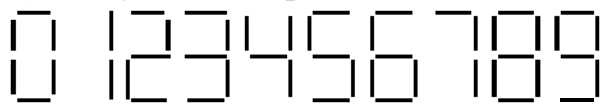
1. Хранитель времени

В электронных часах время отображается на табло в виде последовательности цифр, указывающих часы (от 0 до 23), минуты (от 0 до 59) и секунды (от 0 до 59). Например:

Каждая отдельная цифра на табло отображается в виде светящихся сегментов (отрезков) следующим образом:



Часы потребляют тем больше энергии, чем больше сегментов используется в записи времени. Написать программу, которая определяет время, когда часы потребляют наибольшее и наименьшее количество энергии.

Входные данные: В программу вводится три значения начального времени и три значения конечного времени. Значение вида “0” приравнивается к “00”, “7” к “07” и т.д.

Выходные данные: Время наибольшего и наименьшего потребления энергии часами.

Примеры:

№	Входные данные	Выходные данные
1	21 30 0 23 45 1	22 8 8 21 41 11
2	7 15 0 11 15 59	8 8 8 11 11 11

Решение:

```
program Task1;
{$APPTYPE CONSOLE}

uses
  SysUtils;


const
```

```
  { Перевод цифры в число сегментов. Например, в цифре 0 сегментов 6,
    поэтому a[0]=6 }
  DigitToSegmentsCnt: array [0..9] of Integer = (6, 2, 5, 5, 4, 5, 6, 3, 7,
6);


type
  { Время }
  TTime = record
    Hours, Minutes, Seconds: Integer;
  end;


{ +1 секунда к текущему времени }
procedure IncTime(var Time: TTime);
begin
  with Time do begin
    Inc(Seconds);

    { Проверка переполнения секунд }
    if Seconds >= 60 then begin
      Seconds := 0;

      Inc(Minutes);

      { Проверка переполнения минут }
      if Minutes >= 60 then begin
        Minutes := 0;

        Inc(Hours);

        { Проверка переполнения часов }
        if Hours >= 24 then
          Hours := 0;
      end;
    end;
  end;
end;


{ Расчет числа горящих сегментов по текущему времени }
function TimeToSegmentsCnt(const Time: TTime): Integer;
var
  Digits: array [1..6] of Integer;    { Цифры, отображаемые на циферблате }
  I: Integer;
begin
  { Определение цифр, горящих на циферблате }
  with Time do begin
    Digits[1] := Hours div 10;
    Digits[2] := Hours mod 10;

    Digits[3] := Minutes div 10;
    Digits[4] := Minutes mod 10;

    Digits[5] := Seconds div 10;
    Digits[6] := Seconds mod 10;
  end;

  { Подсчет числа горящих сегментов }
  Result := 0;
```

```
  for I := 1 to 6 do
    Result := Result + DigitToSegmentsCnt[ Digits[I] ];
end;


{ Совпадают ли указанные времена }
function IsEqualTime(const Time1, Time2: TTime): Boolean;
begin
  Result := (Time1.Hours = Time2.Hours) and (Time1.Minutes = Time2.Minutes)
      and (Time1.Seconds = Time2.Seconds);
end;


var
  T1, T2, MaxTime, MinTime: TTime;
  MaxSegments, MinSegments, CurrSegments: Integer;

begin
  { Ввод начального времени }
  Readln(T1.Hours, T1.Minutes, T1.Seconds);

  { Ввод конечного времени }
  Readln(T2.Hours, T2.Minutes, T2.Seconds);

  { Считаем, что входные данные корректны, проверку не делаем. При желании
    можно сделать... }

  { Инкремент времени T1 до совпадения с T2, поиск минимального и
    максимального числа горящих сегментов }
  MaxSegments := TimeToSegmentsCnt(T1);
  MinSegments := MaxSegments;
  MaxTime := T1;
  MinTime := T1;

  while not IsEqualTime(T1, T2) do begin
    IncTime(T1);

    CurrSegments := TimeToSegmentsCnt(T1);

    if CurrSegments > MaxSegments then begin
      MaxSegments := CurrSegments;
      MaxTime := T1;
    end;

    if CurrSegments < MinSegments then begin
      MinSegments := CurrSegments;
      MinTime := T1;
    end;
  end;

  { Вывод результата }
  Writeln(MaxTime.Hours, ' ', MaxTime.Minutes, ' ', MaxTime.Seconds);
  Writeln(MinTime.Hours, ' ', MinTime.Minutes, ' ', MinTime.Seconds);

  Readln;
end.
```

2. Аве Цезарь

Для передачи шифрованных сообщений наши генералы использовали следующий алгоритм: все коды символов в сообщении увеличивались на определенную величину *n*, с каждым новым символом сообщения, *n* увеличивается на единицу. В сообщениях использовалась таблица символов ASCII, символы от #65 до #122. Если в сумме с *n*, код символа выходил за пределы диапазона, то отсчет продолжался от символа #65. Пример: #121 + 5 = #68.

Мы не знаем начальное значение *n*, но знаем, что в каждом зашифрованном сообщении есть последовательность **“Ave_Caesar”**.

