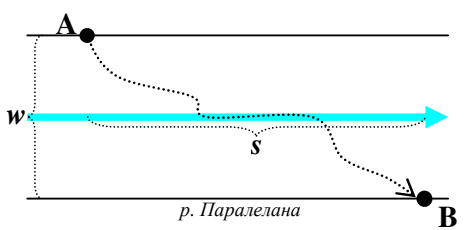
```
#include <iostream>
using namespace std;
void check(int k)
{
   int a; cin >> a; int b=0;
   int i=2; int c;
   while(i<=k)
   {
      cin >> c;
      if(c>a) a=c; else if(c>b) b=c;
      else
      {  i++; while(i<=k){cin >> c; i++;}
         cout << 0; return;
      }
      i++;
   }
   cout << 1;
}
int main()
{
   int n; cin >> n;
   for(int i=1;i<=n;i++)
   { int k; cin >> k; check(k); }
   cout << endl;
}
```

Задача С2. ПРЕСИЧАНЕ, автор Павлин Пеев

Река Паралелана почти замръзна! Преди беше лесно да стигнете с кануто си от коя да е точка A на единия бряг до коя да е точка B, разположена на другия бряг и надолу по течението. Сега по средата е останало водно пространство точно колкото за едно кану. Това все пак е нещо, но нали, за да стигнете до водата, се налага да влачите кануто по леда, а като излезете от водата – да го влачите отново до крайната точка? А в този студ ви се иска да стигнете колкото може по-бързо! Както се вижда от картата на местността, реката в този участък е с прави, успоредни брегове. Широчината ѝ е w км, а хоризонталното преместване от A до B е s км. Вашата скорост по леда е a км/ч, а във водата – b км/ч, като, естествено, $a < b$. Напишете програма **CROSSING**, която да намира минималното необходимо време за придвижване от точка A до точка B.



Вход

На единствения ред на стандартния вход са зададени, разделени с интервали, числата w, s, a и b, в тази последователност. Ако някое от числата има дробна част, тя е разделена от цялата част с точка и е с не повече от два десетични знака. Числата са положителни, като $w < s \le 40$, а $a < b \le 20$.

Изход

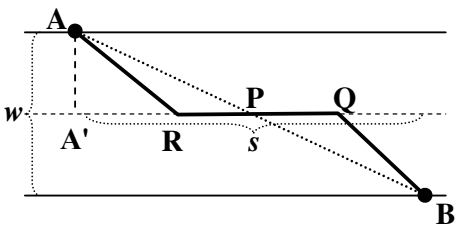
Програмата трябва да изведе на стандартния изход един ред с минималното време, за което можете да стигнете от A до B, закръглено с точност до секунда, в стандартен времеви формат – часове, двоеточие, минути, двоеточие, секунди – и без водещи нули.

ПРИМЕР

Вход	**Изход**
1.5 6 3.12 9.6	1:4:47

Решение

От съображения за симетрия и равнопоставеност, маршрутът трябва да има вид, подобен на начупената линия ARQB на чертежа – симетричен относно P. Тогава е достатъчно да пресметнем времето за достигане до точка P и да го удвоим.



Остава да определим положението на точка R върху средната права. Ясно е, че тя не може да е по-наляво от A', мястото ѝ е в затворената отсечка A'P. Също така е ясно, че с преместването на R по A'P от ляво на дясно, търсеното време или само ще нараства, или само ще намалява, или ще намалява донякъде и после ще започне да расте, т. е., ще има една най-малка стойност – или в A', или в P (когато отсечката RQ ще се изроди в точка), или някъде между тях. На тези разсъждения се основава конкретната реализация на алгоритъма „клатене" – търсене на интервал с ширина 0.1, съдържащ точката P, търсене в него на интервал с ширина 0.01, съдържащ P и т.н., до намиране на интервал с ширина 0.00001, съдържащ P, което е напълно достатъчно за точност до секундата в исканията формат.

```
#include <stdio.h>
#include <math.h>
typedef struct {double x,y;} Point;
double w,s,a,b;
Point A,P;

//Разстояние между точки
double dist(Point p1, Point p2)
{  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+
               (p1.y-p2.y)*(p1.y-p2.y));
}
//Време за изминаване на ARP
double getTime(Point R)
{  return dist(A,R)/a+dist(R,P)/b;  }
```

