается работа по ее округлению.

Предположим, что число имеет значение 999.998. Тогда после округления оно должно быть записано в виде 1000.00. Чтобы обеспечить возможность записи в число дополнительной цифры "1", перед первой цифрой числа с учетом наличия в нем знака вставляется цифра "0".

В процессе округления используются два набора десятичных цифр: строка *DigString* и массив (типизированная константа) *DigAr.* Строка *DigString* используется для определения значения очередной цифры в числе *S* (с помощью функции *Pos*), массив *DigAr* - для вставки старшей цифры при округлении (строку-константу нельзя индексировать).

Округление производится, если третья цифра в дробной части числа больше, чем 4. Замещение цифр выполняется в цикле справа налево под управлением булевской переменной *CondDig.* При этом учитывается возможность перехода в целую часть числа (пропуск разделяющей точки).

Если в дробной части числа свыше двух цифр, то независимо от того, было или не было произведено округление, избыточные цифры удаляются процедурой *Delete.* После этого из состава числа удаляются знак "+" и незначащие нули, если они имеются. Символы числа записываются в файл *F1,* после них записывается разделяющий пробел. Строке S присваивается пустое значение для приема очередного числа.

Если в символьном файле *F* после последней цифры последнего числа нет пробела, то обращение к процедуре *Rounding* в теле цикла ***While*** не будет произведено. В этом случае после окончания цикла проверяется длина строки *S* и, если она не нулевая, то производится обращение к процедуре *Rounding* для округления и записи в файл *F1* последнего числа.

```
Program Task611;
Const  DigString = '0123456789';        { строка цифр }
       DigAr : array[1..10] of char =   { массив цифр }
             ('0','1','2','3','4','5','6','7','8','9');
Type   FileChar = file of char;          { тип файла }
Var    Cond : boolean;
       ch : char;
       S : string;
```

```
     F,F1 : FileChar;  { исходный и буферный файлы }
{ ------------------------------------------------ }
Procedure Rounding;
{ Округление и запись числа в файл }
Var   i,k,k1 : byte;
      CondDig : boolean;
Begin
  k:=Pos('.',S);                  { индикация точки в числе }
  If k>0 then
    If length(S)-k>2 then
      Begin
        If (S[1]='+') or (S[1]='-') then
          Insert('0',S,2)           { установка нуля перед }
        Else                        { первой цифрой числа  }
          Insert('0',S,1);
        Inc(k);
        If S[k+3]>'4' then
          Begin
            CondDig:=true; i:=k+2;   { округление дробной }
            While CondDig do         { части числа        }
              Begin
                k1:=Pos(S[i],DigString);
                If k1=0 then
                  Dec(i)
                Else
                  If k1<10 then
                    Begin
                      S[i]:=DigAr[k1+1]; CondDig:=false
                    End
                  Else
                    Begin
                      S[i]:='0'; Dec(i);
                    End;
              End;
          End;
        Delete(S,k+3,length(S)-k-2); { удаление избыточных }
      End;                           { разрядов дробной части }
  If (S[1]='+') or (S[1]='-') then { удаление нуля после }
    If S[2]='0' then               { знака числа }
      Delete(S,2,1);
  If (S[1]='+') or (S[1]='0') then { удаление знака "+" или }
    Delete(S,1,1);                 { незначащего нуля }
  For i:=1 to length(S) do         { запись числа в сим- }
    Begin                          { вольный файл F1 }
      ch:=S[i];
      Write(F1,ch);
    End;
  ch:=' '; Write(F1,ch);       { запись разделяющего пробела }
  S:='';                       { подготовка строки S к приему }
End { Rounding };              { нового числа }
{ ------------------------------------------------ }
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Reset(F); Rewrite(F1);
  If FileSize(F)>0 then
    Begin
      Cond:=true; S:='';
```

```
      While not eof(F) do    { формирование числа в строке S  }
        Begin                { при чтении символов из файла F }
          Read(F,ch);
          If ch<>' ' then
            Begin
              S:=S+ch; Cond:=false
            End
          Else
            If not Cond then
              Begin
                Cond:=true;    { округление очередного числа }
                Rounding;      { и запись его в файл F1 }
              End;
        End;
      If length(S)>0 then      { округление последнего числа }
        Rounding;              { и запись его в файл F1 }
    End;
  Close(F); Close(F1);
  Erase(F); Rename(F1,'F.dat');
End.
```

6.12. В текстовом файле записаны ненулевые элементы одномерного целочисленного массива. Форма записи элемента имеет вид: (*k*)*m*, где *k* - индекс элемента, *m* - его значение. Предполагается, что индексация элементов массива начинается с 1. Числа разделяются между собой одним или несколькими пробелами. Перед значением *m* также могут быть пробелы. Последний элемент массива всегда содержится в файле вне зависимости от значения этого элемента. Сформировать новый текстовый файл, в который в явном виде записать все элементы массива. Длина строки файла не должна превышать 80 байт.

В основной части программы Task612 последовательно читается из файла *F* строка *S* и направляется на обработку в процедуру *MakeString.* Переменной *i,* определяющей индекс очередного числа, записываемого в выходной файл *F1,* присваивается начальное значение 0. В строку *S1*, в которой накапливаются преобразованные числа, на старте программы засылается пустое значение.

В процедуре *MakeString* выделяются две части очередного числа: индекс *k* и значение *m*, записываемое в строку *Sb.* Для этого используются функции *BegBracket, EndBracket, Space* и *NotSpace*. Методика выделения числа аналогична методике выделения слов, использованной в задачах по обработке строк.

