НАЦИОНАЛНА ОЛИМПИАДА ПО ИНФОРМАТИКА
Общински кръг
26 януари 2008 г.

УСЛОВИЯ И РЕШЕНИЯ НА ЗАДАЧИТЕ

Задача А1. ДРОБИ

Напишете програма **`fract`**, която намира сумата на всички несъкратими дроби $\frac{a}{b}$ (a и b са естествени числа, за които $k \le a < b \le n$).

Вход
От един ред на стандартния вход се въвеждат целите числа k и n ($1 \le k < n \le 100$, $n - k < 10$).

Изход
На един ред на стандартния изход програмата трябва да изведе несъкратима дроб, равна на търсената сума.

ПРИМЕР

Вход
```
1 3
```

Изход
```
3/2
```

Решение: Дробите, с които ще работим в задачата, може да разглеждаме като елементи на структура с два члена p и q, отразяващи съответно числителя и знаменателя на дробта. За да решим задачата, генерираме всяка дроб $\frac{a}{b}$, за която $k \le a < b \le n$, проверяваме дали е несъкратима и ако това е така, я добавяме към текущата променлива *s*, която използваме за намиране на търсената сума. За начални стойности на *s.p* и *s.q* задаваме съответно 0 и 1. След всяко сумиране на две дроби извършваме възможните съкращения в резултата.

```
#include<iostream>

using namespace std;

struct Fract
{ long long p,q; };

int nod(long long a, long long b)
{ long long r = a%b;
  while(r > 0)
```

```
  { a = b;
    b = r;
    r = a%b;
  }
  return b;
}

int main()
{   int k, n;
    cin >> k >> n;
    Fract s, x;
    s.p = 0;
    s.q = 1;
    for(int a=k; a<n; a++)
      for(int b=a+1; b<=n; b++)
        if (nod(a,b) == 1)
          {x.p = a;
           x.q = b;
           long long p = s.p*x.q + s.q*x.p;
           long long q = s.q*x.q;
           int d = nod(p,q);
           s.p = p/d;
           s.q = q/d;
          }
    cout << s.p << "/" << s.q << endl;
    return 0;
}
```

автор: Младен Манев

Задача А2. ПОЛИГОН

В окръжност с радиус единица е вписан правилен многоъгълник с n върха. Върховете му са номерирани с целите числа от 1 до n по посока на часовниковата стрелка. Започваме да се движим от връх 1 по посока на часовниковата стрелка, като изминаваме праволинейна отсечка към следващ връх, който избираме след p последователни върха. Така от връх 1 попадаме във връх с номер p+2. Продължаваме движението на стъпки по същия начин, докато се върнем във връх 1. Напишете програма **gone**, която въвежда n и p, и извежда общата дължина на изминатия път.

Вход
От стандартния вход се въвежда един ред с естествените числа n и p, разделени с интервал, $2 < n < 1000$, $0 < p < n - 1$.

Изход
Изведете на стандартния изход един ред с получената дължина на пътя като число с десетична точка и с точност до петата цифра в дробната част.

ПРИМЕР 1

Вход
```
4 1
```

Изход
```
4.00000
```

ПРИМЕР 2

Вход
```
4 2
```

Изход
```
5.65685
```

Решение: Очевидно е, че дължините на отсечките, изминати при всяка стъпка са равни помежду си. За да решим задачата, трябва да намерим броя на стъпките и дължината на една от отсечките. В програмата броят на стъпките се намира чрез моделиране на процеса – в цикъл последователно пресмятаме номера на поредния посетен връх, докато попаднем отново във връх 1. В променливата `c` се получава търсеният брой. Дължината на отсечката се пресмята с формулата за дължина на отсечка с краища връх 1 и връх p+2. Както е известно, когато променливата i приема стойностите 0, 1, 2, ..., $n-1$, точките с координати $(\cos(2\pi i/n), \sin(2\pi i/n))$ пробягват върховете на правилен n-ъгълник, вписан в окръжност в радуис 1. Така в програмата е използвано, че връх 1 има координати (1, 0), а връх p+2 е с координати $(\cos((p+1)(2\pi/n)), \sin((p+1)(2\pi/n)))$.

```
#include<cmath>
#include<cstdio>

int n,p;

int main()
{
  scanf("%d%d",&n,&p);
  int t=1;
  int c=0;
  do
  {
   t += (p+1);
   if(t>n) t -= n;
   c++;
  }
  while(t!=1);

  double len=sqrt(pow((1-cos((p+1)*(2.0*M_PI/n))),2.0)+
                  pow((sin((p+1)*(2.0*M_PI/n))),2));
  printf("%0.5lf\n",c*len);
}
```

