МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

В.В. Пшеничников

Языки и методы программирования. Курс лекций

Электронное учебное пособие

САМАРА
2011

УДК СГАУ: 004 (075)
ББК 22.18
П 932
Автор: Пшеничников Виктор Владимирович

Пшеничников В.В. Языки и методы программирования. Курс лекций [Электронный ресурс]: электрон. учеб. пособие./ В.В. Пшеничников; Минобрнауки России, Самар.гос. аэрокосм. ун-т им. С.П. Королева (нац. исслед. ун-т); – Электрон. текстовые и граф. дан. (4.9п.л., 0.6 MB). – Самара, 2011. – 1 эл. опт. диск (CD ROM). – Систем. требования : ПК Pentium; Window 98 или выше

Соответствует программе курса «Языки и методы программирования» в 1 семестре направления 010400.62(бакалавриат) "Прикладная математика и информатика».

Подготовлено на кафедре геоинформатики и информационной безопасности СГАУ.

Оглавление

1. Вычисление площадей сложных фигур

Если образующие фигуры можно описать достаточно простыми функциями, то ее площадь можно вычислить, используя геометрическую интерпретацию определенного интеграла.

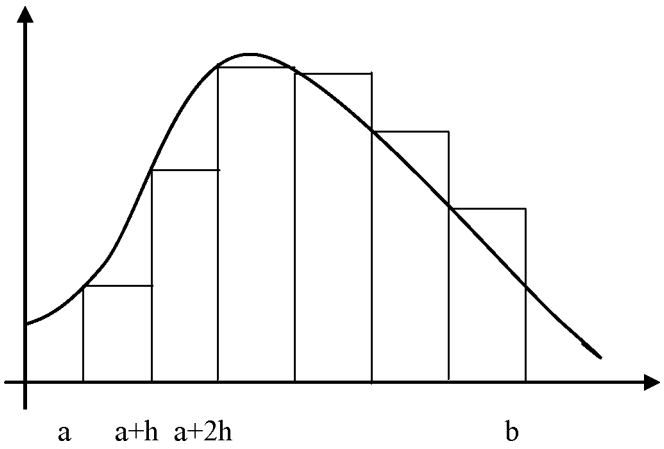


Рис 1. Метод прямоугольников

Используем метод прямоугольников. Интервал интегрирования разобьем на несколько равных отрезков, вычислим значения функции для полученных ординат и подсчитаем сумму площадей полученных прямоугольников.

На рис. 1 видно, что для некоторых отрезков площади прямоугольников считаем с недостатком, а для других с избытком.

Увеличим число разбиений, и точность вычисления площади увеличится.

Если будет необходимо получить площадь с заданной точностью, то будем повторять шаги, удваивая на каждом шаге число разбиений, до тех пор, пока разница между двумя последовательно вычисленными площадями не будет меньше заданной точности.

```
Var a, b, s, h: Real;
          n, i : integer;
Function F(x: Real): Real;
Begin {подинтегральная функция}
 F:=sin(x);
End; {F}
Begin
 Readln(a, b, n);
 {a и b – интервал интегрирования}
 {n - число разбиений}
 H:=(b-a)/n;
 s:=0;
 For i:=0 to n-1 do s:= s + F (a+h*i);
 s:=s*h;
 Writeln(s);
 Readln
End.
```

2. Текстовые файлы

Текстовые файлы хранят информацию в виде последовательности символов. Символы образуют строки произвольной длины. В конце каждой строки находятся два особых символа: #13 #10, которые отделяют строку от следующей. Текстовый файл можно создать в редакторе среды программирования или в блокноте.

Текстовые файлы описываются в программе.

Например, Var F: text;

Файловой переменной назначается конкретный физический файл с помощью оператора

Assign (файловая переменная, имя файла);

Например, Assign (F, 'Inp.txt');

Файл необходимо открыть для чтения (Reset(F)), для записи (Rewrite(F)) или для пополнения (Append(F)).

Создаваемый или пополняемый файл следует обязательно закрыть (Close(F)). Только после закрытия можно быть уверенным

в том, что выходной файл окончательно сформирован. Для чтения и записи используются операторы ввода и вывода с указанием имени файла

Read (F,) Readln (F,) Write(F,) Writeln (F,).

Есть два стандартных текстовых файла: Input (для ввода) и Output (для вывода). Они стандартно назначены на клавиатуру и экран. Описывать их в программе не нужно, а нужно просто переназначить на нужные физические файлы операторами Assign.

Например, Assign (Input, 'Inp.txt') и Assign (Output, 'Out.txt').

В операторах ввода и вывода имена файлов теперь можно не указывать.

Имена файлов для ввода исходных данных и вывода результатов (здесь 'Inp.txt' и 'Out.txt') обычно приведены в условиях задачи.

3. Одномерные массивы

3.1. Описание в программе

Читайте внимательно условие задачи. Описывайте массив по максимально возможному размеру, указанному в условиях задач.

Фактический размер вводится в диалоге с клавиатуры или из текстового файла.

```
Const nmax= 100; { описываем массив с «запасом»}
{в задаче обычно указывается возможный размер}
Type vector= array[1..nmax] of Integer;
Var a: vector;
```

3.2. Ввод и вывод массивов

Так как массив обычно описан «с запасом», сначала вводят конкретный размер массива, а потом поэлементно сам массив.

```
Const nmax=100; { описываем массив с «запасом»}
Type vector=array [1..nmax] of Integer;
Var a: vector;
   i, n, max: Integer;
Begin
 Write ('введите размер массива');
```

```
  Readln(n); {вводим конкретный размер массива}
  Write ('введите элементы массива столбиком');
  For i:=1 to n do Readln(a[i]);
  {можно работать с массивом}
End.
```

3.3. Популярные алгоритмы работы с массивами

3.3.1. Сумма элементов массива

Очень простой алгоритм. Сумма накапливается в переменной «s», которую нужно предварительно обнулить.

```
Const nmax=100; { описываем массив с «запасом»}
Type vector=array[1..nmax] of Integer;
Var a: vector;
   i, n : Integer;
       s: Longint; {возможно и Real}
Begin
  Write ('введите размер массива');
  Readln(n);
  Write ('введите элементы массива столбиком');
  For i:=1 to n do Readln(a[i]);
  s:=0;
  For i:=1 to n do s:=s+a[i];
  Writeln(s);
  Readln
End.
```

3.3.2. Количество положительных элементов в массиве

Программа не требует пояснений. Напомним, что счетчик (в примере счетчик К) нужно обязательно обнулять.

```
Const nmax=100;
Type vector=array[1..nmax] of Integer;
Var a:vector;
   i, n, k: Integer;
Begin
```

```
  Write ('введите размер массива');
  Readln(n);
  Write ('введите элементы массива столбиком');
  For i:=1 to n do Readln(a[i]);
  k:=0;
  For i:=1 to n do
    If a[i] > 0 then k:=k+1;
  Write(k);
  Readln
End.
```

3.3.3. Поиск максимального (минимального) элемента массива

Алгоритм: предполагаем, что максимальным (минимальным) является первый элемент массива. Просматриваем все остальные элементы массива и, если очередной элемент оказывается больше (меньше), чем выбранный нами максимум (минимум), то заменяем максимум (минимум) на этот элемент. Очевидно, мы найдем только величину максимального (минимального) элемента, но не его место в массиве (номер элемента).

```
Const nmax=100; { описываем массив с «запасом»}
Type vector=array [1..nmax] of Integer;
Var a: vector;
   i, n, max: Integer;
Begin
  Write ('введите размер массива');
  Readln(n); {вводим конкретный размер массива}
  Write ('введите элементы массива столбиком');
  For i:=1 to n do Readln(a[i]);
  max:=a[1];
  For i:=2 to n do
    If a[i]>max Then max:=a[i];
  Write('=',max);
  Readln
End.
```

Найдем значение максимального элемента и его номер (описание массива и его ввод не меняем).

Вариант программы:

```
Const nmax=100; { описываем массив с «запасом»}
Type vector=array[1..nmax] of Integer;
Var a:vector;
   i, n, max: Integer;
   num: Integer; { номер максимального элемента}
Begin
 Write ('введите размер массива');
 Readln(n); {вводим конкретный размер массива}
 Write ('введите элементы массива столбиком');
 For i:=1 to n do Readln(a[i]);
 max:=a[1]; num:=1;
 For i:=2 to n do
  If a[i]>max Then  Begin
    max:=a[i]; {запоминаем  новое значение max}
    num:=i; {запоминаем номер этого элемента}
  End;
 Write('максимум =', max , 'номер =', num);
 Readln
End.
```

Но если мы знаем номер элемента, то по номеру всегда получим и сам элемент. С учетом этого, программа примет вид:

```
Const nmax=100; { описываем массив с «запасом»}
{ в задаче обычно указывается возможный размер}
Type vector=array[1..nmax] of Integer;
Var a:vector;
   i, n: Integer;
   num: Integer; {номер максимального элемента}
Begin
 Write ('введите размер массива');
 Readln(n);
 Write ('введите элементы массива столбиком');
 For i:=1 to n do Readln(a[i]);
```

```
{считаем, что максимальный элемент первый}
num:=1; { используем только  его номер}
For i:=2 to n do
  If a[i]>a[num] Then num:=i;
Write('максимум =', a[num] , 'номер =', num);
Readln
End.
```

3.3.4. Сортировка простым обменом (метод “пузырька”)

Алгоритм: сравниваем последовательно, начиная с первого, соседние элементы и если предыдущий элемент больше последующего, меняем их местами. После первого просмотра самый большой элемент окажется на своем месте - станет последним элементом массива.

Второй просмотр: рассматриваем часть массива с первого до предпоследнего. Очередной самый большой элемент станет предпоследним.

Количество просмотров элементов массива равно N-1. В примере, отмечены цветом те элементы, которые уже стоят на своих местах и в очередном просмотре не участвуют.

Первый просмотр

Исходный массив	8	6	4	2	1
Сравниваем 1 и 2	6	8	4	2	1
Сравниваем 2 и 3	6	4	8	2	1
Сравниваем 3 и 4	6	4	2	8	1
Сравниваем 4 и 5	6	4	2	1	8

Второй просмотр

Исходный массив	6	4	2	1	8
Сравниваем 1 и 2	4	6	2	1	8
Сравниваем 2 и 3	4	2	6	1	8
Сравниваем 3 и 4	4	2	1	6	8

Третий просмотр

Исходный массив	4	2	1	6	8
Сравниваем 1 и 2	2	4	1	6	8

Сравниваем 2 и 3	2	1	4	6	8

Четвертый просмотр (последний)

Исходный массив	2	1	4	6	8
Сравниваем 1 и 2	1	2	4	6	8

Реализуем этот алгоритм (N-число элементов массива):

```
For k:=1 To N-1 Do
  For i:=1 To N-k Do
    If A[i]>A[i+1] Then Begin
      T:=A[i]; A[i]:=A[i+1]; A[i+1]:=T;
    End; {If}
```

Так как коды символов упорядочены в соответствии английским и русским алфавитом, то этим алгоритмом можно сортировать массивы символов и массивы символьных строк.

Запишем целиком программу с процедурой сортировки массива

```
Const nmax=100;
Type vector=array[1..nmax] of Integer;
Var a: vector;
      i, n: Integer;
Procedure sort(Var a: vector; n: integer);
Var k, i, T: integer;
Begin
  For k:=1 To N-1 Do
    For i:=1 To N-k Do
      If A[i]>A[i+1] Then Begin
        T:=A[i]; A[i]:=A[i+1]; A[i+1]:=t;
      End;
End;
Begin
  Write ('Размер массива= ');  Readln(n);
  Write ('Вводите элементы через пробел или Enter');
  For i:=1 to n do Read(a[i]);
  sort(a, n);
  For i:=1 to n do Write(a[i],' ');
  Readln;
  Readln;
```

End.

3.3.5. Быстрая сортировка

Метод предложен Ч. Э. Р. Хоаром в 1962 году. Быстрая сортировка", хоть и была разработана более 40 лет назад, является наиболее широко применяемым и одним их самых эффективных алгоритмов.