После выделения составных частей очередного числа текущий индекс *i* увеличивается на 1. Если $i < k$, то в строку *S1* записываются *k-i* нулевых значений. После этого в нее записывается значение числа из строки *Sb*. После каждого числа добавляется разделяющий пробел. Перед добавлением в строку *S1* очередного числа проверяется, достаточно ли в ней места для размещения этого числа. Если возможности строки *S1* исчерпаны, то ее содержимое записывается в файл *F1*, а в начальную часть строки *S1* включается это число.

После окончания цикла чтения строк из файла *F* проверяется длина строки *S1.* Если эта длина ненулевая, то содержащиеся в строке числа записываются в файл *F1.*

```
Program Task612;
Var  i : word;            { индекс очередного числа }
     S,S1 : string[80];   { исходная и преобразованная строки }
     F,F1 : text;         { исходный и преобразованный файлы }
{ ------------------------------------------------ }
Function BegBracket(k:byte):byte;
```

```
{ Поиск ближайшей открывающей скобки }
Var  i : byte;
Begin
  BegBracket:=0;
  For i:=k to length(S) do
    If S[i]='(' then
      Begin
        BegBracket:=i; Exit
      End;
End { BegBracket };
{ ------------------------------------------------- }
Function EndBracket(k:byte):byte;
{ Поиск ближайшей закрывающей скобки }
Var  i : byte;
Begin
  EndBracket:=0;
  For i:=k to length(S) do
    If S[i]=')' then
      Begin
        EndBracket:=i; Exit
      End;
End { EndBracket };
{ ------------------------------------------------- }
Procedure MakeString;
{ Формирование преобразованной строки и запись ее в файл }
Var    k1,              { позиция открывающей скобки }
       k2,              { позиция закрывающей скобки }
       k3,              { позиция непробела }
       k4,              { позиция пробела }
       lb : byte;       { длина строки Sb }
       Code : integer;  { параметр процедуры Val }
       j,               { параметр цикла }
       k : word;        { индекс прочитанного числа }
       Cond : boolean;  { управляющая переменная }
       Sb : string[80];{ строка для выделения числа }
Begin
  Cond:=true; k4:=0;
  While Cond do
    Begin
      k1:=BegBracket(k4+1);
      If k1=0 then
        Cond:=false
      Else
        Begin
          k2:=EndBracket(k1+1); k3:=NotSpace(k2+1);
          k4:=Space(k3+1);
          If k4=0 then
            Begin
              k4:=length(S)+1; Cond:=false;
            End;
          Sb:=Copy(S,k1+1,k2-k1-1); Val(Sb,k,Code);
          Sb:=Copy(S,k3,k4-k3); lb:=length(Sb);
          Inc(i);
          If k>i then
            For j:=1 to k-i do
              If length(S1)+2<80 then
                S1:=S1+'0 '
```

```
            Else
              Begin
                Writeln(F1,S1); S1:='0 ';
              End;
          i:=k;
          If length(S1)+lb<80 then
            S1:=S1+Sb+' '
          Else
            Begin
              Writeln(F1,S1); S1:=Sb+' ';
            End;
        End;
  End;
End { MakeString };
{ ------------------------------------------------ }
Begin
  Assign(F,'F.dat'); Assign(F1,'F1.dat');
  Reset(F); Rewrite(F1);
  S1:=''; i:=0;
  While not eof(F) do
    Begin
      Readln(F,S); MakeString;
    End;
  If length(S1)>0 then
    Writeln(F1,S1);
  Close(F); Close(F1);
End.
```

Задания для самостоятельной работы

6.13. Компонентами типизированного файла являются целые числа. Исключить из файла повторные вхождения одного и того же числа.

6.14. В каждом из двух типизированных файлов *F1* и *F2* компоненты, являющиеся целыми числами, упорядочены по возрастанию. Сформировать новый файл *F3* из содержимого файлов *F1* и *F2*, упорядочив его компоненты по убыванию. В файле *F3* не должно быть повторяющихся чисел. Группировку компонент файла не производить.

6.15. В типизированный файл, компонентами которого являются вещественные числа, после каждого отрицательного элемента вставить нулевой элемент.

6.16. В типизированный файл, компонентами которого являются целые числа, перед каждым нечетным элементом вставить удвоенное значение этого элемента.

6.17. Задан типизированный файл с целочисленными компонентами и произвольное целое число *b*. Если для *i*-ой компоненты файла имеет место $x_i < b < x_{i+1}$, то вставить значение *b* между этими компонентами. Учесть, что таких вставок может быть несколько (или ни одной).

6.18. В типизированном файле с вещественными компонентами обменять местами максимальный элемент с первым нулевым элементом этого файла, если в нем имеется такой элемент. Учесть, что в частном случае максимальный элемент может быть равен нулю.

6.19. Компонентами типизированного файла являются записи, определяющие координаты точек на плоскости. Удалить из файла точку с максимальным расстоянием от начала координат.

6.20. Компонентом файла является запись из двух целых чисел *m* и *n,* являющихся числителем и знаменателем дроби $p = m/n$. При $m = 0$ и $n = 0$ принимается $p = 0$. Значения *p* - это коэффициенты полинома, начиная со старшего и заканчивая свободным членом. Вычислить

по схеме Горнера значение полинома при заданном с клавиатуры значении аргумента. Просмотр файла допускается осуществлять только один раз.