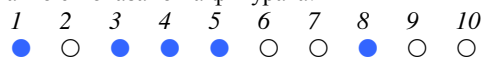
```
//Закръгляне
double round(double x)
{  return floor(x+0.5);  }

//Време в стандартен формат
char *format(double t,char *r)
{  double h,m,s;
   float p=modf(t,&h);
   p=round(p*3600);
   s=fmod(p,60);
   m=floor(p/60);
   sprintf(r,"%.0f:%.0f:%.0f",h,m,s);
   return r;
}
//Намиране на минимално време
double minTime(void)
{  double m,min,x,st;
   Point R;
   R.x=0;R.y=w/2;
   min=getTime(R);
   st=x=0.1; //"едра" стъпка
   do
   {  do
      {  R.x=x;
         m=getTime(R);
         if (m<=min) min=m;
         else break;
         x+=st;
      }while (x<=s/2);
      x-=2*st;
      st/=10;// намаляване на стъпката 10 пъти
      R.x=x;
      m=getTime(R);
      if(fabs(m-min)<0.00001) break;//край
      min=m;
   }while(1);
   return 2*min;
}
int main (void)
{  char buf[16];
   scanf("%lf %lf %lf %lf",&w,&s,&a,&b);
   A.x=0;A.y=w;
   P.x=s/2;P.y=w/2; //Координати на точките A и P.
   printf("%s\n",format(minTime(),buf));
   return 0;
}
```

Задача С3. ОГЪРЛИЦИ, автор Павлин Пеев

Фирма „Гери и Дани" има утвърден оригинален патент за мънистени огърлици в класическите два цвята – синьо и бяло. Освен интересната закопчалка на винт, към патента е включен и уникален метод за подреждане

на мънистата, който е и гаранция за автентичност. Нека при разкопчана огърлица с дължина n да номерираме мънистата, отляво надясно, с целите числа от 1 до n, както е показано на фигурата:



Патентованата подредба се подчинява на следните две правила:

1. Най-лявото мънисто (това с номер 1) е синьо.
2. Нека сините мъниста с нечетни номера са p на брой, а сините мъниста с четни номера – q на брой. Разглеждаме абсолютната стойност d на разликата на числата p и q, $d = |p - q|$. Тя е цяло неотрицателно число. Патентът гласи, че правилна е тази подредба, за която d се дели на 3 без остатък.

Показаната по-горе огърлица, например, не е правилно подредена: тя изпълнява първото условие (мънисто 1 е синьо), но при нея, както се вижда, $p = 3$, $q = 2$, т. е. $d = 1$ и d не се дели на 3. Напротив, огърлиците ●●, ●●○●● и ●○●○●●●○○, например, отговарят и на двете условия.

От фирмата искат да знаят колко са правилните огърлици с определена дължина (брой мъниста). Помогнете им, като напишете програма **NECKLACE**, която решава задачата.

Вход

Програмата трябва да въведе от единствения ред на стандартния вход дължината n ($1 \le n \le 60$) на огърлицата.

Изход

Програмата трябва да изведе броя на различните правилни огърлици с дължина n на стандартния изход.

ПРИМЕР

Вход	**Изход**
6	11

Обяснение: Правилните огърлици с дължина 6 са: ●○○○○●, ●○○○●○○, ●○○●●●, ●○●○●○, ●○●●●○●, ●●○○○○, ●●○○●●, ●●○●●●○, ●●●○○●, ●●●●○○ и ●●●●●●.

Решение

Очевидна е връзката с двоични числа. Ако кодираме синьото мънисто с 1, а бялото – с 0, то зададените правила описват n-цифрените двоични числа без водещи нули, които се делят на 3. Наистина, ако с a_i означим двоичните цифри на едно число A, то $A = \overline{a_1 a_2 a_3 \ldots a_n} = a_1.2^{n-1} + a_2.2^{n-2} + a_3.2^{n-3} + \ldots + a_n$. Като имаме предвид, че $2 \equiv -1 \pmod 3$, то четните степени в това представяне могат да се заместят с 1, а нечетните – с -1 по модул 3. Тогава е ясно, че един признак за делимост на 3 в двоична бройна система е делимостта на 3 на алтернативната сума от двоичните цифри $a_1 - a_2 + a_3 - a_4 + \ldots$. Това изказване всъщност е еквивалентно на формулировката в задачата. Ще припомним само, че, поради аналогични причини, подобно изглежда и известният признак за делимост на 11 в десетична бройна система. И естествено, защото $11_2 = 3$.