Метод основан на подходе "разделяй - и - властвуй". Общая схема такова:

из массива выбирается некоторый опорный элемент P, например средний элемент, запускается процедура разделения массива, которая перемещает все числа, меньшие, либо равные P, влево от него, а все числа, большие, либо равные P – вправо. Теперь массив состоит из двух подмассивов, причем в левом числа меньшие P (или равные), в правом большие. Для обоих подмассивов, если в подмассиве более двух элементов, рекурсивно запускаем ту же процедуру. В конце получится полностью отсортированная последовательность.

```
Procedure QuickSort(m, t: Integer);
Var i, j, x, w: Integer;
Begin
 i:=m; j:=t; x:=A[(m+t) Div 2];
 Repeat
  While A[i]<x Do Inc(i);
  While A[j]>x Do Dec(j);
  If i<=j Then Begin
   w:=A[i]; A[i]:=A[j]; A[j]:=w;
   Inc(i); Dec(j);
  End
 Until i>j;
 If m<j Then QuickSort(m, j);
 If i<t Then QuickSort(i, t);
End;
```

Ниже написана программа, которую вы можете запустить на своем компьютере (сделайте это обязательно!)

```
Const n=10;{размер массива}
Type
  list = array[1..n] of integer;
Const
   a: list=(9,2,3,7,1,8,5,4,6,10);{заполняем массив числами}
Var   i: integer;
{процедура сортировки}
Procedure QuickSort (m, t: Integer);
{ m и t границы сортируемой части массива }
Var i, j, x, w: Integer;
Begin
  i:=m; j:=t; x:=A[(m+t) Div 2];
  Repeat
    While A[i]<x Do Inc(i);
    While A[j]>x Do Dec(j);
    If i<=j Then Begin
      w:=A[i]; A[i]:=A[j]; A[j]:=w;
      Inc(i); Dec(j);
    End
  Until i>j;
  If m<j Then QuickSort(m, j);
  If i<t Then QuickSort(i, t);
End;
{Основная программа}
Begin
  QuickSort (1,n); {указываем весь массив}
  for i:=1 to n do Write(a[i],' '); {вывод массива на экран}
  Readln;
end.
```

3.4. Поиск данных

3.4.1. Линейный поиск

Задача: поиск в массиве элемента с заданным значением. Последовательно просматриваем элементы массива и сравниваем их значения с искомым значением X.

Например.

```
For i:=1 To N Do If A[i]=X Then k:=i;
```

Находится номер (индекс) последнего элемента равного X (последнее вхождение), при этом массив просматривается полностью.

Первое вхождение ищем обратным циклом.

```
For i:=N DownTo 1 Do If A[i]=X Then k:=i;
```

Просто, но не эффективно! Возможно, вхождение уже найдено, а цикл просмотра продолжается до конца. Изменим программу.

```
{первое вхождение}
i:=1;
While (i<=N) And (A[i]<>X) Do Inc(i);{i:=i+1}
If i=N+1 Then Write('X нет в A')
Else Write('номер=', i);
```

Обратите внимание на заголовок цикла. Как только элемент будет найден, второе условие станет ложным и цикл завершится. При этом i будет равно или меньше N. Если нужный элемент не будет найден, то, в конце концов, станет ложным первое условие и цикл завершится при i большим N.

В программе предполагается, что если первое условие будет ложным, то второе условие не будет проверяться (уже ясно, что все сложное условие ложно). Поэтому условие (i<=N) стоит первым, в противном случае при i=N+1 возникает ошибка «нарушение границ массива» (при включенном контроле {$R+}). Вообще, при отладке (и только при отладке!) программы настоятельно рекомендуем включать такой контроль. При нарушениях границ массивов можно «испортить» значения других переменных, что сделает работу программы не предсказуемой.

Последнее вхождение будем искать аналогично.

```
i:=N;
While (i>0) And (A[i]<>X) Do Dec(i);{i:=i-1}
If i=0 Then Then Write('X нет в A')
Else Write('номер=', i);
```

Н. Вирт предлагает исключить одну из проверок, добавив в конец массива искомый элемент. Предлагаем Вам самим разобраться в алгоритме и программе.

```
{не забудьте увеличить размер массива в описании}
{первое вхождение}
i:=1; A[N+1]:=X;
While A[i]<>X Do Inc(i);
If i=N+1 Then Then Write('X нет в A')
Else Write('номер=', i);
{последнее вхождение - первое с конца}
i:=N; A[0]:=X;
While A[i]<>X Do Dec(i);
If i=0 Then Then Write('X нет в A')
Else Write('номер=', i);
```

Очевидно, что число сравнений зависит от места нахождения искомого элемента. В среднем количество сравнений равно (N+1) Div 2, в худшем случае - элемента нет в массиве, N сравнений

3.4.2. Бинарный поиск

Если массив данных отсортирован, то удается сократить время поиска, используя бинарный (метод деления пополам) поиск.

Отсортированность A позволяет, по результату сравнения со средним элементом массива, исключить из рассмотрения одну из половин. Далее применяем тот же прием по отношение к выбранной половине. Очевидно, не следует начинать поиск, если X>A[n] или X<A[1].

Программная реализация бинарного поиска может иметь следующий вид. Значением функции является индекс искомого элемента или ноль, если элемент не найден.

```
Const nmax=100;
```

```
Type vector=array[1..nmax] of Integer;
Var a:vector;
   i, n: Integer;
   X: Integer; {номер максимального элемента}
Function bin( Var A: vector;  n, X: integer): integer ;
{двоичный поиск X в массиве A, n – размер массива}
Var i, j, m: Integer;
        f: Boolean;
Begin
 i:=1; j:=N;
{На первом шаге рассматривается весь массив}
 f:=False;     {Признак того, что X не найден}
 While (i<=j) And Not f Do Begin
   m:=(i+j) Div 2;
   If A[m]=X Then f:=True {Элемент найден}
   Else
     If A[m]<X Then i:=m+1 {*Исключаем левую часть}
     Else j:=m-1; {*Правую часть.*}
 End;
 if f then  bin:=m else bin:=0;
End;
Begin{основная программа}
 Write ('размер массива='); Readln(n);
 Write ('Вводите элементы массива столбиком');
 For i:=1 to n do Readln(a[i]);
 Write ('искомое число=');   Readln(X);
 Write( bin( a, n, X));
 Readln
End.
```

Предложим еще одну, рекурсивную, реализацию изучаемого поиска[2]. Значением переменной t является индекс искомого элемента или ноль, если элемент не найден.

```
Const nmax=100;
Type vector=array[1..nmax] of Integer;
Var a: vector;
```

```
   i, n, t: Integer;
       X: Integer;
Procedure bin(i, j :Integer; Var t: Integer);
{i и j – диапазон поиска, t результат }
{A и X глобальные переменные}
Var m: Integer;
Begin
 If i>j Then t:=0
 Else Begin
        m:=(i+j) Div 2;
        If A[m]<X Then bin(m+1, j, t)
        Else
          If A[m]>X Then bin(i, m-1, t)
          Else t:=m;
       End;
End;
Begin {основная программа}
 Write ('размер массива='); Readln(n);
 Write ('Вводите элементы массива столбиком');
 For i:=1 to n do Readln(a[i]);
 Write ('искомое число='); Readln(X);
 bin(1, n, t); {при первом обращении – весь массив}
 Write(t);
 Readln
End.
```

4. Символьные строки

4.1. Общие сведения

В TP существуют ограничения на длину строки (не более 255 символов). Если вы используете компиляторы Free Pascal или Delphi (именно они используются в тестирующих системах популярных сайтов), то там таких жестких ограничений нет.

Описание в программе:

```
Var S: String;
```

R: String[10];

Принципиальным отличием первого описания от второго является то, что при первом случае описания максимальная длина строки составляет 255 символов, а во втором – 10.

Строка в памяти размещается последовательно символ за символом, нулевой байт строки содержит текущую длину строки в символьном типе.

4.2. Стандартные функции для работы со строками:

1. Функция Length (S) – возвращает текущую длину строки.

S:='дом на траве'; L:=Length(S); - L=12

2. Функция Copy (S, k, n) копирует часть S строки из n символов, начиная k–го символа. Длина строки S не изменяется.

После выполнения операторов

S:='дом на траве'; S1:=Copy(S, 5, 8);

S1 получит значение S1='на траве'

3. Функция Pos (s1, S) – возвращает первое вхождение подстроки в строку.

После выполнения операторов

S:='гиппопотам'; p:=Pos('по', S);

p получит значение p=4. С четвертой позиции начинается первый слог 'по'.

4. Процедура Delete (S, k, n) – удаляет n символов из строки S, начиная с k-го.

После выполнения операторов

S:='на дворе трава, на траве дрова'; Delete(S, 10,16);

S изменится и будет S='на дворе дрова'

5. Процедура Insert (r, S, n) – вставляет r в S с позиции n.

После выполнения операторов

S1:='ике'; S:='на дворе дрова'; Insert(S1, 8);

S изменится и будет S='на дворике дрова'

6. Процедура Str (x[:6:2],S) – преобразует число x в строку S с указанным форматом. Пример ниже поясняет работу функции.

```
Var k: integer;
```

```
    x: real;
Begin
  x:=37.289;  Str(x:6:2, S); {результат S=' 37.29'}
  k:=37;  Str(x, S); {результат S='37'}
End.
```

7. Процедура Val (S, x, Error) – преобразует строку S в число x, если строка S не содержит недопустимых для числа символов, то Error = 0.

```
Var k: integer;
    x: real;
Begin
  S:= '37.289'; Val (S, x, Err); { Err=0}
  S:= '37.289';
  Val (S, k, Err);
  {Err≠0, строка содержит «точку», а k целое}
  S:= '37'; Val(S, k, Err); { Err=0}
End.
```

4.3. Сравнение строк

Символы упорядочены по своим кодам так, что 'a' < 'z', 'A' < 'Z', ‘А’ < ‘Я’, ‘а’<’я’. Операции сравнения символов =,<>,<,>,<=,>= - действуют в соответствии со значениями кодов.

Сравнение строк – «лексикографическое», строки сортируются по алфавиту также как фамилии в вашем школьном журнале.

Пример. В исходном файле список учеников. Упорядочить список и записать в другой файл в виде пронумерованного списка. Не используем стандартные файлы ввода и вывода, поэтому в программе появилось описание текстовых файлов.

```
Const n=100; { максимальное число строк в списке}
Var  a: array [1..n] of string[20];
     Fi, Fo: text;
      i, k , j: integer;
           R: string[20];
Begin
```

```
  {файл Sp.txt должен быть заранее создан}
  Assign (Fi, 'Sp.txt'); Reset (Fi);
  Assign (Fo, 'NewSp.txt'); Rewrite (Fo);
  i:=0; {число строк}
  While not Eof(Fi) do Begin {читаем до конца файла}
    Inc (i);
    Readln(Fi,a[i]);
  End;
  {получили массив из i строк}
  {сортируем массив}
 For k := 1 to (i-1) do
    For j := 1 to (i-k) do
      If a[j]>a[j+1] then Begin
        R:=a[j];
        a[j] := a[j+1];
        a[j+1] := R;
      end;
  {пишем строки в файл, пронумеровывая}
  For k:=1 to i do Writeln(Fo, k, '. ', a[k]);
  Close(Fo); {обязательно закрываем файл}
 End.
```

Исходный файл sp.txt	Файл результатов Newsp.txt
Яковлев В.В Абрамов П.П. Петров Н.Н. Сидоров С.С. Михалов М.М.	1. Абрамов П.П. 2. Михалов М.М. 3. Петров Н.Н. 4. Сидоров С.С. 5. Яковлев В.В

4.4. Несколько полезных приемов обработки строк

4.4.1. Убрать из строки пробелы в начале строки.

Проверяем первый символ строки. Если это пробел, то удаляем его. После чего следующий символ становится первым.

```
Var s: string;
    i: integer;
Begin
 Readln(s);
 While s[1]=' ' do Delete(s,1,1);
 Writeln(s);
End.
```

4.4.2. Убрать из строки пробелы в конце строки.

Проверяем последний символ и, если он «пробел», удаляем его.

```
Var s: string;
    i: integer;
Begin
 Readln(s);
{между апострофами один пробел}
 While s [length(s)]=' ' do Delete(s,length(s),1);
 Writeln(s);
End.
```

4.4.3. Убрать из строки все пробелы.

