Департамент образования администрации города Липецка
МАОУ ДОД Центр дополнительного образования детей
«Стратегия»

Информатика 7-8

Учебно-методическое пособие

Автор-составитель И.А. Шуйкова

Липецк – 2012

УДК 373.167
ББК 32.81я72
И 74

И 74 Информатика 7-8: Учебно-методическое пособие / Авт.-сост. И.А. Шуйкова. – Липецк: Типография «Липецк-Плюс», 2012. – 102 с.

Пособие предназначено для аудиторной и самостоятельной работы школьников, изучающих алгоритмы и основы программирования. Пособие ориентировано на подготовку обучающихся 7-8 классов к школьному и муниципальному этапам Всероссийской олимпиады школьников по информатике.

УДК 373.167
ББК 32.81я72

© МАОУ ДОД Центр дополнительного образования детей «Стратегия», 2012 г.
© И.А. Шуйкова, 2012 г.

СОДЕРЖАНИЕ

Введение 4

Модуль 1. Формализация модели задачи. Работа с файлами 5

Модуль 2. Алгоритмы разветвляющейся структуры. Принадлежность точки фигуре 14

Модуль 3. Алгоритмы циклической структуры. Простые числа 27

Модуль 4. Алгоритмы обработки массивов 36

Модуль 5. Анализ заданий муниципального этапа ВОШ по информатике 43

Модуль 6. Строковый тип данных. Обработка строк 57

Модуль 7. Календарные задачи 66

Модуль 8. Длинная арифметика 70

Модуль 9. Рекурсивный подход в программировании 77

Модуль 10. Геометрические задачи 83

Модуль 11. Динамическое программирование 90

Модуль 12. Задачи регионального этапа Всероссийской олимпиады школьников по информатике 2012 г. 98

Рекомендуемая литература 105

ВВЕДЕНИЕ

Настоящее пособие составлено в соответствии с программой дистанционного обучения очно-заочной школы "Одаренный ребенок", которая ежегодно проходит в липецкой области для школьников, желающих самостоятельно подготовиться к школьным и муниципальным олимпиадам по различным дисциплинам.

Олимпиады по информатике в настоящее время довольно популярны среди школьников, но, несмотря на то, что олимпиадное движение по информатике развивается, масштаб вовлечения школьников в это движение недостаточно велик. Это связано и с тем, что олимпиадное движение по информатике возникло относительно недавно в отличие от олимпиад по другим дисциплинам, и с малым количеством часов, отводимым учебными планами на информатику в школах, и с выбором школьниками олимпиад по другим базовым, смежным с информатикой предметам. В сети Интернет можно найти достаточный перечень книг по программированию, сайты для школьников с набором задач и возможностью их тестирования. Некоторые источники и сайты приведены в конце этого пособия. И в то же время, для школьников 7-8 классов важен системный модульный подход к изучению алгоритмов, на основе которого и строится данное пособие.

Представленный материал предназначен, прежде всего, для школьников, стремящимся освоить сложный и интересный мир программирования. Пособие будет полезно и учителям, работающим с высокомотивированными учащимися.

Модуль 1. Формализация модели задачи. Работа с файлами

Задачи модуля

1. Знакомство с олимпиадными задачами. Формализация модели задачи.
2. Тренировка навыков работы с файлами при написании программ.
3. Рассмотрение примеров считывания информации из файла.
4. Развитие интереса к олимпиадному движению.

1. Теоретический материал

1.1. Примеры олимпиадных задач. Формализация модели задачи

Олимпиадная задача по программированию содержит:

- текст задачи;
- описание формата входных данных;
- описание формата выходных данных;
- примеры входных и выходных данных.

Рассмотрим задачу «Количество чисел», которая входила в состав задач Турнира Архимеда 2011 года.

Задача. Количество чисел

Сколько всего натуральных чисел состоят из не менее чем *a* цифр и не более, чем *b* цифр?

Входные данные

Вводятся два произвольных натуральных числа *a* и *b* через пробел. Каждое не превышает **10 000.**

Выходные данные

Выведите одно число: количество чисел, обладающих указанным свойством.

Примеры

INPUT.TXT	OUTPUT.TXT
1 2	99
1 1	9

Составим модель задачи

Входные данные: *a, b* – натуральные числа, не превышающие 10 000 (тип данных integer).

Выходные данные: количество натуральных чисел, состоящих из не менее, чем *a* цифр и не более, чем *b* цифр.

Решение задачи

Перед решением задачи необходимо вспомнить все математические определения и термины, которые используются в тексте задачи. Напомним, что натуральные числа – это числа, использующиеся при счете предметов, то есть числа 1, 2, 3 и т.д.

Для решения любой задачи и ее понимания полезно сразу разобрать примеры, приведенные в задаче. Согласно *первому примеру* при a=1 и b=2, количество натуральных чисел равно 99. Действительно, это числа 1, 2, 3, 4, 5, 6, 7, 8, 9 (они состоят из одной цифры),10,11,…,99 (они состоят из двух цифр). Всего получается *99 чисел*.

Согласно *второму примеру* при a=1 и b=1, количество натуральных чисел равно 9. Действительно, это числа 1, 2, 3, 4, 5, 6, 7, 8, 9 (они состоят из одной цифры).

Когда вы разобрали примеры, приведенные в задаче, и поняли их, полезно будет привести свой пример и разобрать его.

*Пример***.** a=2 и b=2, количество натуральных чисел равно всем двузначным числам: 10,11,12,…,99. Всего получается $99-10+1=90$ *чисел*.

Как видим, в ответах разобранных примеров встречаются только лишь нули и девятки. Если рассмотреть другие примеры, то ответы также будет содержать нули и девятки.

Проанализировав различные ситуации входных данных, можно сделать вывод:

1. При $a=b$ в ответе будет число, состоящее из одной девятки и нулей, количество которых равно $(a-1)$.
2. При $a<b$ в ответе будет число, состоящее из b цифр. В конце числа ($a-1$) нулей, а в начале числа – девятки. Количество девяток, таким образом, ($b - a+1$).
3. При $a>b$ в ответе будет 0 (таких чисел нет).

Таким образом, для решения этой задачи нет необходимости делать какие-либо вычисления. Фрагмент кода задачи приведен ниже и содержит только печать нужного количества девяток и нулей.

```
for i := a to b do write(9);
for i := 1 to a - 1 do write(0);
```

Обратите внимание, что предварительно составленная модель задачи позволяет в ряде случаев обойтись без сложных вычислений. Во многих олимпиадных задачах есть «изюминка», увидев которую, вы решите задачу легко и красиво. А, значит, ваш код и время работы программы будет меньше, чем при решении задачи стандартным способом «в лоб».

1.2. Техника работы с файлами при написании программ

Для того чтобы использовать файлы в программе, необходимо выполнить следующие действия (каждое из них проиллюстрируем конкретным примером):

– *описать переменную файлового типа в разделе описания переменных*

var f :text; {Описание переменной f как текстового файла}

– *связать файловую переменную с именем файла с помощью процедуры*

Assign (<Имя файловой переменной>, <Имя файла>);

Assign (f, ‘input.txt’);

– *открыть файл для чтения/записи процедурами*

Reset (<Имя файловой переменной>);

Rewrite (<Имя файловой переменной>);

Например, reset (f) – открытие файловой переменной f для чтения.

– *произвести чтение/запись из файла/в файл*

Текстовый файл	
Чтение	Read (<Имя файловой переменной>, <Список переменных>);
	Readln (<Имя файловой переменной>, <Список переменных>); Оператор вводит все значения переменных из списка и передвигает указатель ввода до начала следующей строки.

Запись	Write (<Имя файловой переменной>, <Список переменных>);
	Writeln (<Имя файловой переменной>, <Список переменных>); Оператор выводит все значения переменных из списка и передвигает указатель вывода до начала следующей строки.

Чтение из файла можно производить до тех пор, пока не будет достигнут конец файла.

Логическая функция – Eof (<Имя файловой переменной>) возвращает true, если достигнут конец файла, и false в противном случае.

– *закрыть файл в конце работы процедурой*

Close (<Имя файловой переменной>);

1.3. Пример программы задачи «Количество чисел»

Текст программы приводится на языке Pascal.

```
Program C;                    {Название программы}
{Раздел описания переменных}
Var a, b, i: Integer; f1, f2:text;
Begin                         {Начало программы}
{Связывание файловых переменных с файлами}
assign(f1,'input.txt');
assign(f2,'output.txt');
{Открытие файлов для чтения-записи}
reset(f1); rewrite(f2);
{Считывание входных данных}
read(f1,a, b);
if a > b then
     write(0)
else
     begin
     for i := b downto a  do write(f2,9);
     for i := a - 1 downto 1 do write(f2,0);
     end;
close(f1); close(f2); {Закрытие файлов}
End.
```

2. Примеры задач

2.1. Задача. День святого Франциска Ксавьера

(Турнир Архимеда, 2007 г.)

Сегодня в индийском городке Старый Гоа отмечают День святого Франциска Ксавьера. Святой Франциск Ксавьер – миссионер, член ордена иезуитов, считается величайшим католическим миссионером Нового времени.

Мощи святого Франциска Ксавьера – духовного покровителя индийских христиан – покоятся в Старом Гоа в известном католическом храме – Базилике Бом Иисус. Со дня постройки храма в 1605 году раз в десять лет мощи святого выставляются для всеобщего обозрения.

Индийский крестьянин Вирмарам – человек очень набожный, он не упускал случая коснуться святых мощей. Зная годы его жизни, определите, сколько раз он мог коснуться святых мощей (считается, что в год своего рождения он не мог этого сделать, а в год смерти – мог).

Входные данные

Вводятся два натуральных числа – год рождения и год смерти. Числа разделены пробелом.

Выходные данные

Программа должна определить, сколько раз крестьянин мог коснуться святых мощей, и вывести ответ - одно число.

Примеры

INPUT.TXT	OUTPUT.TXT
1610 1640	3
1615 1645	3
1515 1545	0

Составим модель задачи

Входные данные: a, b – натуральные числа, не превышающие текущего года. (тип данных integer)

Выходные данные: число c – количество раз касания святых мощей.

Решение задачи

Перед решением рассмотрим примеры, приведенные в задаче.

Первый пример. Входные данные 1610, 1640. Крестьянин мог коснуться мощей три раза: 1615, 1625, 1635 годы.

Второй пример. Входные данные 1615 1645. Крестьянин мог коснуться мощей три раза: годы 1625, 1635, 1645. В 1615 году (год рождения) он коснуться мощей не мог, а в 1645 – мог.

Третий пример. Входные данные 1515 1545. Крестьянин умер до строительства храма. В ответе 0.

Пример программы задачи «День святого Франциска Ксавьера»

```
Program task2;
Var a, b, c: Integer; f1,f2:text;
Begin
assign(f1,'input.txt'); reset(f1);
assign(f2,'output.txt'); rewrite(f2);
read(f1,a, b); close(f1);
{год рождения совпадает с годом смерти или превы-
шает его}
if a >=b then
     begin
     writeln(f2,0); close(f2); halt;
     end;
{умер раньше, чем построили храм}
if b<1605 then
     begin
     writeln(f2,0); close(f2); halt;
     end;
{Остальные случаи}
if (a<b) then
     begin
     {родился ранее 1605. Отсчет ведем от 1605}
     if a<1605 then c:=(b-1605) div 10 +1;
     {Родился позже 1605 года. Сначала находим год
a, с которого будем вести отсчет – год первого ра-
за посещения святого места. Затем находим количе-
ство посещений}
     if a>1605 then
```

```
            begin
            a:=1605+((a-1605) div 10 +1)*10;
            c:=(b-a) div 10 +1;
            end;
      {Если родился в 1605 году, то в этом году
святое место посетить не мог по условию задачи.
Поэтому не прибавляем 1}
      if a=1605 then c:=(b-a) div 10 ;
      end;
writeln(f2,c); close(f2);
End.
```

3. Домашнее задание

Ниже приведены задачи для домашнего решения. Вы можете решить данные задачи и попробовать сдать их на тех сайтах с системами автоматизированного тестирования, которые указаны в задачах. В этом случае вы точно будете знать насколько хорошо работает Ваша задача и сколько тестов она проходит.

3.1. Задача. Уравнение

Текст задачи приводится с сайта *http://informatics.mccme.ru*

Решите в целых числах уравнение: $\sqrt{(ax+b)} = c$, где *a, b, c* – данные целые числа. Найдите все решения или сообщите, что решений в целых числах нет.

Входные данные

Вводятся три числа *a, b* и *c*, разделенные пробелами. *a,b,c*<=100.

Выходные данные

Программа должна вывести все решения уравнения в порядке возрастания, либо NO SOLUTION (заглавными буквами), если решений нет. Если решений бесконечно много, вывести MANY SOLUTIONS.

Примеры

INPUT.TXT	OUTPUT.TXT
1 0 0	0
1 2 3	7
1 2 -3	NO SOLUTION

3.2. Задача. Дом - Школа – Дом

Текст задачи приводится с сайта *www.acmp.ru*

Мальчик Вася каждый день ездит на метро. Утром он едет в школу, а вечером того же дня, обратно из школы, домой. Для того, чтобы немного сэкономить, он покупает электронную смарт-карту на X поездок. Когда он хочет зайти в метро, он прикладывает карту к турникету. Если на карте осталось ненулевое количество поездок, то турникет пропускает Васю и списывает с карты одну поездку. Если же на карте не осталось поездок, то турникет не пропускает Васю, и он (Вася) вынужден купить на этой же станции новую карту на X поездок и вновь пройти через турникет.

Вася заметил, что в связи с тем, что утром метро переполнено, покупать новую карту утром накладно по времени, и он может опоздать в школу. В связи с этим он хочет понять: будет ли такой день, что с утра, поехав в школу, окажется, что у него на карточке ноль поездок.

Вася больше никуда на метро не ездит и поэтому заходит в метро только на станции около дома и на станции около школы.

Входные данные

Во входном файле INPUT.TXT содержится ровно 2 строки. В первой содержится слово «School» или «Home» в зависимости от того, где первый раз Вася купил карточку на X поездок. Во второй строке содержится натуральное число X, $1 \le X \le 1000$.

Выходные данные

В выходной файл OUTPUT.TXT следует вывести «Yes», если будет такой день, что утром у Васи на карточке окажется ноль поездок и «No» в противном случае.

Примеры

INPUT.TXT	OUTPUT.TXT
Home 1	Yes
School 2	No

3.3. Задача. Время прибытия

Текст задачи приводится с сайта *www.acmp.ru*

Задано время отправления поезда и время в пути до конечной станции. Требуется написать программу, которая найдет время прибытия этого поезда (возможно, в другие сутки).

Входные данные

Входной файл INPUT.TXT содержит две строки. В первой строке задано время отправления, а во второй строке – время в пути. Время отправления задается в формате «HH:MM», где HH время в часах, которое принимает значение от 00 до 23, MM – время в минутах, которое принимает значение от 00 до 59. Время в пути задается двумя неотрицательными целыми числами – количество часов и количество минут. Числа разделяются одним пробелом. Количество часов не превышает 120, минут – 59.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одну строку – время прибытия поезда на конечную станцию. Формат вывода этого времени совпадает с форматом ввода времени отправления.

Примеры

INPUT.TXT	OUTPUT.TXT
00:00 10 10	10:10
01:02 4 6	05:08
11:00 22 0	09:00

Модуль 2. Алгоритмы разветвляющейся структуры. Принадлежность точки фигуре

Задачи модуля

1. Знакомство с алгоритмами разветвляющейся структуры.
2. Изучение оператора условия, оператора выбора.
3. Рассмотрение примеров использования разветвляющихся алгоритмов.
4. Воспитание интереса к решению олимпиадных задач.

1. Теоретический материал

1.1. Булевский тип данных. Условный оператор. Оператор выбора

Булевский (логический) тип данных

Идентификатор булевского типа – Boolean. Переменная булевского типа может принимать всего два логических значения: True (истина) и False (ложь). Значению False соответствует число 0, значению True – 1. Булевский тип данных относится к группе порядковых типов (переменные которых можно упорядочить), к которым возможно применение функций pred(x) – возвращает предыдущее значение x, succ(x) – возвращает следующее за x значение, ord(x) – возвращает порядковый номер x.

ord(false)=0; succ(false)=true;
ord(true)=1; pred(true)=false.

Над логическими значениями можно выполнять операции сравнения, логические операции: and (логическое и), or (логическое или), not (отрицание).

False *and* False = False	False *or* False = False
False *and* True = False	False *or* True = True
True *and* False = False	True *or* False = True
True *and* True = True	True *or* True = True
not False = True	*not* True = False

Из переменных, констант, операций отношения (<, >, >=, <=, =, <>), логических операций, скобок строятся *логические выражения*. Значение логического выражения имеет булевский тип. Например, x:=((5 div 2)>3) and false – переменная булевского типа.

Составной оператор

Составной оператор объединяет несколько операторов, которые воспринимаются средой программирования как один. Синтаксис записи оператора:

```
begin
      <оператор>;
      <оператор>;
         .   .   .
      <оператор>;
end;
```

Используемые зарезервированные слова begin, end называют при этом операторными скобками.