6.21. Разреженная вещественная матрица записана по строкам в типизированном файле. Компонент файла - это запись, содержащая ненулевой элемент матрицы: два индекса и значение элемента. В первом компоненте вместо индексов записаны размеры матрицы (количество строк и столбцов). Сформировать текстовый файл, в первой строке которого записать размеры матрицы, а в остальных - все ее элементы, как нулевые, так и ненулевые. При записи чисел использовать формат 8:2. Элементы матрицы разделять в строке файла одним пробелом. Каждую строку матрицы начинать с новой строки файла.

Длина строки текстового файла не должна превышать 80 символов. Если в одной строке файла не размещаются все элементы строки матрицы, запись их продолжать на следующей строке файла.

Матрицу, читаемую из типизированного файла, не формировать в оперативной памяти.

6.22. То же, но из текстового файла (в нем полная матрица) в типизированный.

6.23. Дан текстовый файл *F1*. Переписать в обратном порядке его строки в файл *F2*. Содержимое файла F1 в оперативную память не копировать (в виде массива, стека, очереди и т.п.).

6.24. Заданы два текстовых файла *F1* и *F2*, длина строк которых не превышает 80 символов. Сравнить два файла и определить количество несовпадающих в них строк. Если две строки отличаются лишь количеством пробелов, такие строки считать одинаковыми. Если один из файлов короче другого, то условно дополнить более короткий файл пустыми строками.

6.25. Заданы два текстовых файла с внешними именами *'F1.pas'* и *'F2.pas'*. Если в какой-либо строке файла *F1.pas* имеется фраза

{$I F2.pas} (*это запись директивы Include*),

то включить в это место файла *F1.pas* текст файла *F2.pas*. Учесть, что между любыми элементами директивы *Include* может быть произвольное количество пробелов. Если перед директивой *Include* или после нее имеются какие-либо слова, то эти слова должны быть записаны в отдельных строках файла *F1.pas*. В файле *F1.pas* могут быть комментарии, ограниченные, как и директива *Include*, фигурными скобками.

6.26. Созданный ранее текстовый файл объявлен в программе как символьный (***file of*** *char*). Слова в строках файла разделяются пробелами. Отпечатать первое слово третьей строки файла, если оно существует. Признаком конца строки являются два последовательных символа #13 и #10, признаком конца файла - символ #26.

6.27. Задан символьный файл (***file of*** *char*), элементами которого являются цифры и пробелы, разделяющие между собой числа. Удалить из файла нечетные по значению числа.

6.28. В символьном файле записаны слова, разделенные пробелами. Удалить из файла слова, состоящие менее чем из трех букв, и лишние пробелы.

6.29. В символьном файле записан произвольный текст длиной n слов ($n \geq 5000$). Слова разделены пробелами и знаками препинания. Считая, что длина слова не превышает k букв ($k \leq 16$), получить m ($m \leq 100$) наиболее часто встречающихся слов, число их повторений и частоту повторений в процентах. Список слов отпечатать в порядке уменьшения значения p, а при одинаковом значении p - в алфавитном порядке.

6.30. Даны два символьные файла *F1* и *F2*. Файл *F1* содержит произвольный текст, слова которого разделены пробелами и знаками препинания. В файле *F2* записаны m пар слов ($m \leq 10$), которые разделены запятыми (длина слова не превышает n символов, $n \leq 16$). Первое слово пары считается заменяемым, второе - заменяющим. Найти в файле *F1* все заменяемые слова и заменить их на соответствующие заменяющие слова. Результат разместить в файле *F3*.

7. Линейные списки

7.0. Общие замечания.

В качестве линейных списков рассматриваются стеки, очереди и деки. Как правило, в задачах считается, что информационная часть в компоненте списка - это одно число (целое или вещественное).

В приведенных ниже программах предполагается, что стек или очередь уже сформированы тем или иным способом, например, путем ввода из текстового файла. Поскольку файл в частном случае может быть пустым, то и программа обработки должна правильно реагировать на наличие пустого списка.

7.1. Информационная часть элемента очереди - вещественное число. Вставить после каждого отрицательного элемента очереди нулевой элемент.

В программе Task701 последовательно просматриваются элементы непустой очереди и, если для очередного элемента *Run^.Inf* < 0, то после него вставляется новый элемент *Elem* с информационной частью, равной нулю. Если новый элемент добавляется в конец очереди (*Run=R*), то ее правый указатель *R* переносится на новый элемент.

```
Program Task701;
Type PoinType = ^DynType;
     DynType = record
        Inf : real;       { информационная часть очереди }
        Next : PoinType; { указатель след.элемента очереди }
     end;
Var  L,R,                 { левый и правый указатели очереди }
     Run,                 { текущий указатель }
     Elem : PoinType;     { указатель нового элемента }
Begin
  Формирование очереди
  If L<>nil then
    Begin
      Run:=L;
      While Run<>nil do
        Begin
          If Run^.Inf<0 then
            Begin
              New(Elem);
              Elem^.Inf:=0; Elem^.Next:=Run^.Next;
              Run^.Next:=Elem;
              If Run=R then
                R:=Elem;
            End;
          Run:=Run^.Next;
        End;
    End;
  Печать очереди
End.
```

7.2. Информационная часть элемента стека - целое число. Вставить перед каждым нечетным элементом его удвоенное значение.