Пока в строке есть пробелы, функция Pos будет возвращать позицию (номер) пробела, если пробелов нет, то ноль. Функция Delete удаляет пробел в этой позиции.

```
Var s: string;
Begin
 Readln(s);
 While Pos (' ', s)<>0 do
 {между апострофами один пробел}
    Delete(s, Pos(' ',s),1);
  Writeln(s);
End.
```

4.4.4. Убрать из строки «лишние» пробелы.

Ищем в строке два соседних пробела и удаляем один из них.

```
Var s: string;
Begin
```

```
  Readln (s);
  While pos ('  ', s)<>0 do
    {между апострофами два пробела}
    Delete(s, pos('  ',s),1);
  Writeln (s);
  Readln
End.
```

4.4.5. Подсчитать количество цифр в натуральном числе.

Вспомните как умно и красиво мы решали эту задачу с использованием операций mod и div. Теперь преобразуем число в строку и определим ее длину. И все!

```
Var  x: integer;
       s: string;
Begin
  readln (x);
  Str(x, s);
  Writeln (Length (s));
  Readln
End.
```

4.4.6. Заменить фрагмент R в строке S на фрагмент T.

Идея проста: пока фрагменты вида R в строке есть (pos(R, S) <>0), определяем позицию первого вхождения K, удаляем его и вставляем на его место фрагмент вида T.

```
While Pos(R, S)<> 0 do Begin
  K:=Pos(R, S);
  Delete(S, K, Length(R));
  Insert(T, S, K);
End;
```

Обратите внимание, что в каждом цикле дважды обращаемся к функции Pos? Один раз в заголовке цикла и второй при вычислении позиции первого вхождения. Улучшим программу, вычисляя K перед каждым выполнением цикла. Теперь одно вычисление K за циклом, а в цикле только одно.

```
K:=Pos(R, S);
```

```
While K<>0 do Begin
  Delete(S,K, Length(R));
  Insert(T, S, K);
  k:=pos(R, S);
End;
```

Наберите эту программу на компьютере, добавив описания переменных и ввод – вывод. Придумайте тесты для проверки программы. «Хитрый» тест: программа не сможет выполнить задачу, если во вставляемом фрагменте содержится заменяемый.

Например, R='12' и T='123'. Объясните возникшие проблемы. На соревнованиях такие «хитрые» тесты вполне возможны.

4.4.7. Строка палиндром

Строка, которая читается одинаково в обоих направлениях, называется строкой палиндромом (строка «перевертыш»).

Надеемся, что предыдущий материал позволит вам самостоятельно разобраться в программе и понять использованный там алгоритм. Основная идея: проверять симметрию строки с двух концов до середины.

В качестве тестовых примеров введите строки с четным и нечетным числом букв.

```
Var s: String;
Function palindro (s: String): Boolean;
{алгоритм реализован в виде функции}
  Var  n, i: Byte;
  Begin
    i:=1;
    n:=length (s);
    While (i<=n div 2) and(s[i]=s[n-i+1]) Do inc(i);
    If i>n div 2 Then palindro:=true Else palindro:=false;
  End;
Begin{основная программа}
  Readln(s);
  If palindro(s) Then write('yes') Else write('no');
  Readln
```

```
End.
```

4.4.8. Выделение слов из строки

Под словами будем понимать последовательности символов разделенных пробелами (кроме, соответственно, первого и последнего слов).

Пусть s='мама мыла раму'. Признаком конца слова можно считать пробел, кроме последнего слова. Чтобы не обрабатывать последнее слово специальным образом, добавим в конец строки оператором S:=s+' ' символ «пробел».

Просмотрим все элементы строки и если символ не пробел, то добавляем его к строке t. Если пробел, то в t уже целое слово. Выдаем его на экран, и начинаем формировать в t новое слово. Перед формированием «очищаем» слово t (t:=''- пустая строка).

```
Var s, r, t: string;
          i: integer;
Begin
 Readln(s);
 s:=s+' '; t:='''; {t пустая строка}
 For i:=1 to length(s) do
  If s[i]<>' ' then t:=t+s[i]
  else Begin
         Writeln(t);
         t:='';
       End;
 readln
End.
```

Используем эту идею для решения более сложной задачи: найти в строке самое длинное слово – палиндром. Для этого формируем как прямое слово t:=t+s[i], так и обратное r:=s[i]+r. Кроме того, последовательно проверяем длины слов и ищем самое длинное.

```
Var s, r, t, wmax: string;
          i, lmax, l: integer;
Begin
```

```
Readln(s); s:=s+' ';
r:='';{"обратное" слово-пустая строка}
t:='';{"прямое" слово-пустая строка }
wmax:='';{искомое слово – пустая строка}
For i:=1 to length(s) do
 If s[i]<>' ' then {символ не равен пробелу}
   Begin
      r:=s[i] + r;  t:=t + s[i]
   End
 else Begin
      If r = t then
       If length(r)>length (wmax)
       then wmax:=r;
      r:=''; t:=''; {готовимся формировать новые слова}
      End;
Write(wmax);
Readln
End.
```

Теперь представим, что «очень много» символов последовательно записано в текстовом файле (на олимпиадах обычно бывает именно так). Тогда последним символом окажется код #26 (признак конца файла). Но, если при создании файла в конце строки будет нажата клавиша «Enter», то в конце файла могут появиться и символы #13 (CR) или (и) #10 (LF). Не забудьте об этом при участии в соревнованиях, именно из этого символа команда СГАУ на четвертьфинале чемпионата мира сдала программу только с третьей попытки, заработав 40 минут штрафного времени. Обязательно закрывайте выходной файл, иначе результат в нем может не появиться.

Используем стандартные файлы ввода и вывода. Тогда их можно не описывать и не указывать их имена в операторах ввода и вывода

Будем считывать символы из файла последовательно и по алгоритму, изложенному выше, решим задачу.

```
Var r, t, wmax: string;
```

```
                ch: char;
Begin
  Assign (input, 'input.txt'); Reset(input);
  Assign (output, 'output.txt'); Rewrite(output);
  r:=''; t:=''; wmax:='';
  Repeat
    Read (ch);
    If (ch<>' ')and(ch<>#26)and(ch<>#10)and(ch<>#13)  then
      Begin r:=ch+r; t:=t+ch End
    else Begin
          If r=t then
            If length (r)>length (wmax) then wmax:=r;
           r:=''; t:='';
          End;
   Until (ch=#26) or (ch=#10) or (ch=#13);
   Write (wmax);
   Close (output);
End.
```

5. Множества

Предлагаем изучить множества самостоятельно по [1, 2]. При изучении обратите внимание на типы данных в множествах и организации ввода и вывода множеств. Дадим лишь краткие пояснения и рекомендации. Практика программирования показывает, что наиболее полезным видом множеств является множество символов.

Появляется новая для вас операция In (проверка принадлежности элемента множеству), операции объединения множеств (+), пересечения множеств (*) и вычитания (-).

Все остальные пояснения в примерах.

5.1. Множество символов в строке

Множество символов, принадлежащих строке легко получить по следующей программе:

```
Var M:set of char;
```

```
  S:string;
  i: integer;
Begin
 S:='мама мыла раму';
 For i:=1 to Length(S) do M:=M+[s[i]];
 {просто символ нельзя добавит к множеству M}
 {нужно построить из него множество []}
 {и объединить его с множеством M}
End.
```

5.2. Вывод элементов множества на экран

Используем символьную переменную ch (Var ch: char;), перебираем все значения символов, начиная с пробела (код #32) и, если символ есть в множестве, выдаем его на экран.

```
For ch:=#32 to #255 do
 If ch in M then write(ch);
```

5.3. Ввод множества символов

Множество можно задать в разделе констант.

Например, так

```
Const M: set of char = ['A'..'Z'];
R: set of char = ['0'..'9'];
```

Можно получить в результате присваивания, описав в разделе переменных (Var M:set of char;) и присвоив в программе

```
M:= ['0'..'9', '+', '-'];
```

При вводе с клавиатуры или из текстового файла, заданное множество лучше вводить в виде строки, из символов которой потом легко получить множество или каждый символ сразу добавлять к множеству.

5.4. Количество различных символов в строке

Все пояснения в тексте программы.

```
Var M:set of char;
  S  : string;
  k, i: integer;
```

```
  ch : char;
Begin
 S:='мама мыла раму';
 M:=[]; {пустое множество - обязательно}
 к:=0;  {обязательно обнуляем счетчик}
 For i:=1 to Length(S) do
  If Not (S[i] In M) then {символа еще нет мн-ве}
    Begin
        k:=k+1;
       {добавляем символ к множеству,}
       {чтобы не посчитать его повторно}
        M:=M+[s[i]];
    End;
 Write(k);
 Readln;
End.
```

6. Двухмерные массивы (матрицы)

6.1. Описание матрицы.

```
Const n=4; m=5;
Type matrix=array[1..n, 1..n] of Real;
                              {или другой тип}
Var A:matrix;
```

На рис.2. показан вид описанной матрицы. Обратите внимание, что первый индекс – номер строки, второй - -номер столбца.

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}

Рис. 2. Вид матрицы.

6.2. Ввод элементов матрицы.

Элементы вводятся последовательно по строкам, i – номер строки, j – номер столбца.

```
For i:=1 to n do
  For j:=1 to m do Read(A[i, j]);
```

Если конкретный размер матрицы может меняться, то следует описать матрицу с запасом (максимальный размер по условию задачи), и вводить размеры матрицы, а затем ее элементы.

```
Const max=100;
Type matrix=array[1..max, 1..max] of Real;
                              { или другой тип}
Var A: matrix;
Begin
  Write ('Введите число строк'); Radln (n);
  Write ('Введите число столбцов'); Radln (m);
  Write ('Введите', m*n, 'элементов матрицы');
  For i:=1 to n do
    For j:=1 to m do Read (A[i, j]);
```

6.3. Вывод элементов матрицы.

Вывод матрицы на экран или в текстовый файл нужно производить в «матричной форме» (по строкам и столбцам), разделяя числа пробелами.

```
For i:=1 to n do Begin
   For j:=1 to m do Write (A[i, j], ' ');{ выдаем строку}
   Writeln {переводим курсор на следующую строку}
End;
```

6.4. Основные алгоритмы работы с матрицами.

6.4.1. Сумма элементов матрицы.

Простой алгоритм не требует дополнительных пояснений. Еще раз обращаем внимание на то, что переменная S должна быть обнулена.

```
S:=0;
For i := 1 to n do
  For j := 1 to m do S:= S+ a[i, j];
```

6.4.2. Сумма главной диагонали квадратной матрицы.

Обращаем внимание на то, что элементы главной диагонали имеют равные номера строк (первый индекс) и номера столбцов (второй индекс).

```
S:=0;
For i:= 1 To n Do S:=S+a[i, i];
```

6.4.3. Сумма побочной диагонали квадратной матрицы.

Посмотрите на индексы побочной диагонали матрицы, первый изменяется по возрастанию, второй по убыванию.

```
S:=0;
For i:= 1 To n Do S:=S+a[i, n+1-i];
```

6.4.4. Транспонирование матриц.

Строки исходной матрицы становятся столбцами транспонированной. Исходная матрица (а) имеет n строк и m столбцов. Транспонированная матрица (в) имеет m строк и n столбцов.

```
For i := 1 to n Do
 For j := 1 to m Do b[j, i] := a [i, j];
```

6.4.5. Транспонирование матрицы в том же массиве (транспонирование квадратной матрицы).

Здесь без рабочей ячейки не обойтись.

```
For i := 1 To n-1 Do
  For j := i To n Do
    Begin
      r := a[i, j]; a[i, j] := a[j, i]; a[j, i] := r;
    End;
```

6.4.6. Умножение матриц.

Дана матрица A размерами n*m и матрица B – m*k. После умножения получим матрицу C размерами n*k.