автор: Емил Келеведжиев

Задача А3. ПРИНАДЛЕЖНОСТ

Множеството от естествени числа *M* се образува по следните правила:

1) $1 \in M$.
2) Ако $a \in M$, а с α означим двоичния му запис, то числото с двоичен запис $\overline{11\alpha}$ принадлежи на *M*.
3) Нека $a \in M$ и $b \in M$ и с α и β означим двоичните записи съответно на *a* и на *b*. Тогава числото с двоичен запис $\overline{\alpha 0 \beta}$ също е от *M*.

Например първите 10 числа от *M*, наредени по големина, са 1, 5, 7, 21, 23, 29, 31, 85, 87, 93.

Напишете програма **belong**, която за въведени три естествени числа *N*, *P* и *Q* определя дали принадлежат или не на *M*.

Вход

От стандартния вход се въвежда един ред с естествените числа *N*, *P* и *Q*, разделени с интервал. Никое от числата не е с повече от 18 десетични цифри.

Изход

Запишете на стандартния изход един ред с три символа, всеки от които 0 или 1: за всяко от входните числа (в същия ред) запишете 0, ако то не принадлежи на *M* или 1, ако принадлежи.

ПРИМЕР

Вход

```
1270 95 2047
```

Изход

```
011
```

Решение: Ограниченията допускат пряка проверка по правилата за всяко от входните числа – 18 десетични цифри се събират в 8-байтов целочислен тип. Такова решение рискува да е по-бавно за някои големи числа (**inM1** в реализацията).

От условието обаче следват следните свойства на двоичния запис на всеки от членовете на *M*:

- Двоичните цифри са нечетен брой. Наистина, 1 е с една цифра; правило 2 добавя нови две цифри и следователно не сменя четността, а правило 3 обединява две предишни дължини, като добавя още една цифра. Тъй като в началото цифрата е една (нечетен брой), няма правило, по което четността да се смени.
- Правилата не пораждат числа, в които има съседни нули. Нещо повече – нулите могат да са само на четно място, съгласно правило 2.
- Всички числа с нечетна дължина в двоичен запис и нули (ако се срещат такива) на четни места участват в *M*. Наистина – за дължина 1 това е тривиално вярно (по правило 1). Да допуснем, че е вярно за всички нечетни дължини, по-малки и равни на нечетното число *n*. Да разгледаме сега числата с дължина на двоичния запис *n*+2. Ако в този запис няма нула, той се поражда от числото 1 (правило 1) чрез (*n*+1)/2 пъти прилагане на правило 2. Ако има (поне) една нула (на нечетно място, разбира се), от лявата и от дясната ѝ страна има записи с нечетни дължини и нули (ако се срещат) на нечетни места. Съгласно индуктивното

предположение, там могат да се срещат всевъзможните записи със съответни дължини (не по-големи от *n*). По правило 3 пък всеки такъв запис е от *M*.

Използването на тези факти води до линеен по броя на двоичните цифри (т. е. логаритмичен по отношение на кандидата) алгоритъм за установяване на принадлежността към *M*. (функцията **inM2**).