Необходимо разработать программу для декодирования зашифрованного текста и определения начального значения n.

Входные данные: В программу вводится последовательность символов длиной от 0 до 255, представляющая собой закодированный текст.

Выходные данные: Необходимо вывести значение начального смещения *n* и декодированный текст.

Примеры:

№	Входные данные	Выходные данные
1	EAkfKjoDmEmBEIAMOMEvRDyTCOYYGU\	4 Ave_Caesar_morituri_te_salutant
2	VsqesCloEkBACysMLsVRBw\AFUDV	3 Sol_lucet_omnibus_Ave_Caesar

Решение:

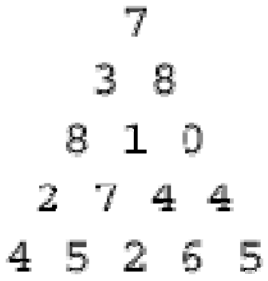
```
program Task2;
{$APPTYPE CONSOLE}

uses
  SysUtils;


var
  EncodedString, DecodedString: String;


function DecodeString(SrcString: String; Key: Integer): String;
var
  I, Code: Integer;
```

```
begin
  SetLength(Result, Length(SrcString));

  for I := 1 to Length(SrcString) do begin
    Code := Byte(SrcString[I]) - Key;

    if Code < 65 then
      Code := Code - (- 122 + 65 - 1);

    Result[I] := Char(Code);

    Inc(Key);
  end;
end;


var
  n: Integer;
  Found: Boolean;


begin
  { Чтение исходной закодированной строки }
  Readln(EncodedString);

  Found := False;

  { Подбор смещения n (с запасом, итераций можно сделать и меньше) }
  for n := 0 to 256 do begin
    DecodedString := DecodeString(EncodedString, n);

    { Проверка наличия искомой подстроки и декодированной строке }
    if Pos('Ave_Caesar', DecodedString) <> 0 then begin
      Writeln(n);
      Writeln(DecodedString);

      Found := True;

      break;
    end;
  end;

  { Декодирование не увенчалось успехом? }
  if not Found then
    Writeln('Can''t decode');

  Readln;
end.
```

3. Финансовая пирамида

На рисунке изображен треугольник из чисел. Вычислите наибольшую сумму чисел, расположенных на пути, начинающемся в **заданной точке** треугольника и заканчивающемся на его основании.

Каждый шаг на пути может идти вниз по диагонали влево или вниз по диагонали вправо.



Входные данные: Введите число строк в треугольнике [0..100], введите значения элементов треугольника [0..99], введите координаты стартовой точки в формате {номер строки} {позиция в строке}, нумерация начинается с нуля.

Выходные данные: Выведите наибольшую сумму элементов пути из заданной точки к основанию.

Пример:

Входные данные	Выходные данные
5 7 3 8 8 1 0 2 7 4 4 4 5 2 6 5 1 0	23

Решение:

```
program Task3;
{$APPTYPE CONSOLE}

uses
```

```
  SysUtils, Math;

var
  N: Integer;                                   { Число строк }
  A, L: array [0..100, 0..100] of Integer;      { Значения элементов }
  X, Y: Integer;                                { Координаты стартовой позиции
}
  F: Text;
  I, J, d1, d2, MaxValue: Integer;

begin
  { Ввод исходных данных (на этот раз из файла) }
  AssignFile(F, 'input.txt');
  Reset(F);

  Readln(F, N);

  for I := 0 to N-1 do begin
    for J := 0 to I do
      Read(F, A[I][J]);

    Readln(F);
  end;

  Readln(F, X, Y);

  CloseFile(F);

  { Сброс текущих длин путей из точки [X,Y] во все расположенные ниже нее }
  for I := Low(L) to High(L) do
    for J := Low(L[I]) to High(L[I]) do
      L[I][J] := 0;

  { Единственная ненулевая длина — из точки [X,Y] }
  L[X][Y] := A[X][Y];

  { Волновой алгоритм — проход по треугольнику сверху вниз }
  for I := X+1 to N-1 do
    for J := 0 to I do begin
      { В позицию [I,J] можно попасть либо из [I-1,J-1], либо из [I-1,J] }

      { Проверка того, чтобы не выйти за пределы массива/треугольника }
      if J-1 >= 0 then
        d1 := L[I-1][J-1]
      else
        d1 := -1;

      { Аналогичную проверку тут можно не делать, т.к. справа в матрице
        стоят нули }
      d2 := L[I-1][J];

      { d1 — расстояние при движении в текущую точку сверху слева,
        d2 — сверху справа, выбираем максимальное из них,
        добавляем к текущему }
      L[I][J] := Max(d1, d2) + A[I][J];
    end;

  { Выбор наибольшего значения из нижней строки }
  MaxValue := L[N-1][0];

  for I := 1 to N-1 do
```

```
    if L[N-1][I] > MaxValue then
      MaxValue := L[N-1][I];

  { Вывод результата на экран }
  Writeln(MaxValue);

  Readln;
end.
```