Умножение ведем по формуле $C_{ij} = \Sigma a_{it} * b_{tj}$

```
For i := 1 To n Do
  For j := 1 To k Do begin
    S:=0;
    For t := 1 To m Do S:= S+A[i, t]*B[t, j];
    C[i, j] := S;
  end;
```

6.4.7. Работа с фрагментами матриц

Рассмотрим квадратную матрицу. Решим четыре похожие задачи по одной простой схеме.

A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}

Рис.3. Квадратная матрица

Задачи:

1) сумма элементов на главной диагонали и выше нее.
2) сумма элементов на главной диагонали и ниже нее.
3) сумма элементов на побочной диагонали и выше нее.
4) сумма элементов на побочной диагонали и ниже нее.

Для первой задачи суммирование элементов первой строки нужно начинать с первого элемента до конца строки, второй строки со второго элемента и так до последней строки, где в суммировании участвует только последний элемент

Для второй задачи из первой строки нужно взять только первый элемент, из второй строки два элемента и так до последней строки.

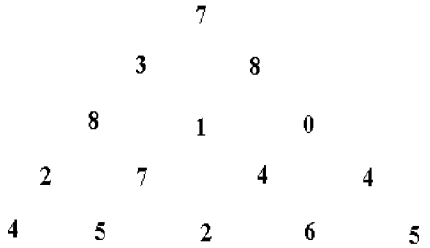
Для третьей задачи суммирование элементов из первой строки нужно взять все элементы, из второй строки на один меньше и так до последней строки.

Для четвертой задачи в первой строке нужно взять только последний элемент, во второй строке два последних и так до последней строки.

7. Динамическое программирование

Достаточно подробно и понятно динамическое программирование описано в [3]. Рассмотрим один из примеров.

Задача «Треугольник». На рисунке изображен треугольник из чисел. Напишите программу, которая вычисляет наибольшую сумму чисел, расположенных на пути, начинающемся в верхней точке треугольника и заканчивающемся на нижней строке треугольника.



- Каждый шаг на пути может осуществляться вниз по диагонали влево или вниз по диагонали вправо.
- Число строк в треугольнике > 1 и <100.
- Треугольник составлен из целых чисел от 0 до 99.

Входные данные запишем в матрицу D.

Матрица D					Матрица R					
7					0	7				
3	8				0	10	15			
8	1	0			0	18	16	15		
2	7	4	4		0	20	25	20	19	
4	5	2	6	5	0	24	30	27	26	24

Будем вычислять матрицу R: array[1..MaxN, 0..MaxN] следующим образом, предварительно обнулив ее элементы:

```
R[1, 1]=D[1, 1]
For i:=2  To N Do
  For j:=1   To i  Do
    R[i, j]=max(D[i,j]+R[i-l,j], D[i, j]+R[i-1, j-1]);
```

где max - функция вычисления максимального из двух чисел.

Осталось найти наибольшее значение в последней строке матрицы R, оно равно 30.

Исходные данные считываем из текстового файла. Используем стандартные текстовые файлы Input и Output (их стандартные назначения: клавиатура и экран), перезначив их на файлы указанные в условии задачи. Пусть первая строка содержит число строк в треугольнике(N), далее N строк исходного треугольника.

Тип данных выбираем Integer, так как исходные данные не выходят за пределы этого типа, а максимальный результат не превысит 99*100 (100 строк в треугольнике, числа в строках до 99).

Ниже приводится полный текст программы с комментариями.

```
Const nmax=100;
Var D: array [1..nmax, 1..nmax] of integer;
   R: array [1..nmax, 0..nmax] of integer;
   i, j, n, rez: integer;
Function max (a, b: integer):integer;
{функция выбирает большее из двух чисел}
Begin
 if a>b then max:=a else max:=b
End;
Begin
 Assign (input,'inp.txt'); Reset(input);
 Assign (output,'out.txt'); Rewrite(output);
 Readln (n); { вводим число строк треугольника}
 For i:=1  to n Do {вводим элементы матрицы D}
  For j:=1   To i  Do Read(D[i, j]);
 For i:=1  to n Do {заполням матрицу R нулями }
  For j:=0   To n  Do R[i, j]:=0;
 {реализуем основной алгоритм}
 R[1,1]:=D[1, 1];
 For i:=2  to n Do
  For j:=1   To i  Do
    R[i, j]:=max(D[i,j]+R[i-1,j], D[i, j]+R[i-1, j-1]);
 {ищем наибольший элемент последней строки}
```

```
rez:=R[n, 1];
For i:=2  to n Do
  If rez < R[n, i]  then rez:=R[n, i];
Write (rez); {выводим результат в файл}
Close (input); Close(output);
End.
```

8. Цифровая геометрия

8.1. Основные отношения

Расстояние между двумя точками с координатами (x1, y1) и (x2,y2)

$$d=\sqrt{(x2-x1)^2+(y2-y1)^2}$$

Уравнения прямой линии:

каноническое уравнение: Ax + By + C = 0

уравнение с угловым коэффициентом: y = kx + b,

уравнение прямой, проходящей через две точки (x1, y1) и (x2,y2):

$$\frac{y-y1}{y2-y1}=\frac{x-x1}{x2-x1}$$

Расстояние от точки P(x0, y0) до прямой Ax+By+C=0 находиться по формуле:

$$s=\frac{|Ax0+By0+C|}{\sqrt{A^2+B^2}}$$

Три точки лежат на одной прямой, если

$$\frac{y3-y1}{y2-y1}=\frac{x3-x1}{x2-x1}$$

Фрагмент программы проверки, лежат ли точки на одной прямой с учетом вещественных координат и заданной точности Eps, будет иметь вид

```
If  Abs((y3-y1)*(x2-x1)-(x3-x1)*(y2-y1))<=Eps then Write('Yes')
Else Write('No');
```

Две прямые A1x + B1y + C1 = 0 и A2x + B2y + C2 = 0

пересекаются в точке:

$$x = \frac{B1C2 - B2C1}{A1B2 - A2B1} \qquad y = \frac{C1A2 - C2A1}{A1B2 - A2B1}$$

Угол между ними:

$$tg\alpha = \frac{A1B2 - A2B1}{A1A2 - B1B2}$$

Прямые параллельны, если A1*B1 - A2*B1 = 0

Прямые перпендикулярны, если A1*A2 + B1*B2 = 0

Уравнение окружности с радиусом R и центром в (x0, y0):

$$(x - x0)^2 + (y - y0)^2 = R^2$$

8.2. Взамное расположение точки и прямой

Если в уравнение прямой проходящей через две точки поставить координаты третьей точки, то получим ноль, если точка принадлежит прямой или числа разных знаков, в зависимости от положения точки относительно прямой. Для вещественных чисел нужно учитывать требуемую точность.

```
Function pr(x, y, x1, y1, x2, y2:longint):longint;
{взаимное расположение точки и прямой}
Begin
  pr:=(x-x1)*(y2-y1)-(x2-x1)*(y-y1)
End;
Var x, y, x1, y1, x2, y2, p: longint;
Begin
   x1:=0; y1:=0; x2:=4; y2:=4;
   Readln(x, y);
   p:=pr(x, y, x1, y1, x2, y2);
  If p<0 then Write('L')
  else if p>0 then Write('p')
       else Write('0');
  Readln
End.
```

8.3. Площадь многоугольника

Известная из школьной программы формула Герона вычисляет площадь треугольника по длинам сторон треугольника (L1, L2, L3).

$$S=\sqrt{P(P-L1)(P-L2)(P-L3)}$$

где P=(L1+L2+L3)/2- полупериметр треугольника.

В задачах, где треугольник задан координатами вершин, проще использовать формулу ориентированной площади треугольника[5].

Пусть вершины треугольника имеют коодинаты (x1, y1), (x2, y2) и (x3, y3). Ориентированная площадь равна площади треугольника, но имеет знак. Знак площади зависит от порядка обхода вершин, при обходе против часовой стрелки знак положительный, по часовой отрицательный.

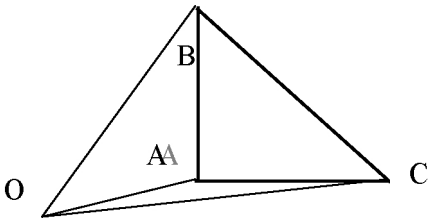


Рис. 4. Ориентированная площадь

Ориентированная площадь треугольника – это его обычная площадь, только со знаком. Знак у ориентированной площади треугольника ABC такой же, как у ориентированного угла между векторами $\overline{AB}$ и $\overline{AC}$, то есть ее знак зависит от порядка перечисления вершин. На рис.4 треугольник ABC прямоугольный. Его ориентированная площадь равна $(\overline{AB} * \overline{AC})/2$. Эту же величину можно вычислить другим способом. Площадь треугольника ABC получится, если из площади треугольника OBC вычесть площади треугольников OAB и OCA. Таким образом, нужно просто сложить ориентированные площади треугольников OAB, OBC и OCA.

Аналогично можно вычислить площадь любого многоугольника, не обязательно выпуклого. Нужно просто сложить ориентированные площади треугольников, одна вершина которых - (0, 0), а две другие - соседние вершины многоугольника. Ориентированная площадь параллелограмма построенного на векторах $a=(X_1, Y_1)$ и $b=(X_2, Y_2)$ равна $2S=X_1*Y_2-X_2*Y_1$, где S ориентированная площадь треугольника.

Запишем программу для вычисления площади по предложенному алгоритму. Рекомендуем использовать ее и для вычисления площадей треугольников, хотя для трех вершин можно получить формулу и не использовать циклическую программу.

В программе координаты вершин нужно вводить последовательно, обходя многоугольник по (или против) часовой стрелке.

```
Const nmax=100;
var
  x, y: array[1..nmax] of real;
  i, n: integer;
  s: real;
begin
  Read(n);
  For i:=1 to n do Read(x[i], y[i]);
  s:= 0.0;
  for i:= 1 to n-1 do begin
    s:=s+ (x[i] * y[i+1]-x[i+1] * y[i]);
  end;
  s:=s+x[n]*y[1]-x[1]*y[n];{от последней к первой}
  s:= abs(s)*0.5;
  Write(s:0:3);
end.
```

8.4. Выпуклая оболочка

Для конечного множества точек выпуклой оболочкой всегда будет выпуклый многоугольник, все вершины которого есть точки этого множества, а все остальные точки множества оказываются

внутри или на сторонах этого многоугольника. Наша задача найти вершины выпуклой оболочки среди заданного множества точек. Будем перечислять точки в порядке их обхода против часовой стрелки. Наиболее простым является алгоритм Джарвиса. Перечисление точек начнем с правой нижней точки P1. Следующей точкой оболочки будет точка P2, при выборе которой все остальные точки лежат слева от вектора $\overline{P1P2}$ (функции vect). Очевидно, что для выбора очередной точки нужно проверять на это условие все множество точек. Если подходящих точек несколько, то выбираем точку, для которой длина вектора $\overline{P1P2}$ максимальна (функция dist). Выбранные точки записываются в массив b. Приведенная программа целиком заимствована из работы [5].

Программа использует целочисленные координаты, в ней нет операций деления и извлечения корня, операций с вещественным результатом. При вещественных координатах возникнут проблемы принадлежности точек выпуклой оболочки с заданной точностью, а именно, точки вблизи границы оболочки могут или не попасть в нее, или наоборот попасть, хотя они находятся снаружи.

Внимательно изучите программу, посмотрите, как авторы проверяют замыкание оболочки.

Попробуйте написать программу самостоятельно, этот алгоритм часто встречается в задачах олимпиад. Там, конечно, не будет явно звучать задача построения выпуклой оболочки, но вы должны увидеть в «замысловатом» сюжете, что это именно так.