Условный оператор

Условные операторы служат для организации ветвления в программах. Синтаксис условного оператора:

```
if<логическое выражение (условие)>then
      <оператор 1>
[else
      <оператор 2>];
```

Часть оператора, заключенная в квадратные скобки, может опускаться. В этом случае оператор будет являться неполным условным оператором. Семантика оператора представлена на рис.1б.

Если необходимо после ключевого слова then или else использовать несколько операторов, а не один, то они оформляются операторными скобками. Например,

```
if d>0 then
      begin
      x1:=(-b-sqrt(d))/(2*a); x2:=(-b+sqrt(d))/(2*a);
      writeln ('Квадратное уравнение имеет два кор-
ня', x1,x2);
      end.
```

Обратите внимание, что перед ключевым словом else точка с запятой не ставится.

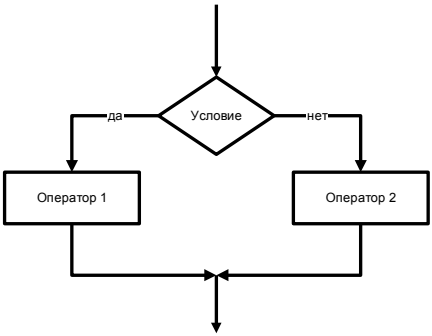


Рис. 1а. Полная разветвляющаяся структура

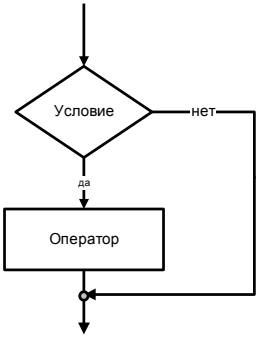


Рис. 1б. Неполная разветвляющаяся структура (обход)

Оператор выбора

Оператор выбора выполняет одно из перечисленных в программе действий в зависимости от значения выражения – селектора.

Синтаксис оператора выбора:

```
Case<выражение>of
      <список констант 1> :<оператор 1>;
                  . . .
      <список констант N> :<оператор N>
else<оператор>;
end;
```

Выражение порядкового типа вычисляется и его значение отыскивается в одном из списков констант. После этого выполняется оператор, соответствующий списку. Если ни в одном из списков необходимого значения найдено не будет, то выполняется оператор после ключевого слова else. Семантика оператора представлена на рис.2.

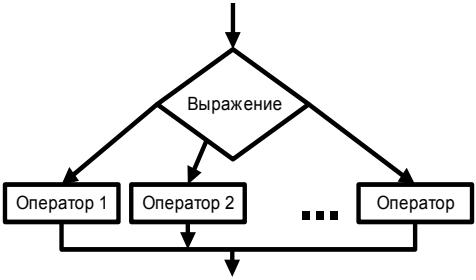


Рис. 2. Структура оператора Case

Пример

```
writeln ('Введите номер месяца');
readln (n);
case n of
     1,2,12 :writeln ('Зима');
     3,4,5 : writeln ('Весна');
     6,7,8 : writeln ('Лето');
     9,10,11 : writeln ('Осень');
end.
```

2. Задачи на анализ условий

2.1. Задача. Задача принадлежности точки фигуре

Задачи принадлежности точки фигуре – классические и распространенные задачи на использование условного оператора. Для решения таких задач необходимо знание математических функций, их графиков, умение составлять математическую модель задачи.

Задание. Принадлежит ли точка (x;y) ромбу с координатами вершин (-1,0), (0,1), (1,0), (0,-1).

1. *Математическая модель задачи*

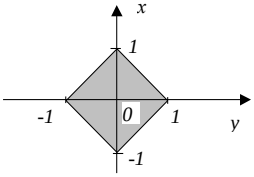


Рис. 3. Графическая иллюстрация задачи

Условие принадлежности точки фигуре:

$$\begin{cases} y <= x+1, \\ y <= -x+1, \\ y >= x-1, \\ y >= -x-1. \end{cases} \text{ или, короче } \begin{cases} y <= -|x|+1, \\ y >= |x|-1. \end{cases}$$

2. *Алгоритм (приведен для наглядности разветвляющейся структуры алгоритма)*

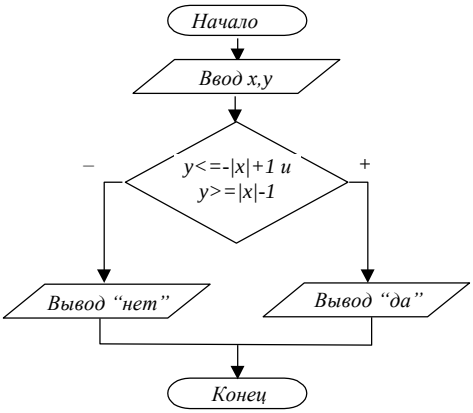


Рис. 4. Блок-схема решения задачи

3. *Программа*

```
Program Romb;
Uses crt;
Var x,y: real;
Begin
Clrscr;
writeln('Введите координаты заданной точки');
readln(x,y);
if (y<= - abs (x)+1) and (y>= abs (x) -1) then
     writeln('Точка принадлежит фигуре')
else
     writeln('Точка не принадлежит фигуре');
End.
```

4. Контрольные примеры.

x y	0.5 0	0.2 0.2	0.5 2	– 1.1 0.3
	да	да	нет	нет

2.2. Задача. Треугольник и точка

В декартовой системе координат на плоскости заданы координаты вершин треугольника и еще одной точки. Определить, принадлежит ли эта точка треугольнику.

Ограничения: координаты вершин — целые числа, для любой точки выполняются следующие условия: -10 000<x, y < 10 000, время работы программы — 1 с.

Ввод из файла tria-pt.in. В четырех строках находятся пары чисел — координаты точек. Числа в первых трех строках — это координаты вершин треугольника, в четвертой строке — координаты тестируемой точки.

Вывод в файл tria-pt.out. Вывести слово In, если точка находится внутри треугольника, или Out — если снаружи.

Примеры

tria-pt.in	tria-pt.in	tria-pt.in	tria-pt.in
0 0	0 0	0 0	0 0
100 0	100 0	100 0	100 0
0 100	0 100	0 100	0 100
100 100	10 10	50 50	0 0
tria-pt.out	tria-pt.out	tria-pt.out	tria-pt.out
Out	In	In	In

Модель задачи

Уравнение прямой вида Ax + By + C = 0 обладает следующим свойством. Если в него подставить координаты точки, принадлежащей этой прямой, то оно превратится в тождество. Если точку взять из одной из полуплоскостей, на которые прямая делит плоскость, то будет выполняться неравенство Ax + By + C < 0, а если из другой — то Ax + By + C > 0.

Рассмотрим уравнения трех прямых, проходящих через две вершины треугольника. Если для всех трех прямых заданная точка

лежит в той же полуплоскости, что и третья вершина треугольника (через которую не проходит прямая), то она принадлежит треугольнику.

Детали реализации

• Обратите внимание на то, что следует отвечать, если точка лежит на границе треугольника. В условии в явном виде ответа нет, но следует обратить внимание на примеры. В одном из них такой случай рассматривается.

• Наиболее часто встречающейся при решении этой задачи ошибкой является использование неподходящих типов данных. Например, использование типа integer. Важно обратить внимание на диапазон типов данных вашего языка программирования.

• Обратите внимание на то, что именно нужно выводить. В олимпиадных задачах нужно выводить именно то, что упомянуто в условии. (С заглавной буквы? Есть ли точка в конце? Ответив на эти вопросы, вы избежите ошибок в работе программы.)

```
{Written by F.Menshikov Приводим решение из книги
«Олимпиадные задачи» Ф.Меньшикова. Самостоятельно
Можно составить решение и не содержащие процедур и
функций}
type
tpoint=record
x,y:longint;
end;
tline=record
a,b,c:longint;
end;
procedure getLine(p1,p2:tpoint;var l:tline);
begin
l.a:=p2.y-p1.y;
l.b:=p1.x-p2.x;
l.c:=p1.y*(p2.x-p1.x)-p1.x*(p2.y-p1.y);
end;
```

```
function lineValue(l:tline;p:tpoint):longint;
begin
lineValue:=l.a*p.x+l.b*p.y+l.c;
end;

function equalSign(x,y:longint):boolean;
begin
if (x=0)or(y=0) then
     equalSign:=true
else
     if (x>0)and(y>0) then
          equalSign:=true
     else
          if (x<0)and(y<0) then
               equalSign:=true
          else
               equalSign:=false;
end;

procedure readPoint(var p:tpoint);
begin
readln(p.x,p.y);
end;

Var
   t1,t2,t3:tpoint;
   p:tpoint;
   l1,l2,l3:tline;
   side1,side2,side3:boolean;
Begin
     assign(input,'tria-pt.in'); reset(input);
     readPoint(t1);
     readPoint(t2);
     readPoint(t3);
     readPoint(p);
     close(input);
     getLine(t1,t2,l3);
```

```
      getLine(t1,t3,l2);
      getLine(t2,t3,l1);
      side1:=equalSign(lineValue(l1,t1),
      lineValue(l1,p));
      side2:=equalSign(lineValue(l2,t2),
      lineValue(l2,p));
      side3:=equalSign(lineValue(l3,t3),
      lineValue(l3,p));
      assign(output,'tria-pt.out');
      rewrite(output);
      if side1 and side2 and side3 then
           writeln('In')
      else
           writeln('Out');
      close(output);
End.
```

2.3. Задача. Преферанс (Турнир Архимеда 2011 г.)

При игре в преферанс-2011 используется колода в 42 карты (в ней 4 туза). По 10 карт раздаётся четырём игрокам и две карты откладываются в прикуп. Каждый игрок похвастал, сколько у него тузов. Определите, сколько игроков заведомо солгали.

Например, они сказали 1, 1, 1, 2. Следовательно, заведомо солгал один игрок. (Какие-то трое могли сказать правду, но все четверо правду сказать не могли, так как тузов всего 4).

Входные данные

Вводятся 4 числа (от 0 до 9 каждое), разделённых пробелом – количество тузов по словам первого, второго, третьего и четвертого игроков.

Выходные данные

Выведите одно число – минимальное количество игроков, которые заведомо солгали. Если все одновременно могли сказать правду, выведите число 0.

Примеры

INPUT.TXT	OUTPUT.TXT
1 1 1 2	1
1 1 1 1	0

Решение задачи

Первое, что требовалось заметить в задаче: игроки могут не только похвастать, что у них больше тузов, чем на самом деле, но преуменьшить реально число тузов, находящихся у них на руках. Поскольку в прикупе осталось всего две карты, у игроков в сумме должно быть *не менее двух тузов*. Теперь мы готовы перейти к решению задачи.

Рассмотрим сумму чисел, названных игроками. Возможны три случая:

1) Сумма меньше 2. В этом случае в прикупе осталось больше двух тузов, чего не может быть. Следовательно, хотя бы один из игроков соврал. Поскольку в этом случае всегда найдется игрок, сказавший, что у него 0 тузов, можно предположить, что соврал только он, и на самом деле у него 1, 2, 3 или 4 туза. Таким образом, ответ в этом случае равен 1: (Код приводится на языке С. Вы самостоятельно можете при желании транслировать его на Pascal)
```
# s – список чисел, названных игроками
If sum(s) < 2:
      print(1);
```
2) Сумма чисел лежит в пределах от 2 до 4 (включительно). В этом случае все игроки могли сказать правду:
```
Elif 2 <= sum(s) <= 4:
      print(0);
```
3) Сумма чисел, названных игроками, превосходит 4. В этом случае один или несколько игроков солгали, и нам нужно определить наибольшее возможное количество игроков, которые могли сказать правду. Ясно, что сумма названных чисел не должна превосходить 4. Чтобы игроков, сказавших правду, было как можно больше, нужно считать, что неправду сказали игроки, назвавшие как можно большие числа (например, если игроки назвали числа 1 3 1 2, то наилучший вариант: не солгали игроки, сказавшие 1, 1 и 2). Таким образом, достаточно отсортировать числа по возрастанию и выбрать несколько первых чисел так, чтобы их сумма была не больше 4. Все остальные игроки при этом окажутся лжецами:

```
else:
{
        s.sort();
        i = 0; sum = 0;
        while sum <= 4:
        {sum += s[i];
                i += 1;}
        # i – номер первого лжеца
        print(4 - i + 1);
}
```

3. Домашнее задание

3.1. Задача. Больше-меньше (*www.acmp.ru*)

Одна из основных операций с числами – их сравнение. Мы подозреваем, что вы в совершенстве владеете этой операцией и можете сравнивать любые числа, в том числе и целые. В данной задаче необходимо сравнить два целых числа.

Входные данные

В двух строчках входного файла INPUT.TXT записаны числа A и B, не превосходящие по абсолютной величине 2*109.

Выходные данные

Запишите в выходной файл один символ “<”, если A < B, “>”, если A > B и “=”, если A=B.

Примеры

INPUT.TXT	OUTPUT.TXT
5 7	<
-7 -12	>
13 13	=

3.2. Задача. Кирпичная кладка

Заданы размеры A, B прямоугольного отверстия в стене и размеры X, Y, Z кирпича. Определить пройдет ли кирпич через отверстие.

Входные данные

Во входном файле INPUT.TXT записано пять натуральных чисел – A, B, X, Y, Z.

Выходные данные

Запишите в выходной файл OUTPUT.TXT ответ в виде сообщения YES или NO.

Примеры

INPUT.TXT	OUTPUT.TXT
12 18 14 20 11	YES
16 24 8 28 32	NO

3.3. Задача. Шахматный конь

Вася решил научиться играть в шахматы. Он нашел книгу с записями партий и внимательно их изучает. Может быть, когда-нибудь Вася станет великим шахматистом, но пока он еще учится в начальной школе, и ему нелегко дается шахматная нотация. Больше всего трудностей у Васи вызывают ходы шахматного коня. Он попросил вас написать программу, которая сможет сообщить Васе, на какие клетки можно пойти конем с заданной клетки.

Вы, наверное, тоже знаете, что конь в шахматах всегда перемещается либо на две клетки по горизонтали и на одну по вертикали, либо на одну по горизонтали и на две по вертикали. Вертикали обозначаются маленькими латинскими буквами от a до h, а горизонтали - цифрами от 1 до 8. Любая клетка на шахматной доске обозначается буквой соответствующей вертикали и цифрой соответствующей горизонтали, например, c6 или e2.

Входные данные

Во входном файле INPUT.TXT записано 2 символа – координаты клетки, где стоит конь.

Выходные данные

В выходной файл OUTPUT.TXT в произвольном порядке выведите все координаты клеток, на которые за один ход может попасть конь, находящийся на заданной клетке.

Примеры

INPUT.TXT	OUTPUT.TXT
e2	c1 c3 d4 f4 g1 g3
a1	b3 c2

Модуль 3. Алгоритмы циклической структуры. Простые числа

Задачи модуля

1. Знакомство с алгоритмами циклической структуры.
2. Изучение оператора цикла с параметром, оператора цикла без параметра.
3. Рассмотрение примеров использования циклических алгоритмов.
4. Воспитание интереса к решению олимпиадных задач.

1. Теоретический материал

1.1. Операторы цикла без параметра. Операторы цикла с параметром

Операторы цикла без параметра

К операторам цикла без параметра относятся цикл с предусловием While и цикл с постусловием Repeat. Синтаксис и семантика операторов:

Цикл с предусловием

While<логическое выражение>do
<оператор>;

Оператор будет выполняться пока истинно логическое выражение.

Цикл с постусловием

Repeat
<оператор 1>;
...
<оператор N>;
Until<логическое выражение>;

Часть программы, заключенная между ключевыми словами Repeatи Until, повторяется до тех пор, пока не станет истинным логическое выражение.

Пример

Вычислить сумму первых *n* членов суммы 1+1/2 +1/3 +1/4 +…

```
writeln ('Введите количество членов гармонического ряда');
readln (kol);
```

```
s:=0;       {Сумма n-первых членов }
n:=0;       {Счетчик количества членов }
repeat                                  while n<kol do
     n:=n+1;                                 begin
     s:=s+1/n;                               n:=n+1;
until n>kol;                                 s:=s+1/n;
                                             end;
```

В отличие от оператора While в операторе Repeat вычисление логического выражения происходит не до, а после очередного повторения цикла. Из-за этого цикл Repeat выполнится хотя бы один раз, а цикл While может не выполниться ни разу.

Оператор цикла с параметром

Оператор For обеспечивает повторение цикла, управляемое переменной – счетчиком. Синтаксис оператора цикла с параметром:
For<счетчик цикла>:=<выражение 1>to<выражение 2>do
<оператор>;

Счетчик цикла должен являться переменной порядкового типа. Выражение 1 и выражение 2 – выражения порядкового типа и представляют собой начальное и конечное значение счетчика соответственно. Выполнение начинается с вычисления выражения 1 и выражения 2. Затем счетчик получает значение выражения 1 и делается проверка, не превышает ли значение счетчика выражения 2. Если не превышает, выполняется оператор. После завершения оператора счетчик получает следующее по порядку значение и все повторяется, начиная с проверки. Когда значение счетчика становится равным выражению 2, оператор выполняется последний раз.