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
using namespace std;
long long N=5, P, Q;
int rec(char *s,int l)
{if (l<=0) return 0;
 if (l==1) return *s=='1';
 if (s[0]=='1' && s[1]=='1' && rec(&s[2],l-2)) return 1;
 for (int i=1;i<l-1;i++)
  if (s[i]=='0' && rec(s,i) && rec(&s[i+1],l-i-1)) return 1;
 return 0;
}
char *strrev(char *s)
{char c;
 for (int i=0,j=strlen(s)-1;i<j;i++,j--)
 {c=s[i];s[i]=s[j];s[j]=c;}
 return s;
}
int inM1(long long a)
{char b[64],i;
 for (i=0;a;i++)
  {b[i]='0'+ (a&1); a>>=1;}
 b[i]=0;
 strrev(b);
 return rec(b,strlen(b));
}
int inM2(long long a)
{char d[64],c=0;
 do
 {d[c++]=a&1;
  a>>=1;
 }while (a);
 if (!(c&1)) return 0;
 for (c--;c>=0;c-=2) if (!d[c]) return 0;
 return 1;
}
int main()
{cin>>N>>P>>Q;
 //cout<<inM1(N)<<inM1(P)<<inM1(Q)<<endl;
 cout<<inM2(N)<<inM2(P)<<inM2(Q)<<endl;
 return 0;
}
```

автор: Павлин Пеев

Задача B1. МАТЕМАТИКА

Госпожа Петрова забелязала, че нейните ученици все още срещат затруднения при събиране и изваждане на цели числа. Затова в края на последните няколко часа тя записвала на дъската последователно по n цели числа, по-големи от -100 и по-малки от 100, между които за домашна работа учениците трябвало да поставят знаците за събиране и изваждане и да пресметнат получения израз. При това числата не можело да се разместват, изважданията трябвало да са точно k на брой, а събиранията да са $n-k-1$ на брой. Госпожа Петрова обаче е обещала да пише шестица само на тези, които освен че са изчислили вярно образувания израз, са получили най-голямо число като отговор. В края на първия срок Иванчо много се нуждае от тази шестица. Той може да събира и изважда цели числа, но иска да е сигурен, че ще получи максимален резултат. Помогнете на Иванчо, като напишете програма **`math`**, която намира тази най-голяма стойност.

Вход
На първия ред на стандартния вход се въвеждат стойностите на n и k ($0 < k < n < 30$). На втория ред на стандартния вход се въвеждат n-те числа.

Изход
На един ред на стандартния изход програмата трябва да изведе търсената най-голяма стойност.

ПРИМЕР

Вход
```
3 1
-10 5 -6
```

Изход
```
1
```

Решение: Записаните на дъската n числа може да разглеждаме като елементи на едномерен масив a. Максималната сума може да се постигне, когато знаците за изваждане се поставят пред най-малките елементи на масива. Тъй като пред първото число знак не може да се поставя, то остава да се намерят най-малките k на брой числа измежду всички числа без първото. За решаване на задачата може да извършим следното:

1. Сортираме елементите на масива без $a[0]$ в низходящ ред.
2. Сумираме елементите на масива с номера от 0 до $n-k-1$.
3. От получената сума изваждаме останалите k елемента на масива.

```
#include<iostream>
#include<algorithm>

using namespace std;

bool comp(int x, int y)
{ return x > y; }
```

```
int main()
{ int n, k;
  cin >> n >> k;
  int a[32];
  for(int i=0; i<n; i++)
    cin >> a[i];
  sort(a+1, a+n, comp);
  int s = a[0];
  for(int i=1; i<=n-k-1; i++)
    s = s + a[i];
  for(int i=n-k; i<n; i++)
    s = s - a[i];
  cout << s << endl;
  return 0;
}
```

автор: Младен Манев

Задача В2. СУМИ ОТ ЦИФРИ

Да разгледаме естественото число n, записано в двоичен запис. С $B(n)$ ще означим сумата от цифрите му. Например за $n = 19 = 10011_2$ имаме $B(19)=3$. Напишете програма **bsum**, която за въведено n определя сумата $S = B(1) + B(2) + \ldots + B(n)$.

Вход
От стандартния вход се въвежда един ред с естественото число n, не по-голямо от 1000000.

Изход
Запишете на стандартния изход един ред с намерената сума S.

ПРИМЕР

Вход
```
387
```

Изход
```
1612
```

Решение: Линейното сканиране (функция **sum**) работи достатъчно бързо за зададените ограничения, ако е написано добре. Сумата на цифрите в двоичния запис всъщност е броят на единиците. Можем да използваме известния алгоритъм с побитово AND за намирането на този брой (функция **count1** в реализацията).
За по-големи ограничения може да се използва следното подобрение (**sum1**):
Лесно се съобразява, че ако n е с едно по-малко от степен на двойката, т.е. $n=2^k-1$, двоичният му запис ще се състои само от единици (k на брой), а заедно с числата преди това записите все едно отразяват всички подмножества на едно k-елементно множество. Следователно, единиците ще се срещат по 2^{k-1} пъти на всяко от k места, значи търсената сума до такова n ще е $k.2^{k-1}$. Идеята може и да се доразвие, но в реализацията е използвана дотук: намираме най-голямото цяло число от вида 2^k-1 (т. е.,

само от единици в двоичния запис), ненадминаващо n, изчисляваме по горната формула сумата дотам и пускаме сканиране до n (функция **sum1**).