Чтобы вставить новый элемент перед произвольным элементом стека, необходимо знать адрес предыдущего элемента. Поскольку для первого элемента не существует предыдущего элемента, то в программе рассматриваются две ситуации: четность первого и четность любого другого элемента стека (в отличие от очереди последний элемент стека не требуется анализировать отдельно).

Если первый элемент стека нечетный, то перед ним вставляется новый элемент *Elem* с соответствующим переносом начального указателя *Beg*. Вне зависимости от этого начальное значение текущего указателя *Run* определяет адрес исходного первого элемента стека.

В цикле ***While*** последовательно проверяются элементы, адресуемые указателем *Run^.Next*. Следовательно, указатель *Run^.Next* определяет при этом адрес текущего элемента, а указатель *Run* - адрес предыдущего элемента. При каждом повторении цикла производится переход на следующий элемент стека (*Run:=Run^.Next*). Если же перед текущим элементом вставляется новый элемент, то дополнительно осуществляется такой же переход к новому элементу (в противном случае перед одним и тем же элементом стека многократно производилась бы вставка нового элемента, т.е. произошло бы зацикливание программы).

```
Program Task702;
Type PoinType = ^DynType;
     DynType = record
         Inf : integer;
        Next : PoinType;
     end;
Var  Beg,                  { указатель начала стека }
     Run,                  { текущий указатель }
     Elem : PoinType;      { указатель нового элемента }
Begin
  Формирование стека
  If Beg<>nil then
    Begin
      Run:=Beg;
      If odd(Beg^.Inf) then
        Begin
          New(Elem);
          Elem^.Inf:=2*Beg^.Inf; Elem^.Next:=Beg;
          Beg:=Elem;
        End;
      While Run^.Next<>nil do
        Begin
          If odd(Run^.Next^.Inf) then
            Begin
              New(Elem);
              Elem^.Inf:=2*Run^.Next^.Inf;
              Elem^.Next:=Run^.Next;
              Run^.Next:=Elem; Run:=Run^.Next;
            End;
          Run:=Run^.Next;
        End;
    End;
  Печать стека
End.
```

7.3. Из очереди, элементами которой являются вещественные числа, удалить все отрицательные элементы, кроме первого такого элемента.

В программе в качестве индикатора номера отрицательного элемента (первый или не первый) используется булевская переменная *Cond*: ее значение равно *true*, если еще не обнаружен отрицательный элемент, в противном случае этой переменной присваивается значение *false*.

Чтобы удалить из очереди любой элемент, кроме первого, необходимо знать адрес предыдущего элемента (для переноса связи от предыдущего к последующему элементу в обход удаляемого элемента). Для этого, как и в задаче 7.2, анализ производится по указателю *Run^.Next*.

Первый элемент очереди в соответствии с условием задачи не может быть удален, однако знак его информационной части определяет значение переменной *Cond*. В цикле ***While*** удаление отрицательного элемента производится, если *Cond = false*. Если *Cond = true*, то это означает, что в очереди впервые встретился такой элемент; в этом случае переменной *Cond* присваивается значение *false*.

При удалении элемента из очереди переход в цикле на следующий элемент (*Run:=Run^.Next*) не производится, так как на место изъятого элемента "надвигается" следующий после него элемент, который также необходимо анализировать на знак его информационной части.

Если из очереди удаляется последний элемент (*Run^.Next = R*), то указатель *R* переносится на предыдущий элемент, который становится теперь последним в очереди.

```
Program Task703;
Type PoinType = ^DynType;
     DynType = record
         Inf : real;
       Next : PoinType;
     end;
Var  L,R,               { левый и правый указатели очереди }
     Run,               { текущий указатель }
     Buf : PoinType;    { буферный указатель для удаления }
     Cond : boolean;    { элемента из очереди }
Begin
  Формирование очереди
  If L<>nil then
    Begin
      If L^.Inf<0 then
        Cond:=false
      Else
        Cond:=true;
      Run:=L;
      While Run^.Next<>nil do
        If Run^.Next^.Inf<0 then
          If not Cond then
            Begin
              Buf:=Run^.Next;
              If Run^.Next=R then
                R:=Run;
              Run^.Next:=Run^.Next^.Next;
              Dispose(Buf);
            End
          Else
            Begin
              Cond:=false; Run:=Run^.Next
            End
        Else
          Run:=Run^.Next;
```

```
    End;
  Печать очереди
End.
```

7.4. Из стека, элементами которого являются целые числа, удалить все нечетные элементы, кроме последнего такого элемента.

В отличие от массива движение по стеку или по очереди может быть только односторонним, в соответствии с направлением их указателей. Поэтому на первом этапе работы программы определяется положение последнего нечетного элемента, адрес которого запоминается в указателе *Last* (более точно, указатель *Last* определяет адрес элемента, предыдущего по отношению к последнему нечетному). Если после окончания просмотра стека значение переменной *Last* остается равным ***nil***, то это означает, что в стеке нет нечетных элементов; в этом случае второй этап работы программы не выполняется.

Удаление нечетных элементов в свою очередь производится в два этапа. Вначале из стека удаляется первый элемент до тех пор, пока на его место не станет четный или последний нечетный элемент. Затем при последовательном просмотре стека по указателю *Run^.Next* удаляются оставшиеся нечетные элементы до тех пор, пока данный указатель не станет равным значению *Last;* чтобы не осуществлять ненужный теперь просмотр остальных элементов, производится принудительный выход из цикла.