Семантика оператора For представлена на рис. 5.

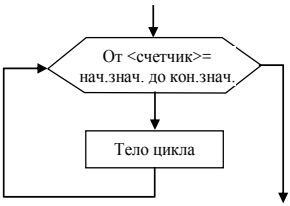


Рис. 5. Структура оператора For

Пример

Вычислить n!

```
{ переменная, накапливающая произведение, получает значение 1}
p:=1;
{перебираются все множители от 1 до числа n включительно}
for i:=1 to n dop:=p*i;  {итерационно вычисляется факториал}
```

Возможен вариант оператора For, когда переменная принимает последовательно убывающие значения:

For<счетчик цикла>:=<выражение 1>downto<выражение 2>do<оператор>;

1.2. Порядковые типы данных

Порядковыми называются все типы, значения которых можно расположить в возрастающем порядке. Из стандартных типов языка Pascal к порядковым относятся: все целые типы, Char, Boolean. Для переменных порядковых типов определены функции pred(x), succ(x), ord(x). Для переменных порядковых типов применимы также все шесть операций сравнения: <, >, >=, <=, =, <>. Операции сравнения возможны, так как все значения порядковых типов нумеруются числами 0, 1, 2, ... , начиная с наименьшего. Исключение составляет тип Integer, для которого ord(x)=x.

Значениями типа Char (символьного типа) являются отдельные символы. Символьные константы заключаются в кавычки, например, ‘3’, ‘d’, ‘Л’, ‘ ‘. Символьные значения можно вводить, выводить, присваивать, сравнивать. Сравнение символов происходит на основе их кодов по таблице Ascii – кодов.

Пример

```
var Name, Surname: char;
begin
writeln (‘Введите инициалы имени, отчества’);
readln (Name, Surname);
end.
```

Для символьного типа в Pascal имеются две стандартные функции:

Chr (n) – возвращает символ с кодом n. Ord (s) – возвращает код символа s.

К пользовательским порядковым типам относятся перечислимый и интервальный типы. Данные типы должны быть предварительно описаны в разделе описания типов type.

Перечислимый тип задается путем перечисления в круглых скобках всех допустимых значений типа. Значения не могут быть ничем иным, кроме имен. Их можно присваивать и сравнивать. Из двух значений большим является то, которое в описании типа расположено правее.

Пример

```
type Distr=(Man, Women);
var p: distr;
begin
writeln ('Ваш пол – Man, Women');
readln (p);
. . .
end.
```

Интервальный тип определяется как интервал значений ранее определенного (базового) типа. Описывается интервал своими крайними значениями в разделе типов:
<имя интервального типа>=<значение 1> ...<значение 2>.

Базовым может служить любой простой тип данных, кроме вещественных.

Например,

```
type
      day=1..31;
      month=1..12;
      letter='a'..'z';
```

2. Задача с использованием циклических алгоритмов

2.1. Задача определения простых чисел в заданном диапазоне

Во многих задачах по программированию необходимо в теле одного цикла использовать другой цикл. Такие структуры являются вложенными.

Вывести на экран все простые числа в заданном диапазоне и

подсчитать их количество. Самый простой (но не самый рациональный способ) решения данной задачи – проверка каждого числа N на делители от 2 до N-1.

```
{Программа определения простых чисел в заданном диапазоне}
Program prost;
Var
     n_gr, v_gr :integer; {Верхняя и нижняя границы диапазона}
     i,j, kol :integer; {i – счетчик цикла, j – делитель числа, kol – количество простых чисел}
     flag: boolean;
Begin
kol:=0;
writeln ('Введите нижнюю и верхнюю границу диапазона');
read (n_gr, v_gr);
for i:=n_gr to v_gr do
     begin
     j:=1; flag:=false; {Предполагаем, что число не делится ни на один из делителей от 2 до i-1, т.е. является простым}
     repeat
          j:=j+1;
          if i mod j =0 then
               flag:=true; {Меняем flag как только число поделилось без остатка хотя бы на один из делителей от 2 до  i-1}
     until flag or (j=i-1);
     if not flag then
          begin
          writeln (i);
          kol:=kol+1;
          end;
     end;
writeln ('Количество простых чисел в диапазоне', kol);
End.
```

Подумайте, как можно сократить количество итераций вложенного цикла? Для этого условие выхода из цикла *until flag or (j=i-1)*

можно сделать более оптимальным, не перебирая все возможные делители до *j=i-1*. Предположите, до какого значения j можно сделать перебор?

2. 2. Задача. Наибольший общий делитель (*www.acmp.ru*)

Даны два натуральных числа A и B. Требуется найти их наибольший делитель (НОД).

Входные данные

Во входном файле INPUT.TXT в единственной строке записаны натуральные числа A и B через пробел (A, B ≤ 10^9)

Выходные данные

В выходной файл OUTPUT.TXT выведите НОД чисел A и B.

Пример

INPUT.TXT	OUTPUT.TXT
12 42	6

Решение задачи (с сайта www.acmp.ru)

Напомним, что наибольшим общим делителем (НОД) для целых чисел a и b называют такое наибольшее число d, что a делится на d и b делится на d. При этом записывают это так: НОД(a,b)=d. Числа a и b называют взаимно простыми, когда НОД(a,b)=1.

Наиболее функциональным и практичным методом поиска НОД является алгортим Евклида, который заключается в следующем: используя свойство НОД(a,b)=НОД(a, a mod b) мы можем свести данную запись к виду НОД(d,0), где d - искомый наибольший делитель. Другими словами, пока у нас оба числа a и b больше нуля следует наибольшее из них заменять на остаток от деления наибольшего на наименьшее, достаточно быстро мы получим таким образом решение задачи.

Реализация алгоритма Евклида для поиска НОД на алгоритмическом языке может выглядеть следующим образом:

```
read(a,b);
while a*b > 0 do
if a >= b then a := a mod b else b := b mod a;
write(a+b);
```

2. 3. Задача. Конец К-го урока

(Московская окружная олимпиада 2009 г.)

В школе продолжительность каждого урока 45 минут, а перемены между уроками – всего 5 минут. Первый урок начинается ровно в 8 часов утра. Напишите программу, отвечающую на вопрос «во сколько в этой школе заканчивается *K*-ый урок?»

Входные данные

Вводится одно натуральное число K, не превышающее 15.

Выходные данные

Выведите время окончания *K*-ого урока: сначала часы, потом минуты, разделяя их пробелом.

Примеры

INPUT.TXT	OUTPUT.TXT
1	8 45
6	12 55

Подход к решению задачи

Эту задачу можно решить и без помощи циклов. Достаточно вывести закономерность – сколько минут проходит от начала первого урока до конца k-урока. Далее полученные минуты перевести в часы и минуты (в часе 60 минут) и прибавить к 8 часам (начало первого урока) полученное количество часов, затем дописать полученные минуты.

3. Домашнее задание

3.1. Наименьшее общее кратное (www.acmp.ru)

Требуется написать программу, определяющую наименьшее общее кратное (НОК) чисел *a* и *b*.

Входные данные

В единственной строке входного файла INPUT.TXT записаны два натуральных числа A и B через пробел, не превышающих 46340.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести целое число – НОК чисел A и B.

Примеры

INPUT.TXT	OUTPUT.TXT
36 27	108
39 65	195

3.2. Задача. Очередь *(www.acmp.ru)*

Студент Василий живёт в общежитии. Отделение банка, в котором он производит оплату за проживание, имеет всего две кассы, поэтому почти всегда длинная очередь к ним. Первая касса открывается в 8.00, вторая – в 8.05. Последний клиент будет принят в 20.00. Очередь единая, и очередной клиент обслуживается, как только освобождается одна из касс. На обслуживание одного клиента уходит ровно 10 минут. Василий приходит ровно в 8.00 и видит, сколько человек стоит перед ним. Требуется определить, сколько времени ему придётся простоять в очереди, и вообще обслужат ли его сегодня.

Входные данные

Входной файл INPUT.TXT содержит единственное натуральное число K – номер Василия в очереди (K<250).

Выходные данные

В выходной файл OUTPUT.TXT выводится строка «NO», если Василий сегодня заплатить уже не успеет, и время его ожидания (в формате «XY», где X – количество целых часов и Y – количество минут), если всё же успеет заплатить.

Примеры

INPUT.TXT	OUTPUT.TXT
1	0 0
20	1 35
235	NO

3.3. Задача. Цифровой корень

Цифровой корень числа получается при сложении цифр числа, затем при сложении цифр вновь полученного числа и так далее до тех пор, пока в качестве суммы не будет получено число, состоящее из одной цифры.

Входные данные

Входной файл INPUT.TXT содержит единственное натуральное число $N \le 10000$.

Выходные данные

В выходной файл OUTPUT.TXT выводится одно число– цифровой корень заданного числа.

Примеры

INPUT.TXT	OUTPUT.TXT
325	1
1901	2

Модуль 4. Алгоритмы обработки массивов

Задачи модуля

1. Знакомство с организацией данных в виде массивов.
2. Изучение стандартных алгоритмов обработки массивов: поиск элемента массива, вычисление суммы элементов массива, поиск максимального (минимального) элемента массива.
3. Изучение методов сортировки массивов.
4. Рассмотрение олимпиадных задач с использованием массивов.

1. Теоретический материал

1.1. Знакомство с организацией данных в виде массивов

Массив - это упорядоченная совокупность переменных, объединенных общим типом и именем. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется. Каждый элемент массива определяется именем, совпадающим с именем массива, а также индексом. Индекс - это величина, характеризующая положение элемента в массиве.

На рисунке показан массив с именем А из десяти элементов – целых чисел.

3	-9	89	2	5	6	78	-90	1	32

Описание данного массива на языке Паскаль выглядит следующим образом:

```
var a : array [1..10] of integer;
```

К каждому элементу массива можно обращаться отдельно так же как к обычной переменной установленного типа. Элемент массива с номером i имеет обозначение a[i]. Благодаря такой записи данные можно обрабатывать в цикле. Входные данные задачи, которые необходимо хранить в массиве, обычно задаются следующим образом: сначала идет число N - количество элементов массива, а за ним следуют отделенные пробелом значения N элементов массива. Алгоритм чтения N элементов массива и вывод их в файл может выглядеть следующим образом:

```
readln(f1, n); {считали количество элементов мас-
сива.}
```

```
for i:=1 to n do read(f1, a[i]); {чтение элементов
из файла в массив}
{вывод элементов из массива в файл через пробел}
for i:=1 to n do write(f2, a[i], ' ');
```

Рассмотрим простейшую задачу на организацию и обработку массивов.

1.2. Задача. Разворот

Дано натуральное число N и последовательность из N элементов. Требуется вывести эту последовательность в обратном порядке.

Входные данные

В первой строке входного файла INPUT.TXT записано натуральное число N ($N \le 10^3$). Во второй строке через пробел идут N целых чисел, по модулю не превосходящих 10^3 - элементы последовательности.

Выходные данные

В выходной файл OUTPUT.TXT выведите заданную последовательность в обратном порядке.

Пример

INPUT.TXT	OUTPUT.TXT
3 1 2 3	3 2 1

Решение

Для решения поставленной задачи считаем данные из файла и выведем их в обратном порядке.

```
{читаем количество элементов в n}
readln(f1, n);
{чтение элементов из файла в массив}
for i:=1 to n do read(f1,a[i]);
{вывод элементов из массива в файл в обратном по-
рядке}
for i:=n downto 1 do write(f2,a[i],' ');
```

2. Стандартные алгоритмы обработки массивов: поиск элемента массива, вычисление суммы элементов массива, поиск максимального (минимального) элемента массива.

2.1. Задача. Поиск элемента массива

Дано натуральное число N, M и последовательность из N элементов. Требуется вывести YES, если такой элемент есть в последовательности и NO, если такого элемента нет в последовательности.

Входные данные

В первой строке входного файла INPUT.TXT записано натуральное число N ($N \le 10^3$), M ($M \le 10^3$). Во второй строке через пробел идут N целых чисел, по модулю не превосходящих 10^3 - элементы последовательности.

Выходные данные

В выходной файл OUTPUT.TXT выведите YES или NO.

Пример

INPUT.TXT	OUTPUT.TXT
3 2 1 2 3	YES

```
Program p;
Const n=1000;
Var i,k,m: Integer; flag: boolean;
    a:array[1..n] of integer;
    f1, f2:text;
Begin
{Начало программы}
{Связывание файловых переменных с файлами}
assign(f1,'input.txt');
assign(f2,'output.txt');
{Открытие файлов для чтения-записи}
reset(f1); rewrite(f2);
{Считывание входных данных}
readln(f1,k,m);
for i := 1 to k  do read(f1,a[i]);
```

```
flag:=false {предполагаем, что искомого элемента
нет}
i:=1;
repeat
      if a[i]=m then flag:=true;
      i:=i+1;
until flag or (i=n+1);
if flag write(f2,'YES') else write(f2,'NO');
{Закрытие файлов}
close(f1); close(f2);
End.
```

2.2. Задача. Вычисление суммы элементов массива

```
S:=0; {полагаем сумму равной нулю}
for i:= 1 to n do
      s:=s+a[i]; {наращиваем сумму, прибавляя оче-
редной элемент}
```

2.3. Задача. Поиск максимального элемента массива

```
var a:array[1..10] of integer;
    max:integer;
    i:integer;
Begin
max:=-(MAXINT+1);
for i:=1 to 10 do
      begin
      if max < a[i] then max:=a[i];
      end;
writeln(max);
End.
```

2.4. Задача. Сортировка массивов методом «пузырька»

Алгоритм сортировки методом «пузырька» заключается в упорядочении элементов массива по возрастанию или убыванию. Он основан на сравнении каждых двух соседних элементах и «если пара правильная» (элементы расположены по возрастанию или убыванию – как требуется в задаче), то она остается без изменений. Если же пара «неправильная», то элементы меняются местами.

```
{ сортировка массива "пузырьком" по возрастанию}
const n = 10;
{ количество элементов в массиве}
var a:array[1..n] of integer;
    i,j,x:integer;
Begin
writeln('введите ',n,' элементов массива');
for i:=1 to n do readln( a[i] );
for i:=1 to n-1 do
     begin
     for j:=i+1 to n do
          begin
          if a[i]>a[j] then
               begin
               x:=a[i]; a[i]:=a[j]; a[j]:=x;
               end;
          end;
     end;
writeln('после сортировки:');
for i:=1 to n do writeln( a[i] );
End.
```

3. Домашнее задание

3.1. Задача. Золотые слитки (Турнир Архимеда 2011 г.)

Дано N золотых слитков. Требуется распилить не более одного из них на две части (не обязательно равные, но с целой массой), после чего разделить слитки на две кучи равной массы.

Входные данные

В первой строке вводится одно натуральное число N, не превосходящее 100.

Во второй строке через пробел вводятся N натуральных чисел, не превосходящих 100 – массы имеющихся слитков.

Выходные данные

Выведите массы слитков, которые вошли в первую кучку (включая массу части распиленного слитка).

Если решений несколько, выведите любое из них.

Если решений нет, выведите фразу NO SOLUTION (заглавными буквами).

Выводить массы можно в произвольном порядке, но масса части распиленного слитка (если таковой имеется) должна быть последней.

Примеры

INPUT.TXT	OUTPUT.TXT
3 5 5 5	NO SOLUTION
4 1 2 3 8	2 3 1 1
4 1 2 3 8	7

3.2. Задача. Единицы (Турнир Архимеда 2011 г.)

В числе подсчитали количество единиц, в получившемся числе опять подсчитали количество единиц и т.д. Например: 111211121112111 – 12 – 1 – 1 – 1 – 1 …

В итоге полученная последовательность стабилизировалась. На каком числе?

Например, последовательность 111211121112111 – 12 – 1 – 1 – 1 – 1… стабилизировалась на числе 1.

Входные данные

Вводится одно натуральное число, состоящее не более чем из 100 цифр.

Выходные данные

Выведите число, на котором стабилизировалась последовательность.

Примеры

INPUT.TXT	OUTPUT.TXT
12345	1
2007	0

3.3. Задача. 2011 (Турнир Архимеда 2011 г.)

Представьте число 2011 в виде суммы K последовательных простых чисел (то есть простых чисел, между которыми нет других простых чисел). Например, число 31 можно представить в виде суммы трех последовательных простых чисел следующим образом: 7+11+13=31.

Входные данные

Вводится одно натуральное число K(от 1 до 2011).

Выходные данные

Выведите слагаемые в порядке возрастания, разделяя их пробелом. Если разложить в сумму K слагаемых невозможно, выведите NO SOLUTION(заглавными буквами).

Примеры

INPUT.TXT	OUTPUT.TXT
3	661 673 677
2	NO SOLUTION

Комментарий: Не забывайте, что 1 не является простым числом.

Модуль 5. Анализ заданий муниципального этапа ВОШ по информатике

Задачи модуля

1. Анализ заданий муниципального этапа ВОШ по информатике.
2. Разбор наиболее часто встречающихся ошибок.
3. Рекомендации по подготовке к муниципальному этапу олимпиады.

1. Теоретический материал

1.1. Задания муниципального этапа олимпиады по информатике (2009-2010)

Вначале данного раздела приведем текст заданий, затем проведем разбор заданий.