```
#include <iostream>
using namespace std;
int count1(long a)
{int s=0;
 while (a)
 {s++;
  a&=(a-1);
 }
 return s;
}
long sum(long n)
{long s=0;
 for (long i=1;i<=n;i++) s+=count1(i);
 return s;
}
long sum1(long n)
{int k=0;
 long s,t=0;
 while (t<=n) {t=(t<<1)|1; k++;}
 t>>=1;
 k--;
 s=k*((long)1<<(k-1));
 for (long i=t+1;i<=n;i++) s+=count1(i);
 return s;
}
int main(void)
{long n;
 cin>>n;
// cout<<sum(n)<<endl;
 cout<<sum1(n)<<endl;
 return 0;
}
```

автор: Павлин Пеев

Задача В3. ПОДРЕДБА ЗА ПАРТИ

То си е мъка да подреждаш гостите около кръгла маса! Ако се окажат в съседство три или повече жени – готов пазар! Ако пък мъж се окаже с две съседки, ще му е трудно да кавалерства и на двете. На Вашето парти ще пристигнат M_1 мъже и N_1 жени, които трябва да настаните около първата маса, както и M_2 мъже и N_2 жени, които трябва настаните около втората маса. Напишете програма **arrange**, която да Ви помогне да решите трудната задача. Хората, предвидени за всяка от двете маси, не бива да се смесват.

Вход
От стандартния вход се въвеждат два реда, на всеки от които има по две естествени числа (брой мъже и брой жени, в този ред), разделени с интервал. Числата не са по-големи от 100.

Изход
Съответно за всяка от масите запишете на стандартния изход по един ред: едно решение на задачата или съобщението `NO`, ако такова няма. Ако решение съществува, опишете го като редица от символи, като за „жена" запишете символа 0 (нула), а за „мъж" – символа 1, като започнете от когото и да е. Не забравяйте, че последният описан гост всъщност е съсед на първия – масите са кръгли!

ПРИМЕР

Вход
```
7 3
2 4
```

Изход
```
1011100111
NO
```

Решение: Започвайки от мъж, ще реализираме "лакома" линейна стратегия на подреждане, максимално изразходваща жени и минимално мъже. Така на второ и трето място поставяме жени. Тъй като не можем да разположим следваща жена, на четвърто място поставяме мъж, след него, уви, още не можем да разположим жена (защото ще се получи мъж с две съседки), налага се да "използваме" още един мъж. Този процес продължаваме, докато още има за разполагане хора и от двата пола: две жени (една, ако е останала последната), следвани от двама мъже (един, ако е последният). След приключването на някой от половете можем да имаме следните ситуации:

- не са останали хора изобщо. Ако последният разположен е бил мъж, наредбата е добра, иначе – не може да бъде осъществена по правилата;
- останала е поне една жена – наредбата не може да се осъществи по правилата;
- останали са мъже (поне един) – спокойно ги разполагаме последователно накрая и получаваме една правилна наредба.

Всъщност, оказва се, че подредбата може да бъде изпълнена, ако броят на мъжете е по-голям от този на жените, не може, ако е по-малък, а ако са равни, може, когато броят на мъжете е четно число. Описаната стратегия обаче не се нуждае от тези разсъждения. Има един случай, при който тя не дава верен отговор – ако имаме само един мъж и една жена, те могат да се разположат на масата. Този случай трябва да се съобрази.

```
#include <iostream>
using namespace std;
int M1,N1,M2,N2;
int arr(int M,int W,char *r)
{int i=0;
 r[i++]='1';M--;
 if (!M) {r[i++]='0';
          W--;
          if (!W) {r[i]=0; return 1;}
          return 0;
```