Если вы не знакомы с типом Record, то следует читать [1] или удовлетвориться нашим кратким пояснением. Запись (Record) это структура данных, состоящая из фиксированного числа компонент (полей). В отличие от массива компоненты могут быть различных типов. Для ссылки поля именуются. Например, в программе поле x в массиве a, будет доступно по имени a[i].x, а поле y по имени a[i].y. В предыдущих примерах мы использовали для координат точек два массива.

```
Type vv=Record
       x, y:longint;
     End;
```

```
Var a, b: array[1..101] of vv;
   min, m, i, j, k, n:integer;
Function vect(a1, a2, b1, b2: vv):longint;
{косое произведение векторов a1a2 и в1в2}
Begin
 vect:=(a2.x-a1.x)*(b2.y-b1.y)-(b2.x-b1.x)*(a2.y-a1.y)
End;
Function dist2(a1, a2:vv): longint;
{квадрат длины вектора}
Begin
 dist2:=sqr(a2.x-a1.x)+sqr(a2.y-a1.y)
End;
Begin
 Readln (n); { количество точек}
 For i:=1 to n do Read (a[i].x, a[i].y);
 { ищем правую нижнюю точку}
 m:=1;
 For i:=2 to n do
  if a[i].y<a[m].y then m:=i
  else
    if (a[i].y = a[m].y) and
      (a[i].x > a[m].x) then m:=i;
 {запишем ее в массив выпуклой оболочки «в»}
 b[1]:=a[m];
 {и переставим на первое место в исходном массиве}
 a[m]:=a[1];
 a[1]:=b[1];
 k:=1;
 min:=2;
 repeat
 {ищем очередную вершину выпуклой облочки}
 For j:=2 to n do
   if (vect (b[k],a[min],b[k], a[j])<0) or
    ((vect (b[k],a[min],b[k], a[j])=0) and
    (dist2 (b[k], a[min])< dist2(b[k], a[j])))
```

```
    then min:=j;
  k:=k+1;
  {записана очередная вершина}
  b[k]:=a[min];
  min:=1;
    until ( b[k].x=b[1].x) and (b[k].y=b[1].y);
  {пока ломаная не замкнется}
 For j:=1 to k-1 do
   Writeln(b[j].x, ' ',b[j].y);
 readln
end.
```

9. Алгоритмы на графах

9.1. Алгоритм Флойда

Наиболее просто найти кратчайший путь между каждой из пар вершин можно с помощью алгоритма Флойда. Пусть в элементе матрицы A[i, j] хранится длина пути из вершины i в вершину j. Если же длины пути из вершины i в вершину k и пути из вершины k в вершину j таковы, что их сумма меньше, чем значение данного элемента матрицы, то его следует переопределить. Для реализации алгоритма массив A первоначально следует заполнить элементами матрицы весов, обозначая отсутствие ребра между двумя вершинами “бесконечностью” — числом, заведомо превосходящим длину любого пути в рассматриваемом графе, но меньшим, чем максимальное значение используемого типа данных, чтобы было возможно выполнять с ним арифметические операции.

```
Const nmax=25;
Var a :array[1..nmax,1..nmax] of real;
      i, j, k, n: integer;
      f, f1: text;
Begin
  Assign (f, '1.txt'); reset(f);
```

```
  Readln (f, n);
  For i:=1 to n do  Begin
   For j:=1 to n do  Read(f, a[i,j]);
   Readln (f);
  End;
  For k := 1 to n do
   For i := 1 to n do
      For j := 1 to n do
        If a [i, j]>a[i, k]+a[k, j] then a[i, j]:= a[i, k]+a[k, j];
  Assign (f1, '2.txt'); Rewrite (f1);
  For i:=1 to 5 do  Begin
   For j:=1 to 5 do  Write(f1,a[i,j]:4:0,' ');
   Writeln(f1);
  End;
  Close(f1);
End.
```

В этом решении в исходной матрице весов будут записаны кратчайшие пути между всеми вершинами.

На соревнованиях в условии задачи вряд ли будет дана матрица весов. Там будет, например, как в задаче «На болоте», она приведена в задачах олимпиад.

Если же нам требуется найти сам кратчайший путь, а не его длину, то при каждом улучшении пути между двумя вершинами мы в соответствующем элементе вспомогательного массива (в программе — w) будем запоминать номер той вершины, через которую кратчайший путь проходит, а затем с помощью рекурсивной процедуры будем его печатать. Идея рекурсии заключается в том, что если мы знаем, что кратчайший путь от вершины i до вершины j проходит через вершину k, то мы можем обратиться к той же самой процедуре с частями пути от i до k и от k до j. Выход из рекурсии, когда на кратчайшем пути между двумя вершинами не окажется промежуточных вершин.

Приведем фрагмент программы, реализующий алгоритм Флойда и печатающий кратчайшие пути между всеми парами вершин графа.

```
Procedure Way(i,j:integer);
{печатает путь между вершинами i и j}
Begin
 If w[i,j]=0 then Write (' ', j)
 {печатаем только вершину j, чтобы избежать повторов}
 else
  Begin
   Way (i, w [i, j]); Way (w [i, j], j)
  End
End;
Begin
 …{заполняем матрицу смежности}
… {реализуем алгоритм Флойда }
 For k:=1 to nn do
  For i:=1 to nn do
   For j:=1 to nn do
    If a[i,k]+a[k,j]<a[i,j] then
     Begin
      a[i,j]:=a[i,k]+a[k,j];
      w[i,j]:=k
     End;
{выводим кратчайшие пути}
 For i:=1 to nn do
  For j:=1 to nn do Begin
   Write(i);
   If i<>j then Way(i,j);
   Writeln
  End
End.
```

Алгоритм Флойда прост и хорошо запоминается. Но упростить его для определения длины кратчайшего пути между двумя конкретными вершинами невозможно. Для

решения такой задачи используйте алгоритм Дейкстры (п. 13.1.3).

10. Задачи олимпиад

На каждой олимпиаде предлагаются задачи разных уровней сложности. Некоторые задачи решаются большинством участников, а некоторые единицами. Особенностями задач является непривычные для многих формулировки. Задачи звучат в форме сказок, фрагментов известных художественных произведений и т.д. Практически там никогда нет фраз «дан массив чисел» или «отсортируйте массив». Вы должны самостоятельно решить вопросы представления исходных данных стандартными типами языка, перейти от «сказочных» формулировок к математическим моделям и известным алгоритмам.

В будущем Вам придется изучить комбинаторику, теорию графов, динамическое программирование, численные методы. А пока решайте задачи на сайтах ACM.TIMUS.RU, ACM.SGU.RU и CONTEST.SAMARA.RU.

Внимательно читайте условия задачи, сторого соблюдайте форматы входных и выходных данных. Помните, что даже лишний пробел в выходных данных может стать причиной ошибки. Обязательно тестируйте свою программу не только на тестах, приведенных в условиях задачи, но и на собственных тестах, как можно более «вредных». При тестировании используйте текстовые файлы, даже если тестирующая программа не требует явного использования файлов. Так, например, сайты ACM.TIMUS.RU и ACM.SGU.RU не требуют явного использования файлов, а сайт CONTEST.SAMARA.RU пока требует. Но при автоматическом тестировании на сайте файлы, конечно, использоваться будут.

10.1. Задачи с сайта contest.samara.ru

10.1.1. Тортики – 1

Сегодня у Маши Петровой день рождения. Она пригласила в гости своих друзей и подруг - всего G человек. У нее есть K

тортиков разного размера, которые она собирается разрезать так, чтобы все кусочки оказались (по размеру) одинаковыми. Маша уже придумала, как это можно сделать красиво. Осталось только выяснить, достанется ли хотя бы один кусочек какого-либо торта каждому из присутствующих.

Формат входного файла input.txt. Первая строка - два целых числа G и K через пробел (G - количество приглашенных гостей, 0<=G<=100, K - количество тортиков, 1<=K<=20).

Вторая строка - K целых чисел (1<=j1, j2, …, jK<=10) через пробел. Каждое из чисел jP обозначает, на какое количество кусочков Маша собирается разрезать тортик № P

Формат выходного файла output.txt. Первая строка: слово YES, если каждому из присутствующих достанется хотя бы один кусочек какого-либо торта, и слово NO, если это не так

Пример входного файла	Пример входного файла
10 2	5 1
4 7	4
Пример выходного файла	Пример выходного файла
YES	NO

Решение. Задача была предложена участникам окружной олимпиады школьников Самарской области по информатике в 2005 году. Задача очень простая и попытки сдать ее были предприняты на первых минутах соревнований. Но было много неудачных попыток, участники забыли, что присутствующие это не только приглашенные гости, но и сама Маша. Решайте и сдавайте самостоятельно. Помните, что за неудачные попытки, по правилам соревнований могут назначаться штрафные баллы. Найти эту задачу можно на сайте contest.samara.ru в пунктах «Соревнования» – «Окружная олимпиада 17.12.2005».

Посмотрите на мониторе этих соревнований, как решили участники эту и другие задачи. Думаем, что многие задачи вам по силам. Оцените время, которое вы затратите на решения задач, сравните со временем участников

Успехов!

10.1.2. Высокие горы

Автор задачи: Рогачева Е.В.

Снарядил старший сын обоз с подарками богатыми, и отправился за высокие горы за своей невестой. А ехать нелегко - то в гору подниматься, то с горы спускаться. Надо остановки делать, чтобы отдохнуть. Но и добраться поскорее хочется.

Обозначим через E «запас сил» обоза. При подъеме в гору на единичную высоту обоз теряет 2*M «сил», а при спуске с такой же высоты - M «сил». Отдыхать обоз может только в «точках перевала» - либо после спуска перед подъемом, либо после подъема перед спуском. Обоз не может двигаться дальше, если до следующего перевала он должен потратить сил больше, чем имеется у него в запасе. За единицу времени он может восстановить V сил. Однако обоз не может набрать сил больше, чем Emax, сколько бы он ни отдыхал.

Ваша задача - написать программу, определяющую минимальное время отдыха, которое потребуется обозу по пути до места назначения.

Формат входного файла input.txt.

Первая строка - целые числа N, E0, Emax, M, V через пробел.

N - количество точек «перевала» при движении обоза, 2 <= N <= 100. E0 - начальный запас сил обоза (0 <= E0 <= Emax), Emax - максимально возможный «запас сил» обоза (1 <= Emax <= 1000) M - количество сил, которое теряется при спуске с единичной высоты (0 <= M <= 1000), V - количество сил, которое восстанавливается за единицу времени при отдыхе (0 <= V <= 1000).

Вторая строка - целые числа h1, h2, …, hN через пробел - высоты, на которых расположены точки перевала (-1000 <= h1, h2, …, hN <= 1000).

Первое число соответствует точке, из которой обоз выезжает, последнее - месту назначения. Гарантируется, что во входном файле ни разу не встречаются два подряд одинаковых числа.

Формат выходного файла output.txt.

Первая строка - целое число T - минимальное время отдыха, которое потребуется обозу в пути или слово NO, если обоз не может проехать до места назначения с указанными ограничениями.

Пример входного файла	Пример выходного файла
3 13 15 3 2	1

Решение. Только 6 участников городского тура олимпиады 2006 Самары прошли все тесты и получили максимальное число баллов. Многие из тех, кто решил другие даже более сложные задачи, не сумел пройти все тесты для этой задачи.

Анализ показал, что в большинстве решений не учтено одно из условий задачи, а именно - V - количество сил, которое восстанавливается за единицу времени при отдыхе, может быть равно нулю($0 <= V <= 1000$). Это означает, что если энергии для движения недостаточно, то при V=0 отдыхать бесполезно и нужно выдать в выходной файл 'NO'.

Второй особенностью является условие ограничение на Emax, даже при V>0 накопить энергию большую Emax нельзя. Учтите и это в своем решении. На сайте есть разбор задач и авторские решения. Но вы, зная особенности тестов, должны теперь самостоятельно решить и сдать решение.

10.1.3. Задача «На болоте» (алгоритм Дейкстры)

Автор задачи: Рогачева Е.В.

Добрался Иван-царевич до болот, тут они с волком и простились. Пригляделся Иван-царевич, и стрелу свою увидел: торчит она в самой дальней кочке. Ну что ж, делать нечего, придется прыгать с кочки на кочку, чтобы до нее добраться.

Ваша задача - определить наиболее короткий маршрут до стрелы.

Формат входного файла input.txt

Первая строка - целое число N ($1 <= N <= 500$) - количество кочек на болоте.

Далее следуют N строк, описывающие кочки. Строка № j+1 (j = 1, 2, …, N) содержит 0 или более пар, состоящих из целого числа P_k ($1 <= P_k <= N$, $j < k < N$) и вещественного числа Q_k ($0{,}001 <= Q_k <=$

1000, j < k < N) через пробел (пары также разделены пробелами). Целое число P_k (первое в паре) - номер кочки, вещественное число Q_k - расстояние от кочки № j до кочки № P_k. Каждая из N строк завершается парой 0 0.