Примечание. В отличие от массива, сдвиг элементов стека при удалении какого-либо из его элементов не производится. Следовательно, значение указателя *Last* при удалении предыдущих элементов не изменяется.

```
Program Task704;
Label 10;
Type PoinType = ^DynType;
     DynType = record
         Inf : integer;
        Next : PoinType;
     end;
Var  Beg,                 { указатель начала стека }
     Run,                 { текущий указатель }
     Buf,                 { буферный указатель }
     Last : PoinType;     { указатель последнего элемента }
Begin
  Формирование стека
  If Beg<>nil then
    Begin
      Run:=Beg; Last:=nil;
      While Run<>nil do
        Begin
          If odd(Run^.Inf) then
            Last:=Run;
          Run:=Run^.Next;
        End;
      If Last<>nil then
        Begin
          While odd(Beg^.Inf) and (Beg<>Last) do
            Begin
              Buf:=Beg; Beg:=Beg^.Next;
              Dispose(Buf);
            End;
          Run:=Beg;
```

```
          While Run^.Next<>nil do
            If odd(Run^.Next^.Inf) then
              If Run^.Next<>Last then
                Begin
                  Buf:=Run^.Next;
                  Run^.Next:=Run^.Next^.Next;
                  Dispose(Buf);
                End
              Else
                Goto 10
            Else
              Run:=Run^.Next;
        End;
    End;
  10:
  Печать стека
End.
```

7.5. Из очереди, информационной частью которой являются вещественные числа, удалить элемент, в минимальной степени отличающийся от среднего арифметического значения ее элементов.

Для решения задачи необходимо определить адрес элемента, для которого модуль разности *dSmin* между его информационной частью и средним арифметическим значением *S* минимален. При этом должно быть определено, является ли такой элемент первым или не первым в очереди, поскольку, как было указано в комментарии к задаче 7.3, при удалении произвольного элемента нужно знать адрес предыдущего элемента, а для первого в очереди такого элемента не существует.

Для индикации положения удаляемого элемента в программе используются два указателя - *Pmin* и *Ppred*. Первый из них непосредственно определяет адрес такого элемента, второй - адрес предшествующего элемента. Если *Pmin = Ppred*, то удаляется первый элемент очереди, при *Pmin ≠ Ppred* - любой другой элемент. В последнем случае дополнительно проверяется, не является ли удаляемый элемент последним в очереди, что используется для изменения положения указателя *R*.

```
Program Task705;
Type PoinType = ^DynType;
     DynType = record
        Inf : real;
        Next : PoinType;
     end;
Var  L,R,              { левый и правый указатели очереди }
     Run,              { текущий указатель }
     Pmin,             { указатель минимального элемента }
     Ppred : PoinType; { указатель элемента, предыдущего по }
     n : word;         { отношению к элементу с адресом Pmin }
     S,dSmin : real;
Begin
  Формирование очереди
  If L<>nil then
    Begin
      S:=0; n:=0;
      Run:=L;
      While Run<>nil do           { определение среднего }
```

```
        Begin                          { арифметического зна- }
          S:=S+Run^.Inf; Inc(n);       { чения S элементов }
          Run:=Run^.Next;              { очереди }
        End;
      S:=S/n;                          { определение адреса }
      dSmin:=abs(S-L^.Inf);            { элемента, в мини-  }
      Pmin:=L; Ppred:=L; Run:=L;       { мальной степени от-}
      While Run^.Next<>nil do          { личающегося от S   }
        Begin
          If abs(S-Run^.Next^.Inf)<dSmin then
            Begin
              dSmin:=abs(S-Run^.Next^.Inf);
              Ppred:=Run; Pmin:=Run^.Next
            End;
          Run:=Run^.Next;
        End;
      If (Pmin=L) and (Ppred=L) then   { удаление элемента }
        L:=L^.Next                     { из очереди }
      Else
        If Pmin=R then
          Begin
            R:=Ppred; R^.Next:=nil
          End
        Else
          Ppred^.Next:=Pmin^.Next;
      Dispose(Pmin);
    End;
  Печать очереди
End.
```

7.6. Каждый элемент очереди содержит два вещественных числа, определяющих координаты точки на плоскости. Удалить из очереди точку с максимальным расстоянием от начала координат.

Расстояние точки от начала координат - это корень квадратный из суммы квадратов координат этой точки.

Методика определения адреса удаляемого элемента и удаления элемента из очереди в основном аналогичны программе Task705.

```
Program Task706;
Type PoinType = ^DynType;
     DynType = record
         x,y : real;
       Next : PoinType;
     end;
Var  L,R,              { левый и правый указатели очереди }
     Run,              { текущий указатель }
     Pmax,             { указатель максимального элемента }
     Ppred : PoinType; { указатель элемента, предыдущего по }
     D,Dmax : real;    { отношению к элементу с адресом Pmax }
Begin
  Формирование очереди
  If L<>nil then
    Begin
      Dmax:=sqrt(sqr(L^.x)+sqr(L^.y));
```

```
      Pmax:=L; Ppred:=L; Run:=L;
      While Run^.Next<>nil do
        Begin
          D:=sqrt(sqr(Run^.Next^.x)+sqr(Run^.Next^.y));
          If D>Dmax then
            Begin
              Dmax:=D; Ppred:=Run; Pmax:=Run^.Next;
            End;
          Run:=Run^.Next;
        End;
      If (Pmax=L) and (Ppred=L) then
        L:=L^.Next
      Else
        If Pmax=R then
          Begin
            R:=Ppred; R^.Next:=nil
          End
        Else
          Ppred^.Next:=Pmax^.Next;
      Dispose(Pmax);
    End;
  Печать очереди
End.
```

7.7. Элементы двух очередей, содержащих целочисленные значения, упорядочены по возрастанию. Сформировать из этих элементов новую очередь, сохранив их исходную упорядоченность. Группировку элементов очереди не производить.