```
         }
 while (M>0 && W>0)
 {r[i++]='0';W--;
  if (W) {r[i++]='0';W--;}
  r[i++]='1';M--;
  if (M) {r[i++]='1';M--;}
 }
 if (W) return 0;
 while (M--) r[i++]='1';
 r[i]=0;
 return 1;
}
int main(void)
{char b[256];
 cin>>M1>>N1;
 cin>>M2>>N2;
 if (arr(M1,N1,b)) cout<<b<<endl; else cout<<"NO\n";
 if (arr(M2,N2,b)) cout<<b<<endl; else cout<<"NO\n";
 return 0;
}
```

автор: Павлин Пеев

Задача С1. ТРИЪГЪЛНИЦИ

Напишете програма **`triangles`**, която намира колко са различните триъгълници с периметър P см, за които дължините на страните им, измерени в сантиметри, са цели числа, ако P е цяло число.

Вход
От стандартния вход се въвежда един ред със стойността на P ($2 < P < 100000$).

Изход
На един ред на стандартния изход програмата трябва да изведе търсеният брой триъгълници.

ПРИМЕР

Вход
```
5
```

Изход
```
1
```

Решение: Нека дължините на страните на триъгълника са a, b и c. Без ограничение може да считаме, че $a \le b \le c$. Освен това, за да съществува триъгълник с дължини на страните a, b и c, трябва сборът на всеки две от тези числа да е по-голям от третото число. Тъй като $a \le b \le c$, то достатъчно е само да бъде изпълнено условието $a + b > c$. Остава само да генерираме всички тройки числа a, b и c, за които $a \le b \le c$, $a + b + c = P$ и $a + b > c$ и да ги изброим.

```
#include<iostream>

using namespace std;

int main()
{
    int P, a, b, c, s = 0;
    cin >> P;
    for(int a=1; 3*a <= P; a++)
      for(int b=a; a + 2*b <= P; b++)
        { c = P - a - b;
          if (a + b > c) s++;
        }
    cout << s << endl;
    return 0;
}
```

автор: Младен Манев

Задача С2. ЧАСОВНИК

Електронен часовник отмерва часове и минути от `00:00` до `23:59`. Напишете програма **`clock`**, която по две зададени показания на часовника намира колко време след първото показание на циферблата на часовника ще се появи второто показание.

Вход
На първия ред на стандартния вход се въвежда първото показание на часовника, а на втория ред – второто показание на часовника. Двете показания са различни и се състоят от по пет символа, като първите два показват часа, третият е „**:**”, а последните два показват минутите.

Изход
На един ред на стандартния изход програмата трябва да изведе търсеното време във формата на входните данни.

ПРИМЕР

Вход
```
23:58
01:02
```

Изход
```
01:04
```

Решение: За да решим задачата първо преобразуваме началното и крайното показание на часовника в минути. Ако крайното показание на часовника показва по-малък час от началното, към него прибавяме 1440 (1440 минути са 24 часа). Търсеното време в минути получаваме като от крайното показание на часовника извадим началното. Остава резултатът да се изведе във формата на входните данни.

```
#include<iostream>

using namespace std;

int main()
{ char a1, a2, a3, a4 , a5, b1, b2, b3, b4, b5;
  cin >> a1 >> a2 >> a3 >> a4 >> a5;
  cin >> b1 >> b2 >> b3 >> b4 >> b5;
  int a = (10 * (a1 - '0') + (a2 - '0'))*60 + 10*(a4 - '0') +
(a5 - '0');
  int b = (10 * (b1 - '0') + (b2 - '0'))*60 + 10*(b4 - '0') +
(b5 - '0');
  if (a >= b) b = b + 1440;
  int c = b - a;
  if (c/60 < 10) cout << 0;
  cout << c/60 << ':';
  if (c%60 < 10) cout << 0;
  cout << c%60 << endl;
  return 0;
}
```

автор: Младен Манев

Задача С3. МИНИМАЛНА РАЗЛИКА

Дадени са 4 различни десетични цифри, никоя от които не е нула. От тях са образувани две двуцифрени числа a и b, като всяка цифра е взета точно веднъж. Да приемем, че $a > b$. Напишете програма **mindif**, която определя стойността на минималната възможна разлика $a - b$.

Вход
От стандартния вход се въвежда един ред с 4 различни десетични цифри, разделени с интервал.

Изход
Запишете на стандартния изход един ред с получената минимална разлика.

ПРИМЕР

Вход
```
5 2 9 1
```

Изход
```
6
```

Обяснение: Най-малка разлика в примера се получава, когато умаляемото е 25, а умалителят – 19.

Решение: Задачата може да се реши с изчерпване на случаите (те са 24) като, например, се разгледат всички пермутации на дадените четири цифри, от първите две в получения ред се състави *a*, а от вторите две – *b*, образува се разликата им *a* – *b* и се запомни минималната положителна разлика по време на целия процес. Има и по-ефективен алгоритъм:

- Щом търсим минимална разлика, най-старшите цифри трябва да са възможно най-близки, значи ще изберем за старши цифри някои с най-малка разлика.
- След избора на първи цифри за умаляемото и умалителя, останалите две цифри имат единствено разумно разположение с цел минимизиране на разликата – по-голямата от тях да е втора цифра на умалителя, а по-малката – втора цифра на умаляемото.
- Избираме най-малката разлика при този процес. Може да има най-много три избора за двойка старши цифри, затова алгоритъмът е по-ефективен.

Разбира се, и за двата подхода подреждането на входните данни облекчава програмирането.

```
#include <iostream>
using namespace std;
int a[4];
int MaxNo(int start)
{int i,m=start;
 for(i=start+1;i<4;i++) if (a[i]>a[m]) m=i;
 return m;
}
void selSort(void)
{int i,j,c;
 for (i=0;i<3;i++){j=MaxNo(i);
                   c=a[i];
                   a[i]=a[j];
                   a[j]=c;
                  }
}
int better(void)
{int i,j,m=100,p,q,d=10;
 selSort();
 for (i=0;i<3;i++) if (a[i]-a[i+1]<d) d=a[i]-a[i+1];
 for (i=0;i<3;i++)
  if (a[i]-a[i+1]==d)
  {p=10*a[i];
   q=10*a[i+1];
   switch(i)
   {case 0:{p+=a[3];q+=a[2];break;}
    case 1:{p+=a[3];q+=a[0];break;}
    case 2:{p+=a[1];q+=a[0];}
   }
   if (p-q<m) m=p-q;
  }
 return m;
}
int main(void)
{int i;
```

```
 for (i=0;i<4;i++) cin>>a[i];
 cout<<better()<<endl;
 return 0;
}
```

автор: Павлин Пеев

Задача D1. ПРАВОЪГЪЛНИЦИ

Напишете програма **`rectangles`**, която намира колко са различните правоъгълници с лице S см^2, за които дължините на страните им, измерени в сантиметри, са цели числа, ако S е цяло число.

Вход
От стандартния вход се въвежда един ред със стойността на S ($1 < S < 100000$).

Изход
На един ред на стандартния изход програмата трябва да изведе търсения брой правоъгълници.

ПРИМЕР

Вход
```
4
```

Изход
```
2
```

Решение: Нека дължините на страните на правоъгълника са a и b. Без ограничение може да считаме, че $a \leq b$. S, a и b са цели числа. Следователно правоъгълник със страна a и лице S съществува само, ако S се дели на a (в противен случай b няма да бъде цяло число). За да изброим само правоъгълниците, за които $a \leq b$, на a даваме само такива стойности, за които $a^2 \leq S$.

```
#include<iostream>

using namespace std;

int main()
{ int S, a, sum = 0;
  cin >> S;
  for(int a=1; a*a <= S; a++)
    if (S % a == 0) sum++;
  cout << sum << endl;
  return 0;
}
```

автор: Младен Манев

Задача D2. ДЕЛИМОСТ НА 3

Нека са зададени 3 различни ненулеви десетични цифри. Напишете програма **`least3`**, която определя най-малкото цяло положително число, което се дели на 3 и в чийто десетичен запис се съдържат само цифри измежду дадените. Всяка от цифрите може да не участва в резултата, а може и да се среща веднъж или повече пъти.

Вход
От стандартния вход се въвежда един ред с трите различни десетични цифри, разделени с интервал.

Изход
Изведете на стандартния изход един ред с търсеното число.

ПРИМЕР 1

Вход
```
4 7 2
```

Изход
```
24
```

ПРИМЕР 2

Вход
```
7 1 4
```

Изход
```
111
```

Решение: Ако наредим цифрите по големина $a < b < c$, то кандидатите за решение (в нарастващ ред) са: a, b, c, $\overline{ab}$, $\overline{ac}$, $\overline{bc}$ и $\overline{aaa}$. Наистина, ако поне една цифра се дели на три, най-малката такава е едноцифреното решение. Иначе пробваме двуцифрените по големина (няма смисъл да се разглеждат другите варианти – ако $\overline{ab}$ не е кратно на 3, $\overline{ba}$ също не е – според признака за делимост; $\overline{aa}$ е кратно на 3 само ако и самото a е такова). И накрая, число с три еднакви цифри винаги се дели на 3 (пак по признака за делимост), а $\overline{aaa}$ е и най-малкото трицифрено число, съставимо от трите цифри.

```
#include <iostream>
using