Указания к выполнению заданий

Решение каждой задачи оформляется в виде отдельного файла с именем, представляющем собой написание латинскими буквами фамилии конкурсанта (не более 7 символов) и завершающемся цифрой 1, 2 или 3 - номером решаемой задачи. Входной и выходной файлы располагаются в текущей директории, из которой будет запускаться программа. Не допускается пользоваться никакими своими носителями и книгами.

Задание 1. Неповторяющаяся последовательность

Из чисел 1, 2, 3 требуется сформировать вектор длины N, так чтобы в нём не было двух одинаковых рядом стоящих подвекторов.

Входная информация содержится в файле *vect.in*. В первой строке указывается количество чисел в векторе (0 <=N<= 30 000). В выходном файле *vect.out* записывается полученный вектор без пробелов. Если такой вектор сформировать нельзя, то записывается 0.

Время работы программы не более 3 секунд.

Пример

Vect.in	Vect.in	Vect.in
7	5	7
Vect.out	Vect.out	Vect.out
1231213	12213	12312131231213

В первом случае вектор сформирован правильно, во втором повторяется подвектор 2, в третьем повторяется подвектор 1231213.

Задание 2. Максимальный вектор

При решении задачи распознавания линейных образов возникла следующая задача. На входе дан вектор X, содержащий N целых чисел; выходом является максимальная сумма элементов среди всех непрерывных подвекторов входного вектора.

Входная информация содержится в файле *max.in*. В первой строке указывается количество чисел в векторе (0 <= N<= 30 000). В следующих строках записываются элементы вектора. В выходном файле *max.out* записывается максимальная сумма подвектора.

Если все числа вектора отрицательные, то программа должна записать 0. Время работы программы не более 3 секунд.

Пример

max.in	max.in	max.in
7 1 -3 5 -7 2 4 -2	5 5 -2 4 -3 5	3 -2 -4 -1
max.out	max.out	max.out
6	9	0

Задание 3. Площадь многоугольника

Определить площадь фигуры, заданной координатами своих вершин на плоскости в декартовой системе координат. Стороны многоугольника не соприкасаются (за исключением соседних – в вершинах) и не пересекаются. Входная информация содержится в файле *square.in*. В первой строке указывается количество вершин (3 <= N<= 30 000). В последующих N строках содержатся целочисленные координаты вершин, разделённые пробелом. Если соединить точки в данном порядке, а также первую и последнюю точки, то получится заданный многоугольник.

В выходном файле *square.out* записывается площадь многоугольника, округлённая до 1 знака после запятой.

Время работы программы не более 3 секунд.

Пример

Square.in	Square.in
4 5 0 0 5 -5 0 0 -5	4 0 4 0 0 3 0 1 1
Square.out	Square.out
50.0	3.5

2. Разбор Задания муниципального этапа олимпиады по информатике (2009-2010) (Автор: Ю.А. Кондрашин)

2.1. Задание. Неповторяющаяся последовательность

Для решения задачи необходимо оформить проверку последовательности на правильность, т.е. отсутствие двух рядом стоящих одинаковых подпоследовательностей. Если длина вектора X равна N, то длина подпоследовательности может быть от 1 до N\2 (деление нацело). Это верно если проверка проводится с начала вектора. Если же вектор проверяется с позиции L (1 <L<N), то длина может изменяться 1 до (N-L+1) \ 2.

1	*2*	*3*	*1*	*2*	*1*	*3*	*1*	*2*
1			L					N

Далее нужно задать начало последовательности, например, {1, 2} и записать процедуру добавления в последовательность ещё одной цифры. Поскольку добавлять ту же цифру, что и предыдущая бессмысленно, можно воспользоваться одним из двух вариантов: добавлять по списку (1, 2, 3) или по формуле (`X[I]:=X[I-1] mod 3 + 1`).

Однако может случиться так, что в последовательность нельзя будет добавить ни одну из трёх цифр, например, {1, 2, 1, 3, 1, 2, 1, ?}. Потому что 1 приведет к повторению {1}-{1}; 2 приведёт к повторению {1, 2} – {1, 2}; а 3 приведёт к повторению {1, 2, 1, 3}-{1, 2, 1, 3}. На этом простой алгоритм может закончиться, но при анализе последовательности можно обнаружить, что если изменить одну из цифр раньше, то можно преодолеть этот барьер в 8 символов. Следовательно, в алгоритм надо добавить возможность изменения предыдущей цифры.

Причём, если её изменение тоже не помогает избавиться от повторения, то процедуру возврата следует повторить ещё раз. Здесь же можно записать ограничение, что при попытке изменить вторую цифру последовательности алгоритм должен закончить работу, поскольку искомая последовательность не может быть построена.

Таким образом, примерная схема работы программы может быть представлена в следующем виде:

// N – требуемый размер последовательности
// L – размер найденной последовательности
Пока (L < N)
    Если (Можно Добавить (X, L))
        L := L+1 // увеличить размер
    Иначе
        Пока (Убрать(X, L)) // убрать 1 последний символ
            Если (L>2)
                X[L] := 0 // только для наглядности
                L := L-1
            Иначе // последовательности такой длины не существует
                Закончить

Осталось записать реализацию функций добавления и удаления элемента.

Про первую мы уже писали, она достаточно простая и может выглядеть следующим образом

Функция Можно Добавить (Массив X, Размер L)
Для всех I := 1 to 3
    X[L+1] := I // Возможно следует добавить I
    Если Последовательность Верная (X, L)
        Вернуть Да // Добавление прошло успешно
Вернуть Нет // Нельзя добавить ни одной цифры

Функция убрать может просто изменять последнюю цифру на следующее возможное значение и если полученная последовательность правильная, возвращать результат, а в противном случае возвращать ошибку, так как никакую цифру поставить нельзя (Одна цифра уже стояла, вторая равна предыдущей и третью мы только что проверили). Один из возможных вариантов реализации представлен ниже

Функция Убрать (Массив X, Размер L)

X[L] := X[L]+1
Если X[L] = X[L-1])
X[L] := X[L]+1
Если X[L] > 3
Вернуть Ошибку
Вернуть Проверка (X, L)

Осталось реализовать только функцию Проверка, про которую мы писали в самом начале. Можно реализовать алгоритм перебора всех подпоследовательностей, как было описано вначале.

Функция Проверка (Массив X, Размер L)
Для всех возможных начал B от 1 до N-1
Для всех длин D от 1 до (N-B) \ 2
Fl := ДА // Наверное, подпоследовательности длины D совпадают
Для всех элементов подвектора I от 1 до D
Если X[B+I-1] не равно X[B+D+I-1])
Fl := Нет // есть различия! закончить цикл
Если Fl = Да
Вернуть Ошибка; // Нет
Вернуть Есть; // Да

Полученное простое решение будет реализовывать алгоритм проверки за время, пропорциональное $O(N^3)$. Это не позволит решить задачу при больших значениях N, поэтому алгоритм надо оптимизировать.

Одним из мест для оптимизации должна стать функция проверки, которая потребляет наибольшее время. При анализе работы можно заметить, что эта функция многократно проверяет уже обработанную часть вектора - от 1 до предпоследнего знака, которая не изменялась. Значит повторная подпоследовательность может заканчиваться только в последней позиции, и проверять следует с конца последовательности, и только заканчивающиеся в последней позиции подпоследовательности. Это приведёт к алгоритму проверки со временем $O(N^2)$.

Функция Проверка (Массив X, Размер L)
Для всех длин D от 1 до N \ 2

Fl := ДА // Наверное, подпоследовательности длины D совпадают

Если X[N-D+1] не равно X[N-D-I])

Fl := Нет // есть различия!

Если Fl = Да

Вернуть Ошибка; // Нет

Вернуть Есть; // Да

На этом примере были рассмотрены следующие методы разработки алгоритмов.

Уменьшение количества вариантов для перебора за счёт сохранения информации о предыдущей работе.

Реализация поиска с возвратом.

2.2. Задание. Максимальный подвектор

Вариант 1 O(N^3)

5	*-3*	*2*	*7*	*-5*	*-6*	*5*	*4*
		L	U				N

Очевидным решением этой задачи будет перебор всех пар чисел L и U, удовлетворяющих условию $1 \le L \le U \le N$. Для любой пары надо вычислить сумму элементов вектора X[L..U] и проверить, больше ли эта сумма, чем максимальная сумма, найденная до сих пор. Программа может иметь следующий вид.

```
Max := 0
For L := 1 to N do
      For U := L to N do
            Sum := 0
            For I := L to U do
                  Sum := Sum + X[I]
            If (Sum>Max) then
                  Max :=Sum
```

Эта программа короткая, простая и её легко понять. К несчастью, она медленно работает, поскольку в ней используется тройной цикл от размера вектора. Например, для N=1000 внутренний цикл будет выполняться более 100 000 000 раз.

Вариант 2 $O(N^2)$

Есть два очевидных способа увеличить его быстродействие, работающих за квадратичное время от размера, которые вычисляют сумму подвектора X[L..U] не за U-L+1 шагов, а за постоянное. Например, можно заметить, что сумма подвектора X[L..U] может быть вычислена на основе суммы предыдущего подвектора X[L..U-1] добавлением нового элемента X[U]. Это приводит ко второму варианту программы.

```
Max := 0
For L := 1 to N do
      Sum := 0
      For U := L to N do
            Sum := Sum + X[U]
            If (Sum > Max) then
                  Max := Sum
```

Также можно вычислить предварительно суммы во внутреннем цикле, обращаясь к структуре данных, созданной до начала выполнения внешнего цикла. В элементе I массива CumArray содержится кумулятивная сумма элементов вектора X[1..N], так что сумма элементов вектора X[L..U] может быть найдена с помощью вычисления разности CumArray[U]- CumArray[L-1]. Что приводит к следующему алгоритму.

```
CumArray[0] := 0
For I := 1 to N do
      CumArray[I] := CumArray[I-1] + X[I]
MaxSoFar := 0
For L := 1 to N do
      Sum :=CumArray[U] - CumArray[L-1]
      MaxSoFar := max (MaxSoFar, Sum)
```

Вариант 3 $O(N \log_2 N)$

Для последующего ускорения алгоритма можно воспользоваться методом «разделяй и властвуй», который основан на решении двух подзадач размерности N/2 и объединении результатов.

Если разбить исходный вектор на два подвектора A и B примерно одного размера и найти в них максимальные суммы MaxA и

MaxB, то решение лежит в одном из двух векторов A или B или пересекает границу между векторами A и B. В последнем случае назовём его MaxC. Таким образом, алгоритм должен вычислить рекурсивно суммы подвекторов A и B, вычислит сумму вектора C, а затем вернёт максимальное из этих значений.

Для маленьких векторов сумму вычислить легко. Если вектор пустой, то его значение равно 0. Если вектор состоит из 1 элемента, то его значение равно самому элементу, если он положительный, и равно 0 в противном случае

При вычислении MaxC можно заметить, что его компонент в A – это максимальный подвектор, начинающийся на границе и простирающийся в A, аналогично для его компонента в подвекторе B. Это приводит к следующему решению.

```
Answer :=MaxSum(1,N)
recursive function MaxSum(L, U)
If L > U then
     Return 0
If L = U then
     Return max(0, X[L])
M := (L+U)/2
// подвектор A - это X[L..M], а B - это [M+1..U]
Sum := 0
MaxSoLeft := 0
For I := M downto L do
     Sum := Sum + X[I]
     MaxSoLeft := max(MaxSoLeft, Sum)
Sum := 0
MaxSoRight := 0
For I := M+1 to U do
     Sum := Sum + X[I]
     MaxSoRightt := max(MaxSoLeft, Sum)
MaxCrossing :=MaxToLeft + MaxToRight$
MaxInA :=MaxSum(L, M)
MaxInB :=MaxSum(M+1, U)
Return max(MaxCrossing, MAxInA, MaxInB)
```

Эта программа работает за время пропорциональное $N \log_2 N$. При разбиении исходного вектора на три можно получить время про-

порциональное N $\log_3$ N и так далее, но накладные расходы на реализацию могут съесть весь выигрыш по времени, да и реализация станет заметно сложнее.

Вариант 4 O(N)

А сейчас мы рассмотрим простейший алгоритм, работающий с массивами: он начинает с левой границы массива (элемент X[1]) и осуществляет сканирование до его правой границы (элемент X[N]), отслеживая подвектор с максимальной суммой. В начальный момент максимум равен 0. Если мы уже нашли решение для подвектора X[1..I-1], то при добавлении элемента X[I] решение будет равно ранее найденной максимальной сумме для X[1..I-1] или максимальной сумме подвектора, заканчивающегося в позиции I. Таким образом, получим следующее решение.

```
MaxSoFar := 0
MaxEndingHere := 0
For L := 1 to N do
     MaxEndingHere := max (MaxEndingHee+X[I], 0);
     MaxSoFar := max (MaxSoFar, MaxEndingHere);
```

Ключом к пониманию этой программы является переменная MaxEndingHere. Перед первым оператором присваивания в цикле переменная MaxEndingHere содержит сумму элементов максимального подвектора, оканчивающегося в позиции I-1. Оператор присваивания изменяет её так, чтобы она содержала сумму элементов максимального подвектора, оканчивающегося в позиции I. Этот оператор увеличивает её на значение элемента X[I] до тех пор, пока она остаётся положительной; когда переменная становится отрицательной, оператор устанавливает её в 0, так как максимальный подвектор, оканчивающийся в позиции I, - пустой вектор. Эта самая короткая и быстрая программа, просматривающая массив только один раз. Более того, даже использовать массив не обязательно, поскольку элемент можно обработать сразу после ввода.

На этом примере были рассмотрены важные методы разработки алгоритмов.

Сохраняйте значения переменных, чтобы избежать повторных вычислений (варианты 2 и 4).

Проводите предварительную обработку информации для построения структур данных (вариант 2б).

Алгоритм «разделяй и властвуй» (вариант 3).

Сканирующий алгоритм. Если получено решение для вектора X[1..I-1], то как его можно расширить на вектор X[1..I]? (вариант 4)

Задача заимствована из книги Бентли Д. Жемчужины творчества программистов: Пер. с англ., - М.: Радио и связь, 1990

2.3. Задание. Площадь многоугольника

Данное задание рассчитано на знание теории геометрических разделов математики «Фигуры и их площади», которая не рассматривается в 7 классе. С данной оговоркой приведем здесь разбор задания «Площадь многоугольника», которое смогут выполнить желающие школьники.

Вариант 1

Пусть даны две точки: $T_1(x_1, y_1)$ и $T_2(x_2, y_2)$. Найдём площадь трапеции, образованной отрезком T_1T_2, перпендикулярами, опущенными из этих двух точек на ось OX, и проекцией этого отрезка на ось OX. Она равна $S = (x_1 - x_2)\dfrac{y_1 + y_2}{2}$ - высоте трапеции (x_1-x_2), умноженной на полусумму оснований $\dfrac{y_1 + y_2}{2}$. На самом деле, если нужно найти именно площадь трапеции, то нужно взять найденную величину по модулю – площадь отрицательной не бывает. Но для дальнейших рассуждений потребуется именно формула трапеций со знаком.

Площадь многоугольника со знаком равна $S_м=S_т(1, 2)+ S_т(2, 3)+\ldots+ S_т(N-1, N)+ S_т(N-1, 1)$, где $S_т(i, j)$ – площадь со знаком соответствующей трапеции для i-й и j-й точек многоугольника (см. Рис). Заметьте, что знак $S_т(i, j)$ зависит от того, $x_i < x_j$ или $x_i > x_j$. Приведённый рисунок позволяет убедиться в этом.

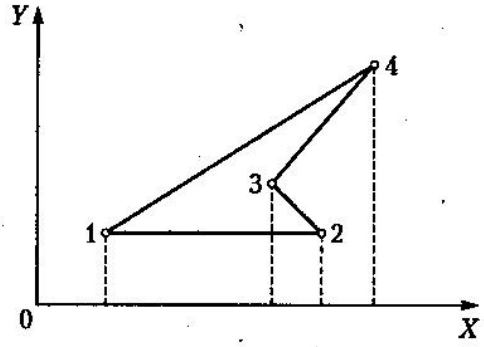


Рис 6. Вычисление площади многоугольника через площади трапеций.

Нужно не забыть перед выводом ответа взять полученную площадь по модулю, ведь координаты многоугольника во входных данных могли быть заданы как в порядке обхода по часовой стрелке, так и против неё.

Вариант 2

Для выпуклого многоугольника очевидна формула Sм=Sтр(1, 2, 3)+ Sтр(1, 3, 4)+…+ Sтр(1, N-2, N-1)+ Sтр(1, N-1, N), где Sт(i, j, k) – площадь треугольника, образованного i-й, j-й и k-й точками.