Алгоритм реализации задачи 7.7 во многом аналогичен программе Task113, выполняющей подобную работу по отношению к одномерным массивам.

В стартовой части программы Task707 текущие указатели *Run1* и *Run2* устанавливаются на начало исходных очередей 1 и 2 (*L1* и *L2*), конечным указателям *L3* и *R3* новой очереди присваивается пустое значение.

Программа Task707 состоит из трех основных фрагментов.

Фрагмент 1. Если обе исходные очереди не пустые, то выполняются следующие действия:

- отводится память для нового элемента очереди 3;
- сравниваются два элемента *Run1^.Inf* и *Run2^.Inf* исходных очередей;
- меньший из элементов записывается в очередь 3;
- в той очереди, из которой переписан элемент, выполняется переход к указателю следующего элемента (*Run1^.Next* или *Run2^.Next*).

Работа фрагмента 1 продолжается до тех пор, пока не будет исчерпана очередь 1 или очередь 2. Неиспользованные элементы оставшейся очереди обрабатываются в фрагменте 2 или фрагменте 3.

Фрагмент 2 вступает в работу, если очередь 2 пустая, а очередь 1 содержит еще хотя бы один элемент (*Run1≠**nil***). Это имеет место в двух случаях:

- очередь 2 изначально пустая (фрагмент 1 не работает);
- при отработке фрагмента 1 первой исчерпана очередь 2.

В фрагменте 2 все имеющиеся еще элементы очереди 1 переписываются в очередь 3.

Фрагмент 3 аналогичен фрагменту 2, но по отношению ко второй очереди.

```
Program Task707;
Type PoinType = ^DynType;
```

```
      DynType = record
          Inf : integer;
        Next : PoinType;
      end;
Var   Cond3 : boolean;
      L1,R1,L2,R2,     { конечные указатели исходных очередей }
      L3,R3,           { конечные указатели новой очереди }
      Run1,Run2,Run3   { текущие указатели }
           : PoinType;
Begin
  Формирование очередей 1 и 2
  Run1:=L1; Run2:=L2;
  L3:=nil; R3:=nil; Cond3:=true;
  If (L1<>nil) and (L2<>nil) then
    Begin
      While (Run1<>nil) and (Run2<>nil) do { Фрагмент 1 }
        Begin
          New(Run3);
          If Run1^.Inf<Run2^.Inf then
            Begin
              Run3^.Inf:=Run1^.Inf;
              Run1:=Run1^.Next;
            End
          Else
            Begin
              Run3^.Inf:=Run2^.Inf;
              Run2:=Run2^.Next;
            End;
          Run3^.Next:=nil;
          If Cond3 then
            Begin
              L3:=Run3; Cond3:=false
            End
          Else
            R3^.Next:=Run3;
          R3:=Run3;
        End;
    End;
  While Run1<>nil do                        { Фрагмент 2 }
    Begin
      New(Run3);
      Run3^.Inf:=Run1^.Inf; Run3^.Next:=nil;
      If Cond3 then
        Begin
          L3:=Run3; Cond3:=false
        End
      Else
        R3^.Next:=Run3;
      R3:=Run3; Run1:=Run1^.Next;
    End;
  While Run2<>nil do                        { Фрагмент 3 }
    Begin
      New(Run3); Run3^.Inf:=Run2^.Inf;
      If Cond3 then
        Begin
          L3:=Run3; Cond3:=false
        End
```

```
      Else
        R3^.Next:=Run3;
      R3:=Run3; Run2:=Run2^.Next;
    End;
  Печать новой очереди
End.
```

7.8. Элементом очереди является строка с объявленной длиной 80 символов. Удалить из очереди строку, содержащую максимальное количество цифр.

С точки зрения организации поиска удаляемого элемента и его удаления программа Task708 имеет некоторые отличия от программы, реализующей задачу 7.5. Эти отличия заключаются в следующем:

- вместо двух указателей *Pmin* и *Ppred* для индикации удаляемого элемента используется один указатель *Elem*, определяющий адрес элемента, который предшествует удаляемому;
- для определения порядкового номера удаляемого элемента (первый или не первый) применяется булевская переменная *CondDig*; этой переменной первоначально присваивается значение *true;* если в процессе просмотра очереди обнаружится элемент, в строке которого содержится большее количество цифр, чем в первом элементе, то переменной *CondDig* присваивается значение *false;*
- в качестве буферного указателя для задания адреса удаляемого элемента используется переменная *Del*; если *CondDig = true*, этой переменной присваивается значение указателя *L*, в противном случае - значение *Elem^.Next.*