При этом первая строка описывает кочку, на которую Иван-царевич встает первой, а последняя - кочку, до которой и нужно добраться.

Обратите внимание, что во входном файле указаны расстояния только между такими парами кочек № j и № P_k, в которых j < P_k. Расстояние от кочки № P_k до кочки № j в точности равно расстоянию от кочки № j до кочки № P_k. Гарантируется, что путь от кочки № 1 до кочки № N всегда существует.

Примечание. Поскольку Иван-царевич может прыгнуть не далее определенного расстояния, во входном файле указаны только принципиально достижимые для него кочки, когда он стоит на кочке №j.

Формат выходного файла output.txt

Первая строка - вещественное число S с точностью до 2 знаков после запятой - минимальное расстояние, которое придется преодолеть Ивану-царевичу, добираясь до стрелы.

Пример входного файла	Пример выходного файла.
3 2 4.6 3 5.1 0 0 0 0 0 0	5.10

Приведем решение автора задачи.

```
Const
  NMax = 500;
  QMin = 0.001;
  QMax = 1000;
  WayMax = (NMax*NMax+1)*QMax;
  {это число будет служить заменой "бесконечно большого" в}
{алгоритме  Дейкстры. Разумеется, возможны и другие варианты}
Type
  T_Incidence_Vector = array[1..NMax] of real;
```

```
  T_Incidence_Matrix = array[1..NMax] of T_Incidence_Vector;
  T_Hillok = array[1..NMax] of boolean;
  T_Hillok_Way = array[1..NMax] of real;
Var
  fin, fout: TextFile;
  IM: T_Incidence_Matrix;
  S: real;
  N: integer;
Procedure ReadData;
Var
  P, j: integer;
  Q: real;
Begin
  AssignFile(fin, 'input.txt');  Reset(fin);
  Readln(fin, N);
  {сначала заполним матрицу }
  For j := 1 to N-1 do Begin
    Read(fin, P, Q);
    While not((P = 0) and (Q < QMin)) do
    Begin
      IM[j, P] := Q;
      IM[P, j] := Q; {так как граф неориентированный,}
                     {и матрица симметрична}
      Read(fin, P, Q);
    End;
    Readln(fin);
  End;
{последнюю строчку на всякий случай обработаем}
{отдельно - хотя в ней, если}
{следовать условию задачи, должно быть только 0 0}
  Read(fin, P, Q);
  While not((P = 0) and (Q < QMin)) do Begin
    IM[N, P] := Q;
    IM[P, N] := Q;
    read(fin, P, Q);
```

```
  End;
  CloseFile(fin);
End;
Function min(a, b: real): real;
Begin
  If a < b then result := a
  else result := b;
End;
Function MinInHillokWay(const h: T_Hillok; const hw: T_Hillok_Way;
                        out minNum: integer): real;
Var
  i, j: integer;
Begin
{ищем первую кочку вне множества; кочка № 1 в}
{него заведомо входит.}
{проверка того, что хотя бы одна кочка еще}
{есть, здесь не проводится - это задача основного цикла}
{процедуры Solution}
  i := 2;
  While h[i] do i := i+1;
  {найденную принимаем за "потенциальный минимум"}
  result := hw[i];
  minNum := i;
  {а теперь просматриваем оставшиеся кочки и, если найдется что-
то более  подходящее, обновляем значение минимума}
  For j := i+1 to N do Begin
    If (not(h[j])and (hw[j] < result))
    then Begin
      result := hw[j];
      minNum := j;
    End;
  End;
End;
Procedure Solution;
Var
```

```
  h: T_Hillok; {множество кочек, которые уже}
                  { используются в маршрутах}
  hw: T_Hillok_Way; {уже имеющиеся длины}
                       { маршрутов от первой до других кочек}
  i, k: integer;
  add_hillok: integer;
  to_hillok: real;
Begin
  For i := 1 to N do h[i] := false;
  h[1] := true;
  For i := 1 to N do hw[i] := WayMax;
  hw[1] := 0.0;
  {первую итерацию алгоритма Дейкстры выполним отдельно}
  For i := 2 to N do Begin
    {если до кочки существует прямой путь из первой}
    If (IM[1, i] > 0) then hw[i] := IM[1, i];
  End;
{а теперь анализируем кочки до тех пор,}
{пока множество не рассмотренных не опустеет}
{Поскольку может оказаться, что просмотреть надо все}
{вершины, можно просто пустить цикл от 1 до N}
  For k := 1 to N do Begin
    {находим кочку, которая ближе всего к первой }
    {кочке, но еще не входит в множество h}
    to_hillok := MinInHillokWay(h, hw, add_hillok);
    {и добавляем ее к множеству h}
    h[add_hillok] := true;
    If add_hillok = N {если встретили кочку N, }
    then break;          {дальше можно не считать}
{теперь расстояния до остальных надо пересчитать
    {с учетом этой кочки как промежуточной}
    For i := 1 to N do Begin
      {если кочка еще не входит в множество уже рассмотренных и
       до нее существует прямой путь из вновь добавленной}
      If (not(h[i]) and (IM[add_hillok, i] > 0))
```

```
      then hw[i] := min(hw[i], to_hillok+IM[add_hillok, i])
    { to_hillok == hw[add_hillok], просто чтобы не}
  {обращаться по многу раз к массиву }
   End; {for i}
 End; {while}
 {теперь осталось просто забрать значение из массива hw}
 S:= hw[N];
End; {Solution}
Procedure WriteData;
Begin
 AssignFile (fout, 'output.txt'); Rewrite (fout);
 Write (fout, S:0:2);
 CloseFile (fout);
End;
Begin
 ReadData;
 Solution;
 WriteData;
End.
```

10.1.4. Задача «На болоте» (алгоритм Флойда)

Так как жестких временных ограничений не было, то многие участники решили эту задачу методом Флойда. Повторимся, что этот метод прост и легок в запоминании, но будет искать кратчайшие пути между всеми вершинами. Приведем программу Антона Александрова (Самарский лицей информационных технологий). Программа написана в консольном приложении Delphi.

```
program Project1;
{$APPTYPE CONSOLE}
uses SysUtils;
var
  i, n, t, k, j: integer;
  ar: array [1 .. 500, 1 .. 500] of real;
Begin
```

```
{ TODO -oUser -cConsole Main : Insert code here }
Assignfile (input, 'input.txt'); Reset (input);
Assignfile (output, 'output.txt'); Rewrite (output);
Readln (n);
For i:= 1 to n do
  For t:= 1 to n do
    ar [i, t]:= high (integer); {макс. целое}
For i:= 1 to n do Begin
  Repeat
    Read (t);
    If t > 0 then Begin
      Read (ar [i, t]);
      ar [t, i]:= ar [i, t];
    End;
  Until t = 0;
  Readln;
End;
For k:= 1 to n do Begin
  For i:= 1 to n - 1 do
    For j:= i + 1 to n do
      If ar [i, j] > ar [i, k] + ar [k, j] then
      Begin
        ar [i, j]:= ar [i, k] + ar [k, j];
        ar [j, i]:= ar [i, j];
      End;
End;
Write (ar [1, n]: 0: 2);
Closefile (input); Closefile (output);
End.
```

10.2. Задачи с сайта ACM.TIMUS.RU

Зарегистрируйтесь на сайте и запишете выделенный вам код. При работе с сайтом текстовые файлы не описываются и не назначаются. Нельзя подключать модули (даже привычный Uses Crt;) использовать операторы вида Write(‘Введите число’). Следует

убирать и поставленный во время отладки в конце программы оператор Readln.

Сдайте задачу «A+B» (номер на сайте 1000)-там все числа целые. Достаточно написать:

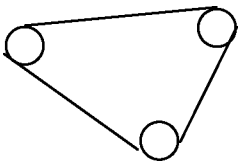
```
Var a, b: integer;
Begin
  Read(a, b);
  Write(a+b);
End.
```

Успехов!

10.2.1. Задача «Ниточка» (номер на сайте 1020)

Авторы задачи: Александр Петров и Никита Шамгунов

Злоумышленники варварски вбили в ни в чем не повинную плоскую поверхность N гвоздей, да так, что только шляпки остались. Мало того, они в своих подлых целях вбили все гвозди в вершины выпуклого многоугольника. После этого они... страшно сказать... они натянули ниточку вокруг всех гвоздей, так, что поверхности стало совсем больно! Вот как примерно они это сделали:



Определить длину этой ниточки.

Исходные данные. В первой строке входа к этой задаче находятся два числа — количество гвоздей N ($1 \le N \le 100$), и вещественное число R — радиус шляпок гвоздей. Все шляпки имеют одинаковый радиус. Далее на входе располагаются еще N строк, в каждой из которых записана через пробел пара

вещественных координат центра очередного гвоздя; координаты не превосходят по абсолютной величине числа 100.

Описания гвоздей приводятся в порядке обхода вершин многоугольника (либо по часовой стрелке, либо против часовой стрелки), начиная с произвольного. Шляпки разных гвоздей не соприкасаются.

Результат.

Вещественное число, указанное ровно с двумя верными знаками после запятой - длину ниточки, натянутой вокруг всех гвоздей.

Пример

Исходные данные	**Результат**
4 1 0.0 0.0 2.0 0.0 2.0 2.0 0.0 2.0	14.28

Решение. Длина ниточки это сумма расстояний между центрами гвоздей плюс сумма дуг прилегания нити к шляпкам гвоздей. Вся хитрость задачи именно во второй сумме. Посмотрите на рис.5

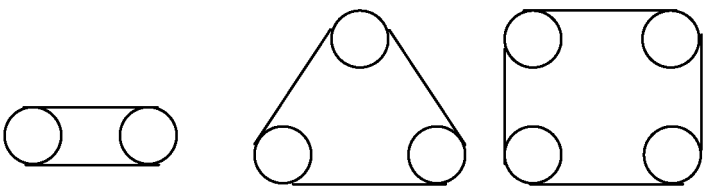


Рис.5. Примеры размещения гвоздей

Вы по рисунку уже, наверное, поняли, что сумма всех дуг прилегания составит длину одной полной окружности. Напомним, что по условию задачи, гвозди вбиты в вершины выпуклого многоугольника. В этом случае наш интуитивный вывод о сумме дуг можно доказать. Расстояния между точками на плоскости (координаты центров гвоздей) вы можете определить по известным формулам. Для выполнения требования по формату результата

используем оператор вывода Write(S:0:2). Очевидно, что координаты точек и все длины – вещественные числа.

Вам по силам написать эту программу самостоятельно. Запустите приведенный в задаче тест. Сдайте задачу тестирующей системе сайта.

Надеемся, что у вас все получилось!

10.2.2. Демократия в опасности (номер на сайте 1025)

Автор задачи: Леонид Волков

В одном из островных государств Карибского бассейна все решения традиционно принимались простым большинством голосов на общем собрании граждан, которых, к счастью, было не очень много. Одна из местных партий, стремясь прийти к власти как можно более законным путем, смогла добиться некоторой реформы избирательной системы. Главным аргументом было то, что население острова в последнее время значительно возросло, и проведение общих собраний перестало быть легкой задачей. Суть реформы состояла в следующем: с момента введения ее в действие все избиратели острова делились на K групп (необязательно равных по численности). Голосование по любому вопросу теперь следовало проводить отдельно в каждой группе, причем считалось, что группа высказывается «за», если «за» голосует более половины людей в этой группе, а в противном случае считалось, что группа высказывается «против». После проведения голосования в группах подсчитывалось количество групп, высказавшихся «за» и «против», и решение вопрос решался положительно в том и только том случае, когда групп, высказавшихся «за», оказывалось более половины общего количества групп. Эта система вначале была с радостью принята жителями острова. Когда первые восторги рассеялись, стали очевидны, однако, некоторые недостатки новой системы. Оказалось, что сторонники партии, предложившей систему, смогли оказать некоторое влияние на формирование групп избирателей. Благодаря этому, они получили возможность проводить некоторые решения, не обладая при этом реальным большинством голосов!