Для площади невыпуклого многоугольника справедлива эта же формула с одной поправкой: при переходе от i-й точки к j-й по часовой стрелке относительно первой площадь треугольника нужно добавлять, а при переходе против часовой стрелки – вычитать. (См. Рис)

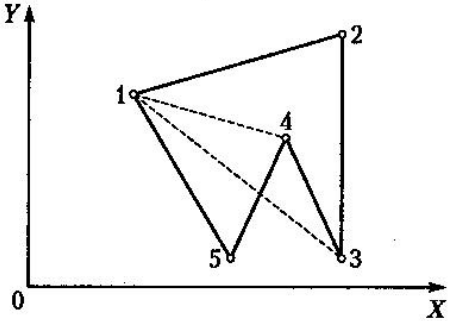


Рис 7. Вычисление площади многоугольника через площади треугольников

Для этих вариантов можно не считывать данные в память – для всех приведённых формул достаточно координат только нескольких точек – первой и двух последовательных. Ещё можно убрать за скобки деление на 2.

Оба примера дают полезные формулы для определения площадей фигур, что также можно использовать и в других задачах.

3. Домашнее задание

3.1. Задача. Орешки (*www.acmp.ru*)

Белочка собрала в лесу N шишек с орешками. Белочка очень привередливо выбирала шишки, и брала только те, в которых ровно M орешков. Также известно, что для пропитания зимой ей необходимо не менее K орешков. Определите, хватит ли на зиму орешков белочке.

Входные данные

Входной файл INPUT.TXT содержит три натуральных числа: N, M и K (N, M <= 100, K <= 10 000).

Выходные данные

В выходной файл OUTPUT.TXT выведите «YES» если белочке хватит орешков и «NO» в противном случае.

Примеры

INPUT.TXT	OUTPUT.TXT
4 5 20	YES
4 5 21	NO
3 2 1	YES

3.2. Задача. Напёрстки (*www.acmp.ru*)

Шулер показывает следующий трюк. Он имеет три одинаковых наперстка. Под первый (левый) он кладет маленький шарик. Затем он очень быстро выполняет ряд перемещений наперстков, каждое из которых – это одно из трех перемещений - A, B, C:

- A - обменять местами левый и центральный наперстки,
- B - обменять местами правый и центральный наперстки,
- C - обменять местами левый и правый наперстки.

Необходимо определить, под каким из наперстков окажется шарик после всех перемещений.

Входные данные

В единственной строке входного файла INPUT.TXT записана строка длиной не более 50 символов из множества {A, B, C} – последовательность перемещений.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести номер наперстка, под которым окажется шарик после перемещений.

Пример

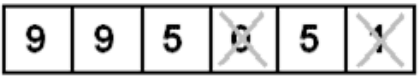
INPUT.TXT	OUTPUT.TXT
CBABCACCC	1

3.3. Задача. Странная лотерея (*www.acmp.ru*)

В честь успешного окончания первой четверти родители Пети и Вовы подарили им один лотерейный билет. Как обычно, ребята всерьез не восприняли данный подарок ввиду довольно скудной вероятности выигрыша по их мнению. Но какова было удивление братьев, когда они узнали из средств массовой информации, что номер их билета является выигрышным.

После продолжительных минут радости ребята бросились узнавать сумму выигрыша, но как оказалось, узнать это не так уж просто. Дело в том, что сумма выигрыша отчасти определяется самим владельцем или владельцами (если их несколько) следующим образом. На обратной стороне билета есть определенная секция, стерев слой защитного покрытия, можно увидеть целое положительное число N. После чего, каждый из владельцев билета должен зачеркнуть ровно по одной цифре данного числа N, полученное число и является суммой выигрыша.

Ниже приведен пример с N равным 995051.



Максимальное число, которое может быть получено из данного N посредством вычеркивания двух цифр является 9955. Помогите Пете и Вове в этой непростой, но очень актуальной для них задачей.

Входные данные

В единственной строке входного файла INPUT.TXT находится число N ($100 \le N < 10^{250}$). Число не содержит лидирующих нулей. Строка входного файла заканчивается переводом строки и не содержит символов отличных от "0"-"9".

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – максимальное значение, которое может быть получено из N посредством вычеркивания из него ровно двух цифр.

Пример

INPUT.TXT	OUTPUT.TXT
102	2
995051	9955

Модуль 6. Строковый тип данных. Обработка строк

Задачи модуля

1. Знакомство с символьным и строковым типом данных.
2. Тренировка техники работы с переменными строкового типа.
3. Рассмотрение примеров считывания строковой информации из файла.
4. Разбор олимпиадных задач.

1. Теоретический материал

1.1. Символьный тип данных

Переменные и константы символьного типа могут принимать значения из множества символов таблицы Ascii-кодов (American Standart Codingfor Information Interchange – Американский стандартный код для обмена информацией). Первая половина этой таблицы стандартизирована – стандартный раздел Ascii-кодов (символы с кодами 0-127), а вторая половина уникальна для каждой страны – расширенный раздел (символы с кодами 128-255). В нашей стране в расширенном разделе располагаются символы кириллицы (коды 128-175 и 224-241), символы псевдографики и ряд дополнительных символов.

Например, заглавным символам кириллицы ‘А’-’У’ соответствуют следующие коды:

код	сим вол	код	сим вол	код	сим вол	код	сим вол	код	сим вол
128	А	133	Е	138	К	143	П	143	П
129	Б	134	Ж	139	Л	144	Р	144	Р
130	В	135	З	140	М	145	С	145	С
131	Г	136	И	141	Н	146	Т	146	Т
132	Д	137	Й	142	О	147	У	147	У

Сравнение символов производится на основе таблицы Ascii – кодов. Например сравнение ‘М’>’Т’ ложно, а сравнение ’Й’<’Л’ истинно.

Определить код символа возможно при помощи функции Ord (s). Например, функция Ord(‘Д’) вернет код 132. Функция Chr (n), наоборот, возвращает символ, соответствующий коду *n*. Например,

функция Chr(147) возвратит символ ‘У’. Функция Pred(s) вернет символ, предыдущий символу s, а функция Succ(x), наоборот, последующий. Например, значением функции Pred(‘Г’) будет являться символ ‘В’.

1.2. Строковый тип данных

Строка представляет собой последовательность символов, заключенную в кавычки. Например, ‘Компьютер’, ‘Mycomputer’, ‘23’ – значения строкового типа.

Пустая строка изображается двумя одинарными кавычками – ‘’.

Синтаксис описания переменных строкового типа:

Var<Идентификатор>: string[<Максимальная длина строки>];

Указание длины строки при ее описании не является обязательным, при отсутствии такого указания максимальная длина строки принимается равной 255.

Например,

```
var  family: string[15];
name: string[10];
address: string;
```

Переменная family может хранить строку не более 15 символов, переменная name – не более 10 символов, переменная address – не более 255 символов.

К любому символу строки можно обратиться по номеру (индексу) символа в строке. То есть тип string можно понимать как составной тип данных, представляющий собой массив символов – array [0..225] of char. Элемент массива с индексом 0 хранит символ, код которого равен длине строки.

Например,

```
Var  book: string;
Begin
book:='Pascal';
writeln (ord(book[0]));
End.
```

После выполнения программы на экран будет выведено число символов в строке – 6.

Строки можно сравнивать, вводить, выводить и соединять. Сравнение строк производится в лексикографическом порядке. Соеди-

нение строк производится при помощи операции +. Например, результатом ‘Pas’+’cal’ будет являться строка ‘Pascal’.

Для работы со строками в языке Pascal предусмотрены стандартные процедуры и функции.

Функции	**Примеры**
Concat(s_1, ..., s_n : String): String Соединяет последовательно строки s_1, ..., s_n и возвращает полученное значение.	s1:= ‘Pedagogical’; s2:= ’university’; s3:=Concat(s1, ‘ ‘, s2) Результат – ‘Pedagogical university’
Length(s: String) : Integer Возвращает длину строки	Length(s1) Результат – 11.
Copy (s: String, i: integer, c :integer): String Возвращает подстроку строки *s* длиной *c*, начиная с позиции *i*.	Copy(s3, 13, 10) Результат – ’university’
Pos(s1: String, s2: String): Integer Возвращает номер начальной позиции первого вхождения подстроки *s1* в строку *s2*.	Pos(s2, s3) Результат – 13
Процедуры	**Примеры**
Delete(s: string, i: integer, c: integer) Удаляет с символов из строки s, начиная с позиции i.	Delete (s1, 8, 4); writeln (s2); Результат – ‘Pedagog’
Insert (s1: string, s2: string i: integer) Вставляет подстроку s1 в строку s2, начиная с позиции i.	Insert (‘Technical ‘, s2, 1); writeln (s2); Результат – ‘Technical university’

Str(x: числовой тип, s :string) Преобразует число x в строку s. Параметр x может содержать форматы преобразования (аналогично их использованию в процедуре write).	Str(pi:4:2, s) Результатом будет являться переменная s строкового типа, значение которой равно ‘3.14’.
Val(s: String, x : числовой тип) Преобразует строковое значение s в число x.	Year:=’1980’; Val (Year, x) Результат – переменная x = 1980.

1.3. Примеры алгоритмов обработки строк

Пример 1

Заменить в строке s каждую группу символов *s1* на *s2*.

```
Var   s, s1, s2: string;
      n, k: integer;
Begin
writeln(‘Введите строку, заменяемую группу символов, символы замены);
readln(s, s1, s2);
n:=length(s1);  {Количество символов в заменяемой группе}
k:=pos(s1, s);  {Позиция первого вхождения s1 в s}
while k<>0      {Повторение цикла, пока s1 входит в s}
      begin
      delete(s, k, n);  {Удаление в строке s с позиции k заменяемой группы}
      insert(s2, s, k); {Вставка новых символов на место удаленных}
      k:= pos(s1, s);  {Определение дальнейшей позии вхождения}
      end;
writeln (s);
End.
```

Пример 2

Описать функцию определения длины самой длинной цифровой последовательности, входящей в строку.

```
Function Max (var s: string): integer;
const m=['0'..'9'];        {Константа типа множе-
ство}
var k, kmax, i, n : integer; b: boolean;
begin
n:=length(s);
kmax:=0;   {Максимальное значение количества под-
ряд идущих цифр}
i:=1;      {Текущая позиция, с которой начинается
просмотр строки}
b:=true;  {Флаг символизирует, что строка еще не
кончилась}
while b do      {Пока строка не кончилась}
      begin
      k:=0;
      while (i<=n) and (s[i] in m) do
      {Подсчет количества подряд идущих цифр }
            begin
            i:=i+1; k:=k+1;
            end;
      if k>kmax then kmax:=k;
      {Пропустить не цифры}
      while (i<=n) and  not(s[i] in m) do
            i:=i+1;
      if i=n+1 then b:=false;
      end;
max:=kmax;
end;
```

2. Примеры задач

2.1. Задача. Восстановление скобок (Школьный этап всероссийской олимпиады школьников по информатике. Липецк 2011)

Составьте программу, печатающую все правильные скобочные выражения, которые можно восстановить из входной строки заменой

знаков вопроса на скобки, либо печатающую сообщение: восстановить невозможно.

Исходные данные - строка из круглых скобок и знаков вопроса.

Пример

INPUT.TXT	OUTPUT.TXT
((??)?	((())) (()())
)?	восстановить невозможно

Идея решения задачи

Идея решения заключается в вычислении, так называемого *баланса* скобочной последовательности. Если закодировать открывающиеся скобки единицами, а закрывающиеся как минус единица - «-1», то баланс вычисляется как сумма единиц и минус единиц. И если скобочное выражение составлено правильно, то баланс должен быть равен нулю. Причем для любой части скобочного выражения, начиная с первой скобки и заканчивая любой промежуточной, баланс должен быть больше или равен нулю. Иначе это выражение заведомо неверное. Приведем текст программы.

```
Var f1, f2:text;
    s:string;
    j,i,n,c,flag:integer;
    z:array [0..1000] of integer; {Массив единиц,
нулей и минус единиц}
    a:array [0..1000] of integer; {Массив позиций
знаков вопроса}

procedure pod(k:integer); {Процедура подстановки
вместо знака вопроса скобок}
{на вход процедуре подается к – индекс элемента
массива a}
var b,i1,i,l:integer;
begin
for i:=1 to 2 do {сначала вместо знака вопроса
ставим закрыв. скобку, потом открыв.}
     begin
     if (i=1) then l:=-1 else l:=1;  z[a[k]]:=l;
```

```
     b:=0;  for i1:=1 to a[k] do b:=b+z[i1]; {Вы-
числяем баланс}
     if  (b>=0) and  (k<a[0]) then pod(k+1); {если
баланс >=0, то вызываем рекурсивно еще раз функцию
pod}
     if (k=a[0]) and (b=0) and(z[z[0]]=-1) then
          begin
          flag:=1;
          for i1:=1 to z[0] do
               if z[i1]=-1 then
                    write(f2,chr(93),' ')
               else
                    write(f2,chr(91));
          writeln(f2);
          end;
     end;
end;

Begin
assign(f1,'input.txt');
assign(f2,'output.txt');
reset(f1); rewrite(f2);
read(f1,s); close(f1);
if s[1]=')' then
     begin
     writeln(f2,'восстановить        невозможно');
     close(f2); halt;
     end;
z[0]:=length(s);
a[0]:=0;
for i:=1 to length(s) do
     begin
     if s[i]='?' then
          begin
          z[i]:=0;   a[a[0]+1]:=i;   a[0]:=a[0]+1;
          end;
     if s[i]='(' then z[i]:=1;
```

```
      if s[i]=')' then z[i]:=-1;
      end;
flag:=0;
pod(1);
if flag=0 then    writeln(f2,'восстановить невоз-
можно');
close(f2);
End.
```

3. Домашнее задание

3.1. Задача. Восстановление скобок -2

Случилось вот что. Некто взял правильно записанное математическое выражение со скобками и выкинул из него все, кроме скобок:

(( ) (( ) ( )) (( )) ( ))

Затем Некто занялся такой операцией. Под каждой открывающей скобкой он записывает, сколько скобок (любых) заключено между ней и соответствующей ей закрывающей, а под закрывающими не пишет ничего:

```
 ( ( ) ( ( ) ( ) ) ( ( ) ) ( ) )
14 0  4 0  0   2 0   0
```

Получается ряд чисел: 14 0 4 0 0 2 0 0

Представьте, что Вам дан такой ряд чисел. Восстановите исходную последовательность скобок.

Входные данные

Последовательность целых неотрицательных чисел.

Выходные данные

Фраза - "решение есть", либо - "решения нет".

Если решение есть, то еще вывести последовательность скобок, соответствующая исходной последовательности.

3.2. Задача. Сторожа

В картинной галерее работают сторожа. Для каждого сторожа известно время прихода на работу и время ухода. Определить, всегда ли галерея охраняется.

Входные данные

Входной файл INPUT.TXT содержит целое число N (2 <= N <= 1000) - количество сторожей, а также T1(I), T2(I), I=1,...,N — моменты начала и окончания дежурства I-го сторожа;

Выходные данные

В выходной файл OUTPUT.TXT следует вывести «Yes», если галерея всегда охранялась и «No» в противном случае.

Примеры

INPUT.TXT	OUTPUT.TXT
4 8.00 12.00 11.00 16.00 15.00 19.30 20.00 23.50	No

3.3. Задача. Повторяющийся элемент

Вывести элемент, встречающийся в одномерном массиве чаще других.

Входные данные

В входном файле INPUT.TXT первая строка содержит целое число N (2 <= N <= 1000) - количество элементов одномерного массива. Во второй строке через пробел идут N целых чисел, по модулю не превосходящих 10^3 - элементы последовательности.

Выходные данные

В выходной файл OUTPUT.TXT следует вывести элемент, встречающийся в одномерном массиве чаще других.

Примеры

INPUT.TXT	OUTPUT.TXT
10 1 4 2 2 3 1 2 3 2 1	2

Модуль 7. Календарные задачи

Задачи модуля
1. Знакомство с календарными задачами.
2. Изучение теории, необходимой для решения календарных задач.
3. Разбор олимпиадных задач, содержащих информацию о календаре.

1. Теоретический материал

1.1. Знакомство с календарными задачами

Рассмотрим задачу «Ежедневник», решение которой содержит в себе необходимые подходы к решению календарных задач.

Задача В. Ежедневник (Турнир Архимеда 2011 г.)

Васе подарили два ежедневника на i-й год. Один он использовал в i-м году и теперь интересуется, когда наступит следующий год с точно таким же календарем, чтобы он мог воспользоваться вторым ежедневником.

Входные данные

Вводится одно натуральное число i, не превышающее 2011.

Выходные данные

Выведите одно число – номер года, когда можно будет использовать второй ежегодник.

Примеры

Ввод	Вывод
2011	2022
1	7

Для решения такого рода задач необходимо знать понятие високосного года. Високосный год – это такой год, в котором 366 дней – на одни сутки больше, чем в обычном не високосном годе. Високосным считается год, номер которого кратен 4 и не кратен 100, либо кратен 400. *Примеры*: 2010 - не високосный год; 2008 - високосный; 2000 - високосный; 1900 – не високосный. Лишний день в високосном году добавляется к февралю месяцу, продолжительность которого становится равной тогда не 28, а 29 дням.