В программе предусмотрен соответствующий перенос указателей *L* и *R*, если удаляется первый или последний элементы очереди.

```
Program Task708;
Const  Digits = ['0'..'9'];
Type  PoinType = ^DynType;
      DynType = record
          St : string[80];
        Next : PoinType;
      end;
Var  i,Dig,DigMax : byte;
     CondDig : boolean;
     ch : char;
     L,R,             { левый и правый указатели очереди }
     Run,             { текущий указатель }
     Elem,            { Elem^.Next - удаляемый элемент }
     Del : PoinType; { буферный указатель }
Begin
  Формирование очереди
  Run:=L;
  If Run<>nil then
    Begin
      CondDig:=true; DigMax:=0;
      For i:=1 to length(Run^.St) do
        Begin
          ch:=Run^.St[i];
          If ch in Digits then
            Inc(DigMax);
        End;
      While Run^.Next<>nil do
        Begin
```

```
          Dig:=0;
          For i:=1 to length(Run^.Next^.St) do
            Begin
              ch:=Run^.Next^.St[i];
              If ch in Digits then
                Inc(Dig);
            End;
          If Dig>DigMax then
            Begin
              DigMax:=Dig; CondDig:=false;
              Elem:=Run;
            End;
          Run:=Run^.Next;
        End;
      If CondDig then
        Begin
          Del:=L; L:=L^.Next;
          If Del=R then
            R:=nil;
          Dispose(Del);
        End
      Else
        Begin
          Del:=Elem^.Next;
          If Del=R then
            R:=Elem;
          Elem^.Next:=Elem^.Next^.Next;
          Dispose(Del);
        End;
    End;
  Печать очереди
End.
```

7.9. Задана целочисленная очередь, в которой могут быть подряд идущие одинаковые элементы. Если такие группы элементов обнаружены, оставить в каждой из них лишь один элемент.

В программе Task709 последовательно сравниваются два смежных элемента: *Run* и *Run^.Next*. Если информационные части сравниваемых элементов одинаковы, то второй из них удаляется. При удалении последнего элемента указатель *R* переносится на предыдущий элемент. Изменение значения переменной *Run (*переход к новой паре элементов) выполняется, если не производилось удаления элемента; в противном случае первый элемент сравниваемой пары остается прежним, а второй замещается следующим элементом.

```
Program Task709;
Type PoinType = ^DynType;
     DynType = record
         Inf : integer;
        Next : PoinType;
     end;
Var  L,R,              { левый и правый указатели очереди }
     Run,              { текущий указатель }
     Elem : PoinType; { указатель удаляемого элемента }
Begin
  Формирование очереди
```

```
  Run:=L;
  If Run<>nil then
    While Run^.Next<>nil do
      If Run^.Inf=Run^.Next^.Inf then
        Begin
          Elem:=Run^.Next;
          If Run^.Next=R then
            R:=Run;
          Run^.Next:=Run^.Next^.Next;
          Dispose(Elem);
        End
      Else
        Run:=Run^.Next;
  Печать очереди
End.
```

7.10. Элементами очереди являются два целых числа *m* и *n*. Если в *i*-ом и (*i*+1)-ом элементах имеет место $n_i = n_{i+1}$ (i = 1,3,5,...), то удалить один из них, присвоив оставшемуся значение $m = \max(m_i, m_{i+1})$. Если количество элементов очереди нечетное, то последний ее элемент не анализируется.

Основное различие между программами Task709 и Task710 заключается в том, что в первой из них анализируются все смежные элементы (1 и 2, 2 и 3, 3 и 4 и т.д.), а во второй анализ производится попарно (1 и 2, 3 и 4, 5 и 6 и т.д.). Поэтому во второй программе в случае удаления элемента передвижение по очереди производится на один шаг (*Run:=Run^.Next*), при различии сравниваемых элементов - на два шага (*Run:=Run^.Next^.Next*).

Цикл ***While*** работает до тех пор, пока существуют в очереди оба элемента сравниваемой пары. Если количество элементов очереди нечетное, то на последнем этапе повторения цикла будет иметь место *Run* ≠ ***nil*** и *Run^.Next* = ***nil,*** что приводит к прекращению работы этого цикла.

Отметим здесь следующую деталь: запись условия повторения цикла в виде (*Run^.Next*≠ ***nil***) ***and*** (*Run* ≠ ***nil***) была бы неправильной, так как при *Run* = ***nil*** значение *Run^.Next* не существует. Запись этого же условия в виде (*Run* ≠ ***nil***) ***and*** (*Run^.Next* ≠ ***nil***) не нарушает корректности работы программы, так как вычисление логического выражения производится по так называемой короткой схеме. Это означает, что вычисление выражения прекращается, как только становится очевидным его результат. В данном случае два операнда соединены операцией логического умножения. При *Run* = ***nil*** первый операнд имеет значение *false*, тогда результат логического умножения также будет иметь значение *false* вне зависимости от значения второго операнда. Поэтому при *Run* = ***nil*** значение операнда (*Run^.Next*≠***nil***) в этом выражении не определяется.

```
Program Task710;
Type PoinType = ^DynType;
     DynType = record
          m,n : integer;
         Next : PoinType;
     end;
Var  L,R,              { левый и правый указатели очереди }
     Run,              { текущий указатель }
     Elem : PoinType; { указатель удаляемого элемента }
Begin
```

```
    Формирование очереди
    Run:=L;
    While (Run<>nil) and (Run^.Next<>nil) do
      If Run^.n=Run^.Next^.n then
        Begin
          Elem:=Run^.Next;
          If Run^.Next^.m>Run^.m then
            Run^.m:=Run^.Next^.m;
          If Run^.Next=R then
            R:=Run;
          Run^.Next:=Run^.Next^.Next;
          Dispose(Elem);
          Run:=Run^.Next;
        End
      Else
        Run:=Run^.Next^.Next;
    Печать очереди
  End.
```

Задания для самостоятельной работы

7.11. В составе целочисленной очереди имеется единственный нулевой элемент, разделяющий ее на две части. Сформировать две новые очереди, записав в них соответственно первую и вторую части исходной очереди.