Пусть, например, на острове были сформированы три группы избирателей, численностью в 5, 5 и 7 человек соответственно. Тогда партии достаточно иметь трех сторонников в каждой из первых двух групп, и она сможет провести решение всего 6-ю голосами «за», вместо 9-и, необходимых при общем голосовании.

Вам надо написать программу, которая определяет по заданному разбиению избирателей на группы минимальное количество сторонников партии, достаточное, при некотором распределении их по группам, для принятия любого решения.

Исходные данные.

Вход для этой задачи состоит из двух строк. В первой строке файла записано натуральное число K ≤ 101 — количество групп избирателей. Во второй строке через пробел записаны K натуральных чисел, которые задают количество избирателей в группах. Для упрощения определения понятия "большинство голосов" будем полагать, что и число групп, и количество избирателей в каждой группе суть нечетные числа. Вы можете также считать, что население острова не превосходит 10001 человек.

Результат.

На выход следует записать натуральное число - минимальное количество сторонников партии, способное решить исход голосования.

Пример

Исходные данные	Результат
3 5 7 5	6

Решение. Понятно, что выгодно побеждать в маленьких партиях. Отсортируем партии по возрастанию числа членов. Учитывая, что число партий и число членов в каждой партии нечетные числа, должно победить n div 2 +1 партий. В каждой партии для победы необходимо a[i] div 2+1 голосов.

Текст программы:

```
Var a:array[1..101] of Integer;
    n, i, j, s, q: Integer;
```

```
Begin
 Read(n);{вводим число партий}
 {вводим число членов каждой партии}
 For i:=1 to n do read(a[i]);
 {сортируем 'пузырком'}
 For j:=1 to n-1 do
  For i:=1 to n-j do
   If a[i]>a[i+1] then Begin
      q:=a[i];
      a[i]:=a[i+1];
     a[i+1]:=q;
   End;
 S:=0;
 For i:=1 to n div 2 +1 do
  s:=s+(a[i] div 2 +1);
 Write(s);
End.
```

10.2.3. Один в поле воин (номер на сайте 1197)

Условие этой задачи очень простое: вам всего лишь надо определить, сколько клеток находится под боем шахматного коня, одиноко стоящего на шахматной доске. На всякий случай напомним, что конь ходит буквой «Г» — на две клетки по горизонтали или вертикали в любом направлении, и потом на одну клетку в направлении, перпендикулярном первоначальному.

Исходные данные. В первой строке находится число N ($1 \le N \le 100$) количество тестов в файле. В каждой из последующих N строк содержится очередной тест: два символа (маленькая латинская буква от 'a' до 'h' и цифра от 1 до 8) — стандартное шахматное обозначение клетки, на которой стоит конь.

Результат. Выходной файл должен состоять из N строк, в каждой из них должно находиться единственное число - количество клеток шахматной доски, находящихся под боем коня.

Пример.

Исходные данные	Результат
3 a1 d4 g6	2 8 6

Ниже приведено оригинальное решение Павла Семушина (Самарский лицей информационных технологий).

В программе используется стандартная функция ord, возвращающая код символа. Оператор x:=1+ord(c)-ord('a') дает положение коня по горизонтали (например, если c='a', то x=1), а оператор y:=1+ord(c)-ord('1') по вертикали. В массивах dx и dy закодированы все восемь ходов коня из некоторой точки. Например, dx[1]=1 dy[1]=2 это ход на одну клетку вправо и на две вверх, а dx[4]=1 dy[4]=-2 ход на одну вправо и на две вниз. Те ходы, которые дают координаты от 1 до 8 (битая клетка находится внутри доски), дают нам битые поля.

```
Const
  dx: array[1..8] of integer=(1, 2, 2, 1,-1,-2,-2,-1);
  dy: array[1..8] of integer=(2, 1,-1,-2,-2,-1, 1, 2);
Var
  i, j, x, y, n, s: integer;
  c: char;
Begin
  Readln (n);
  For i:=1 to n do Begin
     Read (c);
     x:=1+ord(c)-ord('a');
     Readln (c);
     y:=1+ord(c)-ord('1');
     s:=0;
     For j:=1 to 8 do
       If ((x+dx[j]) in [1..8]) and ((y+dy[j]) in [1..8])
       Then  s:=s+1;
     Writeln (s);
   End;
```

End.

10.2.4. Задача «Выборы» (номер на сайте 1263)

Автор задачи: Ден Расковалов.

Грядут очередные выборы. Снова все заборы оклеены листовками, почтовые ящики забиты макулатурой, с экранов телевизоров депутаты обещают сделать нашу жизнь лучше. А программист Васечкин снова завален работой. Необходимо написать программу, которая облегчит подсчет голосов избирателей.

Исходные данные. В первой строчке ввода находятся натуральные числа N – число кандидатов (N<=10000) и M – число избирателей, принявших участие в выборах (M<=10000). Далее следует M строк, в каждой из которых находится номер кандидата, за которого проголосовал избиратель.

Результат. На выходе должно содержаться N строк, в i-ой строке должен быть указан процент избирателей, проголосовавших за i-ого кандидата (с точностью до 2-х знаков после запятой).

Пример.

Исходные данные	Результат
3 6	
1	
2	50.00%
3	33.33%
2	16.67%
1	
1	

Решение. Организуем только один массив - массив счетчиков, каждый элемент которого соответствует числу голосов, отданных за соответствующего кандидата (номер кандидата, за которого проголосовал избиратель это номер элемента массива счетчиков). Соблюдайте правила ввода и вывода, указанные в условии задачи и решайте задачу самостоятельно. Наше решение приведено ниже.

```
Var  a: array [1..10000] of Integer;
     i, m, n, k: Integer;
```

```
Begin
{вводим число кандидатов и число избирателей}
 Readln(n, m);
{обнуляем массив счетчиков}
 For i:=1 to n do a[k]:=0;
 For i:=1 to m do Begin
  Readln(k); {вводим k-номер депутата}
  a[k]:=a[k] + 1; {или inc(a[k])}
 End;
 For i:=1 to n do
  Writeln(a[i]/m*100:0:2, '%')
End.
```

10.2.5. Белый тезис (номер на сайте 1335)

Автор задачи Ден Расковалов.

Тут принес ключи бакалавр черной магии Магнус Федорович Редькин, толстый, как всегда озабоченный и разобиженный. Бакалавра он получил триста лет назад за изобретение портков-невидимок. С тех пор он эти портки все совершенствовал и совершенствовал. Портки-невидимки превратились у него сначала в кюлоты-невидимки, потом в штаны-невидимки, и, наконец, совсем недавно о них стали говорить как о брюках-невидимках. И никак он не мог их отладить. На последнем заседании семинара по черной магии, когда он делал очередной доклад "О некоторых новых свойствах брюк-невидимок Редькина", его опять постигла неудача. Во время демонстрации модернизированной модели что-то там заело, и брюки, вместо того чтобы сделать невидимым изобретателя, вдруг со звонким щелчком сделались невидимыми сами. Очень неловко получилось.

Однако Магнус Федорович главным образом работал над диссертацией, тема которой звучала так: "Материализация и линейная натурализация Белого Тезиса, как аргумента достаточно произвольной функции Е не вполне представимого человеческого счастья".

Тут он достиг значительных и важных результатов, из коих следовало, что человечество буквально купалось бы в не вполне представимом счастье, если бы только удалось найти сам Белый Тезис, а главное - понять, что это такое и где его искать.

Согласно последней гипотезе Редькина Белый Тезис представляет собой тройку натуральных чисел (A, B, C), обладающую свойством, что A^2+B^2 делится нацело на C, причем искать его нужно между квадратами двух последовательных натуральных чисел N и N + 1.

Исходные данные содержат единственное целое число N (2 <= N <= 30000).

Результат: три различных числа A, B, C такие, что A^2+B^2 делится нацело на C и N^2 <= A, B, C <= $(N + 1)^2$. Если существует более одной такой тройки, выведите любую. Если такой тройки не существует, выходной поток должен содержать единственную строку "No solution" (без кавычек, естественно).

Примеры

Исходные данные	Результат
2	8 6 4
1000	1000000 1000756 1000976

Решение. Нужно забыть про Магнуса Федоровича и про его штаны-невидимки, важна математическая сторона задачи. По условию задачи N^2 <= A, B, C <= $(N + 1)^2$. Раскроем скобки в правой части и получим N^2 <= A, B, C <= $N^2 + 2N + 1$.

Положим A= $N^2 + 2N$, B= $N^2 + N$ и C= N^2, значения соответствуют данному неравенству и условию делимости.

При самостоятельном решении не забудьте про пробелы между числами в ответе. Нужно писать Write (A, ' ', B, ' ', C);

10.2.6. Проблема Бен Бецалеля (номер на сайте 1336)

Автор задачи: Ден Расковалов.

- Г-голубчики, - сказал Федор Симеонович озадаченно, разобравшись в почерках. Это же п-проблема Бен Б-бецалеля. К-калиостро же доказал, что она н-не имеет р-решения.

- Мы сами знаем, что она не имеет решения, - сказал Хунта, немедленно ощетиниваясь. - Мы хотим знать, как ее решать.

- К-как-то ты странно рассуждаешь, К-кристо... К-как же искать решение, к-когда его нет? Б-бессмыслица какая-то...

- Извини, Теодор, но это ты очень странно рассуждаешь. Бессмыслица - искать решение, если оно и так есть. Речь идет о том, как поступать с задачей, которая решения не имеет. Это глубоко принципиальный вопрос, который, как я вижу, тебе, прикладнику, к сожалению, не доступен. По-видимому, я напрасно начал с тобой беседовать на эту тему.

Задачи, которые не имеют решения, - это, конечно, здорово. Но иногда хочется порешать что-то, в существовании решения, которого никто не сомневается. Например, представить натуральное число в виде отношения квадрата какого-то натурального числа и куба. Только почему эта задача всегда имеет решение? Ну ладно, разберетесь!

Исходные данные.

На входе содержится натуральное число n ($1 <= n <= 10^9$).

Результат.

В первой строчке выходного потока должно содержаться число m. Во второй – число k. Причем, m^2 должно нацело делиться на k^3, и $m^2/k^3=n$, $1 <= m, k <= 10^{100}$.

Примеры

Исходные данные	Результат
18	12 2
1	1 1

Очевидно, что если в числители будет n^4, а в знаменателе n^3, то задача будет решена. Тогда $m=n^2$ и $k=n$. Обратите внимание, что в ответе должно быть две строки и в программе будет фрагмент:

```
Writeln(m);{выдать и перейти на новую строчку}
Write(k);
```

Решение. Для n подойдет тип Longint, но перполнение возникнет уже при вычислении выражения n*n. Для компилятора

TP 7.0 используем тип Comp. При работе с TP в меню Options – Compiler в Numeric processing нужно отметить поле 8087/80287, после чего программа должна успешно запуститься.

Программа будет иметь вид:

```
Var k, m, n:comp;
Begin
  Readln(n);
  m:=n*n;
  k:=n;
  Writeln(m:0:0);
  Write(k:0:0);
End.
```

Сайт, на котором вы будете сдавать эту задачу использует компилятор Delphi. Поэтому программа может быть такой:

```
Var k, m, n: Int64; {в TP такого типа нет}
Begin
  Readln(n);
  m:=n*n;
  k:=n;
  Writeln(m); Write(k);
End.
```

10.2.7. Ферма (номер на сайте 1349)

Автор задачи Пьер Ферма.

Стоит себе ферма. На ферме сидит фермер и считает, сколько разного скота есть у него на ферме - a верблюдов, b баранов, c зеленых тараканов. Почему-то $a^n + b^n = c^n$. Известно n. Найти все остальное.

Исходные данные n ($0 \le n \le 100$)

Результат. Три различных натуральных числа (a, b, c) таких, что $a^n + b^n = c^n$, $0 < a, b, c < 101$. Если решений несколько, вывести то, где a минимально. Если и таких чисел несколько, вывести то, где минимально b и т.д. Вывести -1, если решения нет.