Решим сначала вспомогательную задачу: научимся по номеру года определять, сколько в нем дней. Для удобства напишем соответствующую функцию:

```
function days(n:integer):integer;
if ((n  mod 4) = 0) and ((n mod  100 ) <> 0) or
((n  mod 400) = 0) then
     res:=366
else
     res:= 365;
```

Теперь нам нужно найти ближайший год, следующий за i-м, удовлетворяющий двум условиям:

1) в нем должно быть столько же дней, сколько и в i-м;
2) он должен начинаться в тот же день недели, то есть количество дней в годах с i-го до года, предшествующего искомому, должно делиться на 7.

Будем перебирать года i+1, i+2, … до тех пор, пока не встретим год, удовлетворяющий обоим условиям:

```
m := i;
sum: = days(i);
while ((sum mod 7) <> 0) or (days(i) <> days(m +
1)) then
     begin
     m :=m+ 1;
     sum : sum + days(m);
     end;
writeln(m + 1);
```

Приведем фрагмент программы

```
Var n, i, sum : integer;

function days(n:integer):integer;
begin
if ((n mod 4 = 0) and (n mod 100 <> 0)) or (n mod
400 = 0) then
     result := 366
else
```

```
     result := 365
end;

Begin
read(n);
sum := days(n);
i := n;
while (days(n) <> days(i+1)) or (sum mod 7 <> 0) do
     begin
     inc(i);
     sum := sum + days(i);
     end;
write(i+1);
End.
```

2. Домашнее задание

2.1. Задача. Даты *(www.acmp.ru)*

При обработке различной информации часто приходится оперировать с данными о датах. В этой задаче вам нужно составить программу, которая сможет вывести число, месяц и день недели, которые наступят через K дней, начиная с первого января 2008 года. Напомним, что это был вторник.

Входные данные

Входной файл INPUT.TXT содержит одно целое число K (0 <= K <= 1000) - количество дней, после первого января, через которое наступит (или наступила) интересующая дата.

Выходные данные

В выходной файл OUTPUT.TXT ваша программа должна записать, какие день недели, число и месяц наступят по прошествии заданного времени. Результаты проверяются автоматически, поэтому вам следует придерживаться формата, показанного в примерах.

Примеры

INPUT.TXT	OUTPUT.TXT
0	Tuesday, 01.01
5	Sunday, 06.01

Примечание: По-английски дни недели называются так: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday.

2.2. Задача. День недели

Напишите программу, определяющую по дате день недели.

Входные данные

Входной файл INPUT.TXT содержит строку вида DD.MM.YYYY– дата.

Выходные данные

В выходной файл OUTPUT.TXT ваша программа должна записать, какой день недели будет в этот день.

Примеры

INPUT.TXT	OUTPUT.TXT
02.06.1997	Monday
27.02.2015	Friday

2.3. Задача. Количество дней

Напишите программу, определяющую количество дней между двумя заданными датами.

Входные данные

Входной файл INPUT.TXT содержит две строки вида DD.MM.YYYY– дата.

Выходные данные

В выходной файл OUTPUT.TXT ваша программа должна записать количество дней между ними.

Примеры

INPUT.TXT	OUTPUT.TXT
04.11.2010 13.12.2010	39
16.02.1820 01.09.1833	4946

Модуль 8. Длинная арифметика

Задачи модуля

1. Знакомство с целочисленными типами данных.
2. Знакомство с примерами, показывающими невозможность описать данные задачи стандартными типами.
3. Разбор задач на длинную арифметику.

1. Теоретический материал

Pascal относится к языкам со строгой типизацией. Это означает, что тип всех переменных и констант должен быть предварительно описан. *Тип* – это множество значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими значениями. Простые (скалярные) типы языка можно разделить на две группы: *стандартные* типы и *определяемые пользователем*.

К стандартным типам языка Pascal относятся, в частности, целый и вещественный тип. Каждый из этих типов характеризуется диапазоном представления чисел и размером выделяемой для них памяти.

1.1. Группа целых типов

Группа целых типов			
Название	Идентификатор	Диапазон	Размер памяти
Короткое целое со знаком Целое со знаком Длинное целое со знаком Короткое целое без знака Целое без знака	Shortint Integer Longint Byte Word	-128..127 -32768..32767 -2147483648..2147483647 0..255 0..65535	1 байт 2 байта 4 байта 1 байт 2 байта
Форма записи: Записываются в виде последовательности цифр со знаком или без него, например, 4, -234, 3288.			

Допустимые операции: +, –, *, div (деление нацело), mod (остаток от деления). Все операции возвращают результат целого типа. Определено и обычное деление /. Возвращает частное вещественного типа. Над элементами целых типов определены операции сравнения.

1.2. Группа вещественных типов

Группа вещественных типов				
Название	Идентификатор	Диапазон	Значащие цифры мантиссы	Размер памяти
Вещественное одинарной точности	Single	от $1.5*10^{-45}$ до $3.4*10^{38}$	7..8	4 байта
Вещественное	Real	от $2.9*10^{-39}$ до $1.7*10^{38}$	11..12	6 байтов
Вещественное двойной точности	Double	от $5.0*10^{-324}$ до $1.7*10^{308}$	15..16	8 байтов
Вещественное повышенной точности	Extended	от $3.4*10^{-4932}$ до $1.1*10^{4932}$	19..20	10 байтов
Целое в формате вещественного	Comp	от $-2^{63}+1$ до $2^{63}-1$	19..20	8 байтов
Форма записи: Изображаются с десятичной точкой: 12.3, 34.77, -0.34 или в показательной форме: -0.4E5, 6.7E-10.				
Допустимые операции: +, -, *, /. Результат будет иметь также вещественный тип. Над элементами вещественных типов определены операции сравнения.				

Внимательно изучите таблицу целых типов данных. Давайте порассуждаем. Если в задаче требуется ввести переменную, в которой будут храниться номера месяцев, то переменную можно объявить типом Shortint или Byte. Если речь идет о больших целых числах, то потребуется описание переменной типом Integer или LongInt. Обратим внимание, что диапазон целых чисел типа Integer -32768..32767, то есть переменная данного типа может содержать до пяти цифр и не превышать числа 32767. Если же в задаче будут использованы числа свыше 32767, то необходим тип данных LongInt, диапазон которого -2147483648..2147483647. В языке программирования Delphi существует тип данных int64, диапазон которого еще больше.

А теперь рассмотрим задачу, в которой требуется сложить два числа, которые могут содержать более десяти цифр. Каким типом данных в языке Pascal вы можете описать данные числа? Такие числа не помещаются ни в один из стандартных типов языка. В этом случае и нужен навык оперирования длинными числами.

2. Задача. Сумма двух длинных целых чисел

Если в условии задачи речь идет о числах, которые не могут быть описаны стандартными целыми типами, то необходимо применять алгоритмы работы с длинными числами. Рассмотрим такую задачу.

Снова A+B

Требуется сложить два целых A и B.

Входные данные

Во входном файле INPUT.TXT записано два неотрицательных целых числа, не превышающих 10^{100}, по одному в каждой строке.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести одно целое число – сумму чисел A и B, без лидирующих нулей.

Пример

INPUT.TXT	OUTPUT.TXT
2 3	5

Сложение двух длинных чисел выполняется с использованием тех же принципов, что и при поразрядном сложении вручную на листе бумаги. Считаем сначала числа в массивы *a* и *b*, после чего осуществим сложение таким образом, что результат суммы запишется в массив a. В переменной *c* будем хранить перенос на следующий разряд.

Для хранения длинного числа можно использовать целочисленный массив, где в качестве элемента массива будет одна цифра числа. В 1-м элементе массива будем хранить последнюю цифру числа, во 2-м - предпоследнюю и т.д. до последней цифры. В 0-м элементе можно хранить общее количество цифр в числе. В простейшем случае, для хранения числа 154 достаточно будет использовать следующую запись:

```
const maxsize=100;
var a:array[1..maxsize];
a[0]=3; a[1]=4; a[2]=5; a[3]=1;
```

Идея реализации всех необходимых операций (сложение, вычитание, умножение, деление и т.д.) основана на тех принципах, которыми мы пользуемся при расчетах на бумаге. Даже когда мы реализуем деление "в столбик", фактически мы работаем с небольшими числами, сводя более сложную задачу к набору из более простых подзадач.

Практически все задачи на длинную арифметику требуют чтения из файла и запись в файл длинных чисел, поэтому приведем реализацию этих функций в данном разделе, а в разборе задач будем их использовать:

```
procedure readlong(a:mas)
var i:integer;
    s: string;
begin
readln(s);
a[0]=length(s); {в нулевой элемент массива записа-
ли длину числа}
for i:=1 to a[0] do
     a[i]=ord(s[i])-48; {Считали элементы числа в
виде строки по одной цифре в каждый элемент масси-
ва}
```

```
procedure writelong(a:mas)
for i:=a[0] to 1 do write(a[i]);
```

Так же часто возникает необходимость сравнивать длинные числа. Опишем следующую функцию, которая будет возвращать 0 в случае равенства чисел, -1 когда первое число меньше второго и 1 когда первое больше:

```
function complong(a,b:mas):byte
var i:integer;
begin
if (a[0] < b[0]) then res:= -1;
if (a[0] > b[0]) then res:=1;
for i:=a[0] to 1 do
     begin
     if(a[i] < b[i]) then res:= -1;
     if(a[i] > b[i]) then res:= 1;
     end;
res:= 0;
end;
```

В простейшем случае решение поставленной задачи представимо следующим алгоритмом на языке С (желающие могут переписать эту программу на язык Паскаль):

```
const maxsize=102;
int a[maxsize], b[maxsize];
readlong(a);readlong(b);
m=max(a[0],b[0]);c=0;
for i=1..m{
     c = c+a[i]+b[i];
     a[i] = c mod 10;
     c = c div 10;
}
if(c>0){
     m=m+1;
     a[m] = c;
}
a[0]=m;
writelong(a);
```

Попробуйте самостоятельно решить задачу суммы двух длинных целых чисел и реализовать ее на языке Pascal.

3. Домашнее задание

3.1. Задача. A-B

Требуется найти разность между неотрицательными числами A и B.

Входные данные

Во входном файле INPUT.TXT в двух строках записаны два неотрицательных целых числа A и B, не превышающие 10^{100}.

Выходные данные

В выходной файл OUTPUT.TXT выведите значение A-B.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	7 5	2
2	5 17	-12

3.2. Задача. A*B

Требуется найти произведение между неотрицательными числами A и B.

Входные данные

Во входном файле INPUT.TXT в двух строках записаны два неотрицательных целых числа A и B, не превышающие 10^{100}.

Выходные данные

В выходной файл OUTPUT.TXT выведите значение A*B.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	7 5	35
2	5 17	85

3.3. Задача. 2^N

Необходимо вычислить значение 2^n.

Входные данные

В единственной строке входного файла INPUT.TXT записано натуральное число n ($0 < n < 1000$).

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести значение 2n.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3	8
2	10	1024
3	72	4722366482869645213696

Модуль 9. Рекурсивный подход в программировании

Задачи модуля

1. Знакомство с понятием рекурсии.
2. Знакомство с примерами, показывающими возможность применения рекурсии.
3. Разбор задач на рекурсивный подход в программировании.

1. Теоретический материал

1.1. Знакомство с понятием рекурсии

Рекурсивным называется объект, который частично определяется через самого себя. Идеи рекурсии известны людям издавна. Рекурсивные определения как мощный аналитический аппарат используются во многих областях, особенно в математике. Рассмотрим функцию факториала n!. Как правило, ее определяют как произведение первых n целых чисел: n!=1*2*3*…*n.

Такое произведение можно легко вычислить с помощью итеративных конструкций, например, оператора цикла for.

```
Program factorial;
Var   Fact: longint;
      n,i: integer;
Begin
Write ('Введите число n'); Readln(n);
Fact:=1;
For i:=1 to n do Fact:=Fact*i;
Writeln('Факториал n!=',Fact);
End.
```

Однако существует другое определение факториала, в котором используется рекуррентная формула и которое имеет такой вид:

(1) 0!=1

(2) Для каждого n>0 n!=n*(n-1)!

Определения с помощью рекуррентных формул иногда называют рекурсивными определениями. Если для факториала первое (итеративное) определение может показаться проще, то для чисел Фибоначчи рекурсивное определение дает более удобный способ вычисления.

(1) F(1) =1,
(2) F(2)=1,
(3) Для каждого n>2 F(n)=F(n-1)+F(n-2)

Использование рекурсии позволяет легко запрограммировать вычисления по рекуррентным формулам. Например, программа, использующая рекурсивную функцию для вычисления факториала n! имеет следующий вид.

```
Program Factorial;
Var N: integer;

Function Fact(i: integer): Longint;
begin
     If i=1 then Fact:=1 else Fact:=i*Fact(i-1)
end;

Begin
Write('Введите число n'); readln (n);
Writeln('Факториал n!=', Fact(n));
End.
```

Для программирования рекурсии используют процедуры и функции, которые вызывают из своего тела сами себя.

2. Задачи, решаемые с использованием рекурсивного подхода

Для программирования рекурсии используют процедуры и функции, которые вызывают из своего тела сами себя. Рассмотрим задачу «Монетки».

2.1. Задача. Монетки

В волшебной стране используются монетки достоинством A_1, A_2,…, A_M. Волшебный человечек пришёл в магазин и обнаружил, что у него есть ровно по две монетки каждого достоинства. Ему нужно заплатить сумму N. Напишите программу, определяющую, сможет ли он расплатиться без сдачи.

Входные данные

Во входном файле INPUT.TXT записано сначала число N ($1 \le N \le 10^9$), затем число M($1 \le M \le 15$)и далее M попарно различных чисел A_1, A_2,…, A_M ($1 \le A_i \le 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите количество монет, которое придётся отдать волшебному человечку, если он сможет заплатить указанную сумму без сдачи. Если решений несколько, выведите вариант, в котором волшебный человечек отдаст наименьшее возможное количество монет. Если без сдачи не обойтись, то выведите одно число 0. Если же у волшебного человечка не хватит денег, чтобы заплатить указанную сумму, выведите одно число -1 (минус один).

Примеры

INPUT.TXT	OUTPUT.TXT
5 2 1 2	3
7 2 1 2	-1
5 2 3 4	0

```
Program monetki;
Const m1=15;
Type mas=array[1..m1] of longint;
Var a:mas;n,sum:longint; f1,f2:text;
    k,min,i,j,m:integer;

Procedure p; {процедура, которая будет вызывать
сама себя рекурсивно}
var j:integer;
begin
for j:=0 to 2 do {монет какого-либо достоинства
может не быть вообще, может быть одна или две}
     begin
     if i>=m then break; {если индекс уже больше,
чем количество достоинств монет}
     i:=i+1; {берем монету следующего достоинства}
     sum:= sum+j*a[i]; {считаем сумму денег, при-
бавляя к текущей j монет достоинства a[i]}
```

```
      k:=k+j; {считаем количество монет, которые
набирают подсчитанную сумму денег}
      if (sum=n)and(k<min) then min:=k;{если вычис-
ленная сумма равна требуемой и количество монет
наименьшее из ранее определенных, то запоминаем в
переменной min это количество}
      p; {вызываем процедуру рекурсивно из самой
себя еще раз. Тем самым берем монету следующего
достоинства }
      sum:=sum-j*a[i]; {на рекурсивном возврате
(отступая на шаг назад) возвращаем значение суммы
в прежнее значение}
      i:=i-1; {переходим к монете предыдущего до-
стоинства}
      k:=k-(j);{количество монет уменьшаем на коли-
чество монет текущего шага}
      end;
end;

{начинается сама программа}
Begin
assign(f1,'input.txt');
assign(f2,'output.txt');
reset(f1); rewrite(f2);
readln(f1,n,m);
for i:=1 to m do
      begin
      read(f1,a[i]); sum:=sum+2*a[i];
      end; {сразу считаем полную сумму монет при
считывании данных}
if sum<n then
      begin writeln(f2,-1);
      close(f1);close(f2);exit;
      end
else
      for i:=1 to m-1 do {сортируем  монеты по их
достоинству по убыванию}
```

```
            for j:=1 to m-1 do
                 if a[j]<a[j+1] then
                      begin k:=a[j]; a[j]:=a[j+1];
                      a[j+1]:=k;
                      end;
sum:=0; min:=32;k:=0;i:=0;
p; {вызываем рекурсивно процедуру}
if min<32 then
     writeln(f2,min)
else
     writeln(f2,0) ;
close(f1);close(f2)
End.
```

3. Домашнее задание

3.1. Задача. Числа Фибоначчи

Последовательностью Фибоначчи называется последовательность чисел $a_0, a_1, \ldots, a_n, \ldots$, *где* $a_0=0$, $a_1=1$, $a_k=a_{k-1}+a_{k-2}$ $(k>1)$.

Требуется найти N-е число Фибоначчи.

Входные данные

Во входном файле INPUT.TXT записано целое число N ($0 \le N \le 30$)

Выходные данные

В выходной файл OUTPUT.TXT выведите N-е число Фибоначчи.

Пример

INPUT.TXT	OUTPUT.TXT
7	13

3.2. Задача. Факториал

Факториал числа — это число, умноженное на «себя минус один», затем на «себя минус два» и так далее, до единицы. Обозначается $n! = n*(n-1)*(n-2)*...*1$.

Обратим внимание, что $n!$ можно записать как $n * (n-1)!$, например $3! = 3*(2*1) = 3*2!$

Требуется найти факториал числа N.

Входные данные

Во входном файле INPUT.TXT записано целое число N ($0 \le N \le 30$)

Выходные данные

В выходной файл OUTPUT.TXT выведите N!

Пример

INPUT.TXT	OUTPUT.TXT
6	720

3.3. Задача. Подарки Деда Мороза

Ириска весит X грамм, мандарин – Y грамм, пряник – Z грамм. Требуется написать программу, которая определит, сколько различных вариантов подарков весом ровно W грамм может сделать Дед Мороз.

Входные данные

В единственной строке входного файла INPUT.TXT содержится четыре целых числа X, Y, Z и W ($1 \le X, Y, Z \le 100$, $1 \le W \le 1000$).

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одно целое число – количество вариантов подарков.

Пример

INPUT.TXT	OUTPUT.TXT
10 25 15 40	3

Модуль 10. Геометрические задачи

Задачи модуля

1. Знакомство с олимпиадными задачами геометрической тематики. Формализация модели задачи.
2. Применение математических методов к решению задач.
3. Развитие интереса к задачам геометрической тематики, как к обязательно входящим в состав задач пакета олимпиад по программированию.

1. Теоретический материал

1.1. Примеры геометрических задач. Формализация модели задачи

Геометрия - область математики, известная каждому школьнику. Геометрические задачи довольно часто встречаются на олимпиадах по программированию: обычно, одну такую задачи организаторы олимпиад обязательно включают в набор для разнообразия. Однако, ошибочно считать, что в таких задачах используется исключительно школьная база знаний предмета "Геометрия" и программная реализация должна быть несложной. Для того, чтобы решить задачи геометрической тематики одних школьных знаний по геометрии будет недостаточно, будет хорошим заделом изучение вами вычислительных геометрических формул, которые можно найти в любом справочнике по геометрии.

Как правило, геометрические задачи подразумевают работу со следующими графическими объектами: точки, прямые, отрезки, вектора, отрезки, многоугольники и окружности. Для многих вычислений используются тригонометрические функции, встроенные в языки программирования, в которых углы обычно задаются в радианах. Так же следует отметить, что часто сложности при программировании возникают с неверным выбором типов данных и погрешностями.

Приведем перечень несложных программных элементов, которые наиболее часто используются при реализации геометрических задач:

- построение прямой по двум точкам
- построение точки пересечения двух прямых

- построение прямой, перпендикулярной (параллельной) данной, и проходящей через заданную точку
- проверка принадлежности точки отрезку
- проверка принадлежности точки многоугольнику
- вычисление площади треугольника
- вычисление площади многоугольника
- построение выпуклой оболочки
- вычисление угла между векторами
- вычисление скалярного, векторного и смешанного произведений
- сложение и вычитание векторов, умножение вектора на число.

Данные алгоритмы нужно уметь быстро и безошибочно реализовывать, как составную часть более сложной задачи.

Рассмотрим некоторые задачи геометрической тематики.

2. Примеры задач

2.1. Задача. Принадлежность точки кругу

Дан круг (X0, Y0, R) и точка (X, Y), где X0, Y0, R, X, Y - вещественные числа. Определить, лежит ли эта точка внутри данного круга. X0, Y0 – координаты центра круга, R – радиус круга.

Пример

(0, 0, 1) и (0.5, 0.5)

Результат: TRUE (точка принадлежит кругу)

Вариант решения

Напомним, что если координаты точки подставить в левую часть уравнения окружности и получится величина меньшая, чем радиус в квадрате, то данная точка будет принадлежать кругу. В противном случае точка не принадлежит кругу.

```
Code (Pascal):
Var X0, Y0, R, X, Y: Real;
Begin
ReadLn (X0, Y0, R, X, Y);
WriteLn (Sqr (X - X0) + Sqr (Y - Y0) <= Sqr (R));
End.
```

2.2. Задача. Принадлежность точки прямоугольнику

Дан прямоугольник, заданный двумя противоположными вершинами (X1, Y1); (X2, Y2) и точка (X, Y). Все Xi, Yi - вещественные числа. Определить, лежит ли эта точка внутри данного прямоугольника.

Пример

(1, 1); (7, 4) и (4, 2)

Результат: TRUE (точка принадлежит прямоугольнику)

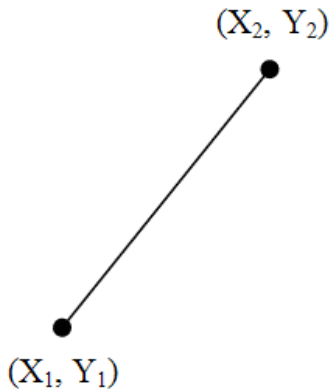
Вариант решения

Напомним, что точка принадлежит прямоугольнику если ее координат X расположена между координатами X1 и X2 точек прямоугольника. Аналогично и координата Y.

```
Code (Pascal):
var X1, Y1, X2, Y2, X, Y: Real;
Begin
ReadLn (X1, Y1, X2, Y2, X, Y);
WriteLn ((Abs (X1 - X2) = Abs (X1 - X) + Abs (X2 - X))
and (Abs (Y1 - Y2) = Abs (Y1 - Y) + Abs (Y2 - Y)));
End.
```

2.3. Задача. Длина отрезка

Отрезок задан координатами своих концевых точек. Требуется вычислить длину этого отрезка.



Входные данные

Входной файл INPUT.TXT содержит координаты концов отрезка в формате X_1 Y_1 X_2 Y_2 . Все координаты – целые числа, не превышающие 1000 по абсолютной величине.

Выходные данные

В выходной файл OUTPUT.TXT выведите длину отрезка с точностью не хуже чем 10^{-5}.

Пример

INPUT.TXT	OUTPUT.TXT
3 4 8 4	5

```
Var x1,y1,x2,y2,y,x:integer;
Begin
assign(input,'input.txt');
assign(output,'output.txt');
reset(input);rewrite(output);
read(x1,y1,x2,y2); close(input);
x:=abs(x1-x2);y:=abs(y1-y2);
if (y=0) or (x=0) then
     write(y+x)
else
     write(sqrt(y*y+x*x):0:5);
close(output);
End.
```

3. Домашнее задание

3.1. Задача. Две окружности

На плоскости даны две окружности. Требуется проверить, пересекаются ли они.

Входные данные

Входной файл INPUT.TXT состоит из двух строк. На каждой строке записана информация об одной окружности – координаты ее центра x и y (целые числа, по модулю не превосходящие 5000) и радиус (целое число $1 \le r \le 1000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите «YES», если окружности пересекаются, и «NO» в противном случае.

Примеры

INPUT.TXT	OUTPUT.TXT
0 0 2 0 3 2	YES
1 1 1 4 4 1	NO

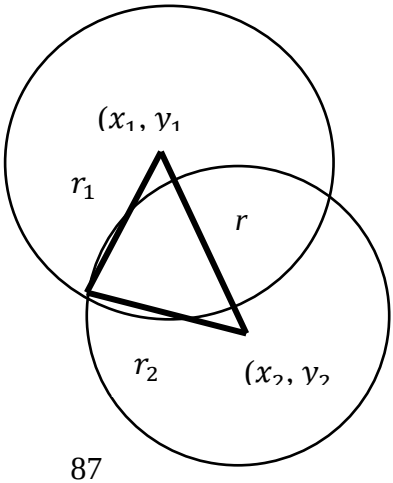
Комментарии к решению. Две окружности

Для начала определимся с тем, что нам известны радиусы окружностей и они равны r1 и r2. Так же по формуле расстояния между точками мы можем вычислить расстояние между центрами данных окружностей:

$r = sqrt((x2-x1)^2 + (y2-y1)^2)$

Заметим так же, что окружности будут пересекаться тогда и только тогда, когда возможен треугольник со сторонами r1, r2 и r. Фигуру, две стороны которой лежат на третьей или одна из сторон имеет нулевую длину так же будем считать треугольником, т.к. окружности могут друг друга касаться(r=r1+r2), либо полностью совпадать (r=0).

Треугольник считается возможным если сумма двух любых его сторон не меньше третьей. Т.е. в нашем случае достаточно проверить, что r1+r2>=r и r+r2>=r1 и r+r1>=r2. При этом желательно использовать вещественные типы данных. Так же можно провести аккуратное сравнение с учетом возможных погрешностей при вычислениях.



3.2. Задача. Треугольник

В декартовой системе координат на плоскости заданы координаты вершин треугольника и еще одной точки. Требуется написать программу, определяющую, принадлежит ли эта точка треугольнику.

Входные данные

В четырех строках входного файла INPUT.TXT находятся пары целых чисел - координаты точек. Числа в первых трех строках - это координаты вершин треугольника (x_1,y_1), (x_2,y_2), (x_3,y_3), в четвертой строке - координаты тестируемой точки (x_4,y_4). Все координаты не превышают 10000 по абсолютной величине.

Выходные данные

В выходной файл OUTPUT.TXT необходимо вывести слово «In», если точка находится внутри треугольника и «Out» в противном случае.

Пример

INPUT.TXT	OUTPUT.TXT
0 0 100 0 0 100 100 100	Out
0 0 100 0 0 100 10 10	In
0 0 100 0 0 100 50 50	In

3.3. Задача. Ферзь и король

Вася продолжает заниматься шахматами, и теперь он изучает различные окончания. Оказалось, что компьютер очень удобно использовать для анализа позиций. Теперь Вася просит вас написать программу, которая сможет определить, что черный король находится под шахом.

Для начала Васе подойдет программа, которая анализирует игровую ситуацию с тем предположением, что на доске находятся три фигуры - белые король и ферзь и черный король. Черный король находится под шахом, если белый ферзь может за один ход попасть на занимаемую им клетку. Шахматный ферзь может перемещаться по вертикали, горизонтали или диагонали, но, в отличие от коня, не может “перепрыгивать” через другие фигуры.

Входные данные

В единственной строке входного файла INPUT.TXT записаны обозначения трех клеток шахматной доски, разделенные пробелами: положения белого короля, белого ферзя и черного короля соответственно. При этом гарантируется, что черный и белый короли не находятся на соседних клетках.

Выходные данные

В выходной файл OUTPUT.TXT выведите слово YES, если черный король находится под шахом и NO, если шаха нет.

Пример

INPUT.TXT	OUTPUT.TXT
d3 d5 d8	YES
a1 a8 b3	NO

Модуль 11. Динамическое программирование

Задачи модуля

1. Знакомство с основными понятиями динамического программирования.
2. Изучение задач, при решении которых возможно использование динамического программирования.
3. Разбор задач, решаемых методом динамического программирования.

1. Теоретический материал

1.1. Введение в динамическое программирование

Динамическое программирование – это метод программирования, который позволяет свести решение основной задачи к решению меньшей подзадачи.

Пусть имеется N объектов N={1,2,3, …., N}.
Необходимо выбрать K объектов из данных N объектов. Например, K=3. В этом случае мы можем составить набор из трех объектов {1,2,3} или набор {1,3,4} и т.д. Задача заключается в том, чтобы найти *количество* указанных всевозможных наборов из трех элементов. Заметим, что наборы {1,2,3} и {2,3,1} одинаковы (поскольку отличаются только перестановкой объектов).

Математическая формула для вычисления сочетаний из N по K позволяет вычислить искомое количество наборов $C_N^K = \dfrac{N!}{K!(N-K)!}$

Данная формула не применима для больших N и K, так как вычисление факториалов больших чисел может привести к переполнению целого типа данных.

Рассмотрим решение данной задачи методом динамического программирования. Для этого сведем решение исходной задачи к решению подзадачи. Предположим, что мы знаем ответ для меньших N и K. Как в этом случае можно найти решение для последующих N и K?

Опишем рекуррентную формулу для вычисления сочетаний $C_N^K = C_{N-1}^K + C_{N-1}^{K-1}$. Эта формулу можно объяснить так: если мы не выбираем последний k-й элемент, то число сочетаний по (k) будет равняться C_{N-1}^K- выбираем из (N-1)-го оставшегося объекта. Или мы выбираем k-й элемент. Это описывается слагаемым C_{N-1}^{K-1}.

Для решения задачи объявим массив M[0..N,0..K] из N строк и K столбцов – максимальных значений N и K. Положим, что в задаче нужно вычислить C_N^K, где N<=5, K<=5. В этом случае объявим массив M[0..5,0..5]. Каждая ячейка матрицы M с индексами i,j – это значение C_i^j.

0	0	0	0	0	0
0	1	0	0	0	0
0	2		0	0	0
0	3			0	0
0	4				0
0	5				

Так как $C_N^0 = 0$, то нулевой столбец заполнен нулями. Так как $C_0^k = 0$, то нулевая строка заполнена нулями. Треугольник матрицы выше главной диагонали представлен нулевыми элементами, так как $C_N^K = 0, при \quad k \geq n$. M[1,1]=1, так как $C_N^N = 1$. Первый столбец заполнен, исходя из свойства $C_N^1 = N$. Используя формулу $C_N^K = C_{N-1}^K + C_{N-1}^{K-1}$ можно вычислить значение в каждой оставшейся ячейке матрицы.

Приведем текст программы решения поставленной задачи.

```
Program cnm;
Const nmax=5; kmax=5;
Var mem:array [0..nmax,0..kmax] of integer;
    i,j,n,k:integer;
Begin
```

```
writeln('n=? k=?'); readln(n,k);
if k>n then begin writeln(0); exit; end;
mem[1,1]:=1;
for i:=2 to n do
     begin
     mem[i,1]:=i;
     for j:=2 to k do
          mem[i,j]:=mem[i-1,j-1]+mem[i-1,j];
     end;
writeln(mem[n,k]);
End.
```

2. Разбор задачи, решаемой методом динамического программирования

2.1. Задача. Поход

Группа школьников решила сходить в поход вдоль Москвы-реки. У Москвы-реки существует множество притоков, которые могут впадать в неё как с правого, так и с левого берега.

Школьники хотят начать поход в некоторой точке на левом берегу и закончить поход в некоторой точке на правом берегу, возможно, переправляясь через реки несколько раз. Как известно, переправа как через реку, так и через приток представляет собой определённую сложность, поэтому они хотят минимизировать число совершённых переправ.

Школьники заранее изучили карту и записали, в какой последовательности в Москву-реку впадают притоки на всём их маршруте.

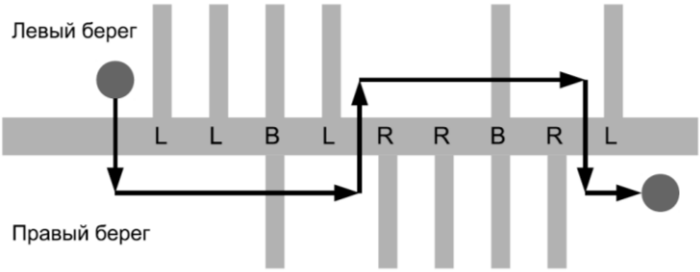
Помогите школьникам по данному описанию притоков определить минимальное количество переправ, которое им придётся совершить во время похода.

Единственная строка содержит описание Москвы-реки между начальной и конечной точкой похода. Длина строки не превосходит 10^5 символов.

Каждый символ строки может быть одной из трёх латинских букв *L*, *R* или *B*. Буква *L* означает, что очередной приток впадает реку с левого берега, *R* – приток впадает в реку с правого берега и *B* – притоки впадают с обоих берегов реки в одном месте. Поход начинается на левом берегу перед описанной частью реки и заканчивается на правом берегу после описанной части.

В качестве результата сдайте файл с одним целым числом – минимальным количеством переправ.

Рисунок, соответствующий примеру:



Входные данные	Результат
LLBLRRBRL	5

Решение задачи

Идея применения метода динамического программирования в данной задаче заключается в том, что анализируется ситуация – как лучше пройти через два близлежащих притока.

```
// written by Pham duc Giam. 2012
#include <iostream>
#include <fstream>
#include <string>

using namespace std;
#define N 100010
char c;
string s;
int x[N],y[N],u[N],v[N],a[N],b[N],i,j,n;
```

```
void main()
{
	//притоки делят берег реки на n+1 отрезок
	//x[i] - минимальное число переправ, необхо-
димое чтобы попасть в i-тый отрезок на левом бере-
гу
	//u[i] - предыдущий отрезок на левом берегу,
от которого до отрезка i, нет ни одного притока на
левом берегу
	//y[i] - минимальное число переправ, необхо-
димое чтобы попасть в i-тый отрезок на правом бе-
регу
	//v[i] - предыдущий отрезок на правом берегу,
от которого до отрезка i, нет ни одного притока на
правом берегу
	ifstream ifs("b2.in");
	ofstream ofs("b2.out");
	x[0] = 0;  //школьники уже на левом берегу,
следовательно x[0] = 0
	y[0] = 1;  //y[0] = 1 т.к нужна 1 переправа
	u[0] = v[0] = 0;
	i = 1;
	ifs >> s;
	n = s.length();
	for(i=1;i<=n;i++)
	{
		c = s[i-1];
		//если нет притоков между соседними от-
резками
		x[i] = x[i-1];
		y[i] = y[i-1];
		u[i] = u[i-1];
		v[i] = v[i-1];
		if(c=='L')
		{
			x[i] = x[u[i-1]]+1;   //школьники
должны пересечь приток на левом берегу
```

```
                u[i] = i;
            }
            else if(c=='R')
            {
                y[i] = y[v[i-1]]+1;   //школьники
должны пересечь приток на правом берегу
      v[i] = i;
            }
            else
            {
                //школьники должны пересечь прито-
            ки на левом и правом берегах
                x[i] = x[u[i-1]]+1;
                y[i] = y[v[i-1]]+1;
                u[i] = v[i] = i;
            }
            if(x[i-1]==0)
                x[i] = 1;
            if(y[i-1]==1)
                y[i] = 2;
            if(x[i]+1<y[i])
                y[i] = x[i]+1;  //школьникам   вы-
годно попасть на правый берег с левого
            if(y[i]+1<x[i])
                x[i] = y[i]+1; //школьникам выгод-
но попасть на левый берег с правого
      }
      cout << y[n] << endl;
}
```

3. Домашнее задание

3.1. Задача. Максимальная подпоследовательность (www.acmp.ru)

Дана числовая последовательность, требуется найти длину наибольшей возрастающей последовательности.

Входные данные

В первой строке входного файла INPUT.TXT записано число N – длина последовательности ($1 \le N \le 1000$). Во второй строке записана

сама последовательность (через пробел). Числа последовательности – целые числа, не превосходящие 10000 по модулю.

Выходные данные

В выходной файл OUTPUT.TXT требуется вывести наибольшую длину возрастающей последовательности.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	6 3 29 5 5 28 6	3

3.2. Задача. Пицца

Пицца – любимое лакомство Васи, он постоянно покупает и с удовольствием употребляет различные сорта этого великолепного блюда. Однажды, в очередной раз, разрезая круглую пиццу на несколько частей, Вася задумался: на какое максимальное количество частей можно разрезать пиццу за N прямых разрезов?

Помогите Васе решить эту задачу, определив максимальное число не обязательно равных кусков, которые может получить Вася, разрезая пиццу таким образом.

Входные данные

Входной файл INPUT.TXT содержит натуральное число N – число прямых разрезов пиццы (N <= 1000).

Выходные данные

В выходной файл OUTPUT.TXT выведите ответ на задачу.

Пример

INPUT.TXT	OUTPUT.TXT
2	4
3	7

3.3. Задача. Ферзь в угол

Рассмотрим бесконечную вправо и вверх шахматную доску, на которой стоит ферзь. Двое по очереди двигают этого ферзя. Разрешается двигать ферзя только вниз, влево или по диагонали вниз влево. Цель игры – задвинуть ферзя в угол, то есть клетку с координатами (1, 1). На рисунке показаны разрешенные движения ферзя.

Требуется написать программу, которая найдет номер игрока, который выиграет при правильной игре.

Входные данные

Входной файл INPUT.TXT содержит координаты ферзя перед первым ходом - два числа M и N, записанные через пробел (1 <= M, N <= 250).

Выходные данные

Выходной файл OUTPUT.TXT должен содержать найденный номер победителя.

Пример

INPUT.TXT	OUTPUT.TXT
3 2	2
6 7	1

Модуль 12. Задачи регионального этапа Всероссийской олимпиады школьников по информатике 2012 г

Задачи модуля

1. Знакомство с задачами ВОШ по информатике регионального этапа.
2. Анализ материалов регионального этапа ВОШ по информатике.
3. Разбор задач, предлагаемых на региональном этапе ВОШ по информатике.

1. Примеры задач

1.1. Задача. Длина отрезка

Петя и Маша пришли в зоопарк. Больше всего Пете понравились цапли. Он был поражен их способностью спать на одной ноге.

В вольере находятся несколько цапель. Некоторые из них стоят на двух ногах, некоторые — на одной. Когда цапля стоит на одной ноге, то другую ее ногу не видно. Петя пересчитал видимые ноги всех цапель, и у него получилось число a.

Через несколько минут к вольеру подошла Маша. За это время некоторые цапли могли поменять позу, поэтому Петя предложил ей заново пересчитать видимые ноги цапель. Когда Маша это сделала, у нее получилось число b.

Выйдя из зоопарка, Петя с Машей заинтересовались, сколько же всего цапель было в вольере. Вскоре ребята поняли, что однозначно определить это число можно не всегда. Теперь они хотят понять, какое минимальное и какое максимальное количество цапель могло быть в вольере.

Требуется написать программу, которая по заданным числам a и b выведет минимальное и максимальное количество цапель, которое могло быть в вольере.

Входные данные

Входной файл содержит два целых числа a и b, разделенных ровно одним пробелом ($1 \le a \le 10^9$, $1 \le b \le 10^9$).

Выходные данные

Выведите в выходной файл два целых числа, разделенных пробелом — минимальное и максимальное число цапель, которое

могло быть в вольере. Гарантируется, что хотя бы одно количество цапель соответствует условию задачи.

Примеры

herons.in	herons.out
3 4	2 3

Комментарий

В приведенном примере возможны следующие варианты:

1) В вольере две цапли. Когда Петя считал ноги, одна цапля стояла на двух ногах, а другая — на одной. Петя насчитал три ноги. Когда Маша считала ноги, обе цапли стояли на двух ногах, Маша насчитала четыре ноги.

2) В вольере три цапли. Когда Петя считал ноги, все цапли стояли на одной ноге, Петя насчитал три ноги. Когда Маша считала ноги, одна цапля стояла на двух ногах, а еще две — на одной. Маша насчитала четыре ноги.

1.2. Задача. Круглый стол

Возрождая древние традиции английских рыцарей, в одном городе члены школьного клуба любителей информатики каждую неделю собираются за круглым столом и обсуждают результаты последних соревнований**.**

Руководитель клуба Иван Петрович недавно заметил, что не все ребята активно участвуют в обсуждении. Понаблюдав за несколькими заседаниями клуба, он заметил, что активность члена клуба зависит от того, кто с кем сидит рядом.

В клуб приходят на занятия m мальчиков и n девочек. Иван Петрович заметил, что мальчик активно участвует в обсуждении только тогда, когда непосредственно рядом с ним с обеих сторон от него сидят девочки, а девочка активно участвует в обсуждении только тогда, когда непосредственно рядом с ней с одной стороны от нее сидит мальчик, а с другой — девочка.

Желая сделать заседание клуба как можно более интересным, Иван Петрович решил разместить участников за круглым столом таким образом, чтобы как можно больше членов клуба приняло активное участие в обсуждении.

Требуется написать программу, которая по заданным числам m и n выведет такой способ размещения m мальчиков и n девочек за круглым столом, при котором максимальное количество членов клуба будет активно участвовать в обсуждении.

Входные данные

Входной файл содержит два целых числа m и n, разделенных ровно одним пробелом ($0 \le m \le 1000$, $0 \le n \le 1000$, $m + n \ge 3$).

Выходные данные

Выходной файл должен содержать строку с расположенными в некотором порядке m символами «B» (заглавная латинская буква) и n символами «G» (заглавная латинская буква). Символ «B» означает мальчика, а символ «G» — девочку.

Символы следует расположить в том порядке, в котором нужно разместить членов клуба вокруг стола. Соседние символы соответствуют членам клуба, которые сидят рядом. Рядом сидят также члены клуба, соответствующие первому и последнему символу выведенной строки.

Примеры

table.in	table.out
1 2	BGG
2 2	BGBG

Комментарий

В первом примере все члены клуба примут активное участие в обсуждении.

Во втором примере мальчики примут активное участие в обсуждении, а девочки нет. В этом примере можно также разместить членов клуба следующим образом: «BBGG». В этом случае активное участие в обсуждении примут обе девочки, а мальчики — нет. Разместить всех так, чтобы три или четыре члена клуба приняли активное участие в обсуждении, нельзя.

1.3. Задача. Повреждённый XML

Формат XML является распространенным способом обмена данными между различными программами. Недавно программист Иванов написал небольшую программу, которая сохраняет некоторую важную информацию в виде XML-строки.

XML-строка состоит из открывающих и закрывающих тегов.

Открывающий тег начинается с открывающей угловой скобки (<), за ней следует имя тега — непустая строка из строчных букв латинского алфавита, а затем закрывающая угловая скобка (>). Примеры открывающих тегов: <a>, <dog>.

Закрывающий тег начинается с открывающей угловой скобки, за ней следует прямой слеш (/), затем имя тега — непустая строка из строчных букв латинского алфавита, а затем закрывающая угловая скобка. Примеры закрывающихся тегов: , </dog>.

XML-строка называется *корректной*, если она может быть получена по следующим правилам:

- Пустая строка является корректной XML-строкой.
- Если A и B — корректные XML-строки, то строка AB, получающаяся приписыванием строки B в конец строки A, также является корректной XML-строкой.
- Если A — корректная XML-строка, то строка <X>A</X>, получающаяся приписыванием в начало A открывающегося тега, а в конец — закрывающегося с таким же именем, также является корректной XML-строкой. Здесь X — любая непустая строка из строчных букв латинского алфавита.

Например, представленные ниже строки:

<a>

<a><ab></ab><c></c>

<a><a><a>

являются корректными XML-строками, а такие строки как:

<a>

<a>

<a>

не являются корректными XML-строками.

Иванов отправил файл с сохраненной XML-строкой по электронной почте своему коллеге Петрову. Однако, к сожалению, файл повредился в процессе пересылки: ровно один символ в строке заменился на некоторый другой символ.

Требуется написать программу, которая по строке, которую получил Петров, восстановит исходную XML-строку, которую отправлял Иванов.

Входные данные

Входной файл содержит одну строку, которая заменой ровно одного символа может быть превращена в корректную XML-строку. Длина строки лежит в пределах от 7 до 1000, включительно. Строка содержит только строчные буквы латинского алфавита и символы «<» (ASCII код 60), «>»(ASCII код 62) и «/»(ASCII код 47).

Строка во входном файле заканчивается переводом строки.

Выходные данные

Выходной файл должен содержать корректную XML-строку, которая может быть получена из строки во входном файле заменой ровно одного символа на другой. Если вариантов ответа несколько, можно вывести любой.

Примеры

xml.in	xml.out
<a>	<a>
<a><aa>	<a>
<a><>a>	<a>
<a/	<a>

1.4. Задача. Игра с числами

Сегодня на уроке математики Петя и Вася изучали понятие арифметической прогрессии. Арифметической прогрессией с разностью d называется последовательность чисел a1, a2, …, ak, в которой разность между любыми двумя последовательными числами равна d. Например, последовательность 2, 5, 8, 11 является арифметической прогрессией с разностью 3.

После урока Петя и Вася придумали новую игру с числами. Игра проходит следующим образом.

В корзине находятся n фишек, на которых написаны различные целые числа a_1, a_2, …, a_n. По ходу игры игроки выкладывают фишки из корзины на стол. Петя и Вася делают ходы по очереди, первым ходит Петя. Ход состоит в том, что игрок берет одну фишку из корзины и выкладывает ее на стол. Игрок может сам решить, какую фишку взять. После этого он должен назвать целое число $d \geq 2$ такое, что все числа на выбранных к данному моменту фишках являются членами некоторой арифметической прогрессии с разностью d, не обязательно последовательными. Например, если на столе выложены фишки с числами 2, 8 и 11, то можно назвать число 3, поскольку эти числа являются членами приведенной в начале условия арифметической прогрессии с разностью 3.

Игрок проигрывает, если он не может сделать ход из-за отсутствия фишек в корзине или из-за того, что добавление любой фишки из корзины на стол приводит к тому, что он не сможет подобрать соответствующее число d.

Например, если в корзине имеются числа 2, 3, 5 и 7, то Петя может выиграть. Для этого ему необходимо первым ходом выложить на стол число 3. После первого хода у него много вариантов назвать число d, например он может назвать $d = 3$. Теперь у Васи два варианта хода.

1) Вася может вторым ходом выложить фишку с числом 5 и назвать $d = 2$. Тогда Петя выкладывает фишку с числом 7, называя $d = 2$. На столе оказываются фишки с числами 3, 5 и 7, а в корзине осталась только фишка с числом 2. Вася не может ее выложить, поскольку после этого он не сможет назвать корректное число d. В этом случае Вася проигрывает.
2) Вася может вторым ходом выложить фишку с числом 7 и также назвать, например, $d = 2$. Тогда Петя выкладывает фишку с числом 5, называя также $d = 2$. Вася снова попадает в ситуацию, когда на столе оказываются фишки с числами 3, 5 и 7, а в корзине осталась только фишка с числом 2, и он также проигрывает.

Заметим, что любой другой первый ход Пети приводит к его проигрышу. Если он выкладывает число 2, то Вася отвечает числом 7,

и Петя не может выложить ни одной фишки. Если Петя выкладывает фишку с числом 5 или 7, то Вася выкладывает фишку с числом 2, и у Пети также нет допустимого хода.

Требуется написать программу, которая по заданному количеству фишек n и числам на фишках $a_1, a_2, \ldots, a_n$ определяет, сможет ли Петя выиграть вне зависимости от действий Васи, и находит все возможные первые ходы Пети, ведущие к выигрышу.

Входные данные

Первая строка входного файла содержит целое число n ($1 \le n \le 200$).

Вторая строка содержит n различных целых чисел $a_1, a_2, \ldots, a_n$ (для всех i от 1 до n выполняется неравенство $1 \le a_i \le 10^5$). Соседние числа разделены ровно одним пробелом.

Выходные данные

Первая строка выходного файла должна содержать число k — количество различных первых ходов, которые может сделать Петя, чтобы выиграть. Если Вася может выиграть вне зависимости от действий Пети, строка должна содержать цифру 0.

Во второй строке должно содержаться k различных целых чисел — все выигрышные числа. Будем называть число выигрышным, если, выложив в качестве первого хода фишку, содержащую это число, Петя может выиграть вне зависимости от действий Васи. Соседние числа в строке должны быть разделены ровно одним пробелом.

Примеры

game.in	game.out
4 2 3 5 7	1 3
2 2 4	0

Комментарий

Первый пример рассматривается в тексте условия этой задачи.

Во втором примере, какую бы фишку не выложил Петя первым ходом, Вася в ответ выкладывает другую фишку, и Петя не может сделать ход из-за отсутствия фишек в корзине.

Рекомендуемая литература

1. Андреева Е.В., Босова Л.Л., Фалина И.Н. Математические основы информатики. Элективный курс: Учебное пособие. – М.: БИНОМ. Лаборатория Знаний, 2007.
2. Босова Л.Л., Босова А.Ю., Коломенская Ю.Г. Занимательные задачи по информатике. – М.: БИНОМ. Лаборатория знаний. 2007.
3. Великович Л.С., Цветкова М.С. Программирование для начинающих. – М.: БИНОМ. Лаборатория знаний. 2007.
4. Долинский М.С. Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач: Учебное пособие. – СПб.: Питер Принт, 2004.
5. Долинский М.С. Решение сложных и олимпиадных задач по программированию. – СПб.: Питер, 2006.
6. Златопольский Д. М. Программирование: типовые задачи, алгоритмы, методы. – М.: БИНОМ. Лаборатория знаний, 2007.
7. Иванов С.Ю., Кирюхин В.М., Окулов С. М. Методика анализа сложных задач по информатике: от простого к сложному. // Информатика и образование. 2006. №10.
8. Кирюхин В.М. Всероссийская олимпиада школьников по информатике. М.: АПК и ППРО, 2005.
9. Кирюхин В.М. Методика проведения и подготовки к участию в олимпиадах по информатике: всероссийская олимпиада школьников. – М.: БИНОМ. Лаборатория знаний, 2012.
10. Кирюхин В.М., Окулов С.М. Методика решения задач по информатике. Международные олимпиады. – М.: БИНОМ. Лаборатория знаний, 2007.
11. Меньшиков Ф.В. Олимпиадные задачи по программированию. – СПб.: Питер, 2006.
12. Окулов С.М. Основы программирования. – М.: БИНОМ. Лаборатория знаний, 2005.

13. Окулов С.М. Программирование в алгоритмах. – М.: БИНОМ. Лаборатория знаний. 2002.
14. Пупышев В.В. 128 задач по началам программирования. – М.: БИНОМ. Лаборатория знаний. 2009.
15. Скиена С.С., Ревилла М.А. Олимпиадные задачи по программированию. Руководство по подготовке к соревнованиям. – М.: Кудиц-образ, 2005.
16. Сулейманов Р.Р. Организация внеклассной работы в школьном клубе программистов: методическое пособие. – М.: БИНОМ. Лаборатория знаний. 2010.

Сайты с возможностью тестирования задач олимпиадного программирования

1. Турниры Архимеда – http://www.arhimedes.org/
2. Московская окружная олимпиада – http://www.olympiads.ru/
3. Олимпиады по информатике. Санкт-Петербург – http://neerc.ifmo.ru/school/ioip/
4. Школа программиста – http://acmp.ru/
5. Codeforces – http://www.codeforces.ru/

Учебное пособие

Информатика 7-8

Учебно-методическое пособие

Автор-составитель Инесса Анатольевна Шуйкова

Компьютерная верстка и техническое
редактирование: И.А. Шуйкова, Е.А. Зайцев