7.12. В составе целочисленного стека имеется единственный нулевой элемент, разделяющий его на две части. Переставить эти части местами, изменив лишь информационные связи (указатели) между элементами стека.

7.13. Заданы две очереди *S1* и *S2,* в которых записаны ненулевые элементы двух разреженных целочисленных матриц одинакового размера. Каждый компонент очереди содержит индексы *i, j* и значение $a_{i,j}$ элемента матрицы. Сформировать новую очередь *S3,* записав в нее элементы суммарной матрицы. Если при сложении элементов $a_{i,j}$ из очередей *S1* и *S2* образуется нулевое значение, такой элемент в очередь *S3* не записывать.

7.14. Элементы очереди - целые числа. Сформировать новую очередь, в которую переписать сначала все положительные числа в их исходном относительном порядке, а затем - отрицательные числа в обратном порядке. Нулевые числа в новую очередь не включать.

7.15. В циклической очереди последовательно записаны числа 1, 2, 3, ..., n . Задано также значение $m > 1$. Начиная отсчет от начала очереди и двигаясь по часовой стрелке, удалить m-ый элемент.

Примечание. В частном случае может быть $m > n$.

7.16. В каждой строке текстового файла записано малыми буквами одно слово анализируемого текста. Длина слова не превышает 15 символов. При однократном чтении файла сформировать очередь, каждый элемент которой содержит две переменные: значение слова и количество его повторений в тексте. Слова в очереди должны быть расположены в алфавитном порядке. Группировку очереди не производить.

7.17. Элементы вещественного массива $X = (x_1, x_2, ..., x_n)$ содержат большое количество нулевых элементов. Ненулевые элементы x_i записаны в очередь в порядке увеличения их индексов. Компонент очереди содержит индекс элемента i и его значение x_i . Требуется элементу x_k присвоить новое значение b . Если элемент x_k отсутствует в очереди, его нужно в нее добавить.

7.18. Элементы вещественного массива $X = (x_1, x_2, ..., x_n)$ содержат большое количество нулевых элементов. Ненулевые элементы x_i записаны в очередь в порядке увеличения их индексов. Компонент очереди содержит индекс элемента i и его значение x_i. Требуется элемент x_k удалить из очереди, если он в нее был записан.

7.19. В текстовом файле записано несколько последовательностей целых чисел, каждая из которых заканчивается нулем. Сформировать типизированный файл, в котором числа каждой последовательности записаны в обратном порядке. В качестве буфера использовать стек, в котором временно размещать числа очередной последовательности.

7.20. В текстовом файле записаны элементы полинома по убывающим степеням (значение коэффициента и значение степени). Элементы с нулевым коэффициентом в файл не включаются. Сформировать очередь, содержащую коэффициенты полинома (в том числе нулевые), после чего по схеме Горнера вычислить его значение, введя с клавиатуры аргумент полинома.

7.21. Каждый элемент очереди содержит два вещественных числа, определяющих координаты точки на плоскости. Найти среди элементов очереди точку, наиболее близкую к первой точке, и удалить ее из состава очереди.

7.22. Информационная часть стека содержит неупорядоченные целочисленные значения. Если среди элементов стека есть такие, что *Inf* [*i*] < *b* < *Inf* [*i*+1] (*Inf*[*i*] - информационная часть *i*-го элемента очереди, *b* - произвольное целое число), то вставить значение *b* между этими элементами.

7.23. Элементом очереди является строка с объявленной длиной 80 символов. В частном случае строка может быть пустой. Удалить из очереди все пустые строки.

7.24. Элементами очереди являются целочисленные переменные, упорядоченные в порядке возрастания или убывания. Первый элемент может повторяться несколько раз. Изменить очередь таким образом, чтобы в ее начале было ровно два одинаковых элемента.

7.25. Для заданного дека с целочисленными компонентами проверить, является ли он симметричным относительно своей середины. Если при этом окажется, что элементы сравниваемой пары не равны друг другу, то удалить эту пару из дека.

7.26. Для заданного дека с целочисленными компонентами проверить, является ли он симметричным относительно своей середины. Если при этом окажется, что элементы сравниваемой пары не равны друг другу, то добавить в дек два новых элемента, обеспечивающие симметричность создаваемых при этом двух пар элементов..

7.27. Информационная часть компонента очереди имеет два элемента: значение и количество повторений (оба – целые числа). При вводе из файла формировать очередь в порядке возрастания значений ее компонентов. Если введенное численное значение уже имеется в очереди, то добавить единицу к счетчику повторений.

7.28. Заданы две очереди, упорядоченные по убыванию. Элементы второй очереди последовательно включить в состав первой очереди, не нарушая ее упорядоченность. При этом в программе должна быть учтена упорядоченность второй очереди.

7.29. Используя заданные две исходные очереди, сформировать третью очередь, включив в нее по одному разу элементы, которые входят хотя бы в одну из исходных очередей.

7.30. Если в заданной очереди имеются подряд идущие одинаковые элементы, то оставить в каждой такой группе лишь один элемент, удалив остальные из состава очереди.

7.31. Исходными данными являются строковый массив и очередь. В массиве записаны наименования *n* городов. Элемент очереди – это пара $(i, j), i = 1..n, j = 1..n, i \neq j$, в которой заданы номера городов, непосредственно соединенных дорогой. Определить список городов, которые напрямую сообщаются более чем с тремя городами.

СОДЕРЖАНИЕ