Примеры

Исходные данные	Результат

0	-1
1	1 2 3

Решение. Понятно, что история про фермера и его скот приведены здесь, чтобы отвлечь вас от математической сути задачи. Великая теорема Ферма (ее также называют «Большой теоремой Ферма») состоит в утверждении, что при значениях $n > 2$ уравнения вида $x^n + y^n = z^n$ не имеют ненулевых решений в натуральных числах.

В решении просто нужно проверить n и для n=1 выдать числа 1, 2, 3, для n=2 числа 3, 4, 5, а для всех остальных –1. И все!

10.2.8. Развод семи гномов (номер на сайте 1243)

Автор задачи: Станислав Васильев

Все мы знаем, чем закончилась история про Белоснежку и семь гномов - Белоснежка уехала с женихом, бросив всех тех, кто бескорыстно помог ей в трудную пору. После её отъезда гномы стали ссориться - каждый считал, что это другие чем-то обидели Белоснежку.

Чтобы не доводить вечные ссоры до кровопролития, некогда дружные гномы решили расстаться, поделив все свое добро, от кружек до алмазов, согласно старинным гномьим законам о разводе. По этим законам, все имущество должно быть самым справедливым образом поделено между гномами, а то, что поделить поровну нельзя, не должно достаться никому из них. Бережливые гномы решили, что неразделенные вещи выкидывать не будут, а отдадут Белоснежке в качестве приданого.

Например, у каждого из гномов с рождения имелось по две пары ботинок, к моменту появления этой задачи самый старый гном одну свою пару износил, значит, после справедливого раздела оставшихся 26 ботинок, каждый гном получит по 3 ботинка, а Белоснежку осчастливят 5 ботинками.

Отметим, что некоторых вещей у гномов очень много - одних только маковых зернышек накопилось 123456123456 штук. Гномы потратили немало времени, пока сосчитали, что Белоснежка

получит всего одно маковое зернышко. Ваша задача помочь гномам рассчитать долю Белоснежки.

Исходные данные. Единственная строка входного файла содержит число N одинаковых вещей, которые хотят поделить гномы ($1<=N<=10^{50}$).

Результат. Единственное число - количество вещей, которые, в результате справедливого раздела, перейдут к Белоснежке.

Пример

Исходные данные	Результат
123456123456	1

Решение. Такие длинные числа не вписываются ни в один из стандартных типов языков программирования. Поэтому прямо вычислить остаток от деления такого числа на 7 не удастся. И как всегда на помощь приходит знание математики. Нас интересует только остаток от деления, а не результат деления.

Алгоритм: берем старшую цифру числа и делим ее на 7, остаток от деления умножаем на 10 и добавляем следующую цифру, и так до конца массива (до самой младшей цифры числа). Последний остаток и будет результатом. Решайте самостоятельно. Можно организовать посимвольное чтение, преобразование символа в число и сразу обработку, можно ввести строку (символьный массив) и обрабатывать ее элементы. Успеха!

10.2.9. Освещение в Хогвартсе (номер на сайте 1448)

Авторы задачи: Александр Мироненко, Станислав Васильев

Коридоры Хогвартса хорошо освещаются: вдоль каждого коридора идет ряд волшебных светильников. Каждый такой светильник может либо светить на полную мощность, либо не светить совсем. Светильников очень много, они расположены через небольшие равные промежутки, поэтому, включая и выключая часть из них, легко регулировать освещенность коридора. Аргус Филч не любит, когда освещение получается неравномерным - в тёмных местах могут прятаться нарушители порядка. Ваша задача - сделать максимально равномерное освещение заданной яркости.

Исходные данные. В первой строке входа находится число N ($1 \le N \le 106$) — количество светильников в коридоре. Во второй строке находится требуемая Аргусу яркость - целое число b ($0 \le b \le 100$). Яркость указана в процентах от максимальной яркости (0 - когда все магические светильники выключены, 100 - когда все включены).

Результат. Вывести такую последовательность из N нулей и единиц (1 обозначает включенный светильник, 0 - выключенный), чтобы для любого отрезка коридора количество включенных светильников отличалось от числа L*b/100 не более чем на 2 (здесь L - общее количество светильников на данном отрезке коридора).

Пример

Исходные данные	Результат
10 33	0100100100

Решение даем без пояснений.

Автор программы Павел Семушин (Самарский лицей информационных технологий).

```
Var   i, j, k, l, m, n: integer;
Begin
  Read(n,m);
  k:=0;
  For i:=1 to n do Begin
    If (i=1) or (k/i*100<m) then
      Begin
       k:=k+1;
       Write('1');
      End
    else Write('0');
  End;
End.
```

10.2.10. Гиперпереход (номер на сайте 1296)

Автор задачи: Ден Расковалов

Гиперпереход, открытый еще в начале XXI-го века, и сейчас остается основным способом перемещения на расстояния до сотен тысяч парсеков. Но совсем недавно физиками открыто новое явление. Оказывается, длительностью альфа-фазы перехода можно легко управлять. Корабль, находящийся в альфа-фазе перехода, накапливает гравитационный потенциал. Чем больше накопленный гравитационный потенциал корабля, тем меньше энергии потребуется ему на прыжок сквозь пространство. Ваша цель – написать программу, которая позволит кораблю за счет выбора времени начала альфа-фазы и ее длительности накопить максимальный гравитационный потенциал.

В самой грубой модели грави-интенсивность – это последовательность целых чисел p_i ($1 <= i <= n$). Будем считать, что, если альфа-фаза началась в момент i, и закончилась в момент j, то накопленный в течении альфа-фазы потенциал – сумма всех чисел, стоящих в последовательности на местах от i до j.

Исходные данные. В первой строчке входа записано число n – число чисел в последовательности, отвечающей за грави-интенсивность ($n<60000$). Далее идут n строк, в каждой записано целое число p_i ($-30000 <= p_i <= 30000$). Первая строчка содержит p_1, вторая - p_2 и т.д.

Результат. Максимальный гравитационный потенциал, который может накопить корабль в альфа-фазе прыжка. Считается, что потенциал корабля в начальный момент времени равен нулю.

Примеры.

Исходные данные	Результат
10 31 -41 59 26 -53 58 97 -93	187

-23 84	
3 -1 -5 -6	0

Решение. Если вам понятны результаты приведенных тестов, то и алгоритм должен быть понятен. Последовательно считываем и суммируем числа. Отслеживаем максимальную сумму. Если сумма становится меньше нуля, обнуляем ее и продолжаем суммирование. Массив, естественно, не нужен. Решайте самостоятельно и сдавайте. Успеха!

10.2.11. Драгоценные камни (Stone pile 1005).

You have a number of stones with known weights W1…Wn. Write a program that will rearrange the stones into two piles such that weight difference between the piles is minimal.

Исходные данные.

Input contains the number of stones N ($1 \le N \le 20$) and weights of the stones W1…Wn ($1 \le Wi \le 100000$) delimited by white space (either space characters or new lines).

Результат.

Your program should output a number representing the minimal possible weight difference between stone piles.

Вольный перевод: нужно поделить камни на две кучи с минимальной разницей весов.

Идея алгоритма, который часто предлагают школьники: считываем числа в массив, сортируем его, по одному большому камню кладем в кучи. Просматриваем остальные камни и кладем очередной камень в ту кучу, в которой в этот момент вес камней меньше. Пишем программу, тестируем.

```
Var a: array[1..20] of integer;
   n,i,j,k:integer;
Begin
  Readln(n);
```

```
  For i:=1 to n do readln(a[i]);
  For i:=n-1 downto 1 do
    For j:=1 to i do
      If a[j]>a[j+1] then Begin
        k:=a[j];
        a[j]:=a[j+1];
        a[j+1]:=k;
      End;
  j:=0;
  k:=0;
  For i:=n downto 1 do
    If j<k then j:=j+a[i] else k:=k+a[i];
  Write(abs(j-k));
  Readln
End.
```

Протестируйте программу, меняя число и веса камней. Все ваши тесты проходят?

Придумайте тесты, которые «не пройдут». Не придумали?

Вот один из них: 5 камней (8, 7, 6, 4, 3). Результат программы 2. Суммы 8+4+3=15 и 7+6=13, разница 2. А правильный результат: 8+6=14 и 7+4+3=14, разница и соответственно ответ 0.

Ниже приведено решение Павла Семушина (Самарский лицей информационных технологий) методом динамического программирования. Возможно только с целыми весами камней. Для вещественных весов пришлось бы делать полный перебор. Размер вспомогательного массива определяется значениями весов камней (современные компиляторы допускают такие размеры массивов).

```
Var
  a: array[1..20] of integer;
  b: array[0..100000] of integer;
{допустимый размер для компилятора на сайте}
  n, i , j, k, s: integer;
Begin
  Readln(n);
```

```
s:=0;
For i:=1 to n do Begin
   Readln(a[i]);
   s:=s+a[i];
End;
k:=s div 2;
For i:=k downto 0 do
  If i>=a[1] then b[i]:=a[1] else b[i]:=0;
For i:=2 to n do
  For j:=k downto 1 do
     If (j>=a[i]) and (a[i]+b[j-a[i]]>b[j]) then b[j]:=a[i]+b[j-a[i]];
Write(abs(s-2*b[k]));
End.
```

Протестируйте эту программу. Теперь даже ваши самые «трудные» тесты пройдут!

11. Процедуры и функции.

11.1. Как написать хорошую программу.

Пока вы дочитали до этого раздела, вам встретились и процедуры и функции, в том числе и рекурсивные. Вам пришлось заглядывать в рекомендованную литературу? Или было все понятно?

Не приводя подробного описания процедур и функций, обращаем ваше внимание на следующие моменты:

- Процедура или функция определяет некоторый законченный блок действий;
- Каждая процедура или функция должна иметь одну точку входа и одну точку выхода;
- Взаимодействие с вызывающей программой должно, по возможности, осуществляться только через параметры.
- Создание таких блоков существенно облегчает задачу программирования и отладки, вы можете каждый из них проверить отдельно, а затем соединить их в правильной последовательности.

11.2. Рекурсивные процедуры

Рекурсивная форма организации алгоритма обычно выглядит изящнее итерационной и дает более короткий текст программы, но при выполнении может вызвать переполнение стека. Подробное описание и много замечательных примеров вы найдете в работе[2]. Ограничимся несколькими полезными примерами [2].

11.2.1. n- ая степень числа

```
Function p(x, n:integer): longint;
Begin
  If n=0 then p:=1 else  p:=x*p (x, n-1)+1;
End;
Begin
  Writeln (p (2, 8));
End.
```

11.2.2. Перевод десятичного числа в двоичную систему

Заменив делитель 2 на другой (<10), получим перевод в любую другую систему счисления с основанием меньшим 10.

```
Procedure Rec (n: integer);
Begin
   If n > 1 then rec (n div 2);
   Write (n mod 2);
End;
Begin
  Rec (25);
End.
```

11.2.3. n-ое число Фибоначчи

Числа Фибоначчи 1, 1, 2, 3, 5, 8, 13, 21,…Первые два числа 1, 1. Каждое следующее число равно сумме двух предыдущих.

```
Function f (n:integer):integer;
Begin
  If n<=2 then f:=1
  else f:=f(n-1) + f(n-2);
End;
```

```
Begin
  Write(f(11));
  Readln
End.
```

11.2.4. Алгоритм Евклида (наибольший общий делитель)

```
Function Euclid (m, n: integer): longint;
Var d: longint;
Begin
  If m>n then d:= Euclid(m-n, n)
  else  IF  m<n then d:= Euclid(m, n-m)
        else d:=m;
  Euclid:=d;
End;
Begin
  Write (Euclid (30, 75));
  Readln
End.
```

Список рекомендуемой литературы

1. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. - М.: «Нолидж», 2002. 576 с.
2. Окулов С.М. Основы программирования. - М.: БИНОМ. Лаборатория знаний, 2002.424 с.
3. Окулов С.М. Программирование в алгоритмах. - М.:БИНОМ. Лаборатория знаний, 2002. 341 с.
4. Иванов Б.Н. Дискретная математика. Алгоритмы и программы. Учебное пособие. - М. «Лаборатория базовых знаний», 2001. 288 с.
5. Андреева Е.В., Егоров Ю.Е. Вычислительная геометрия на плоскости. - М. Информатика №39-44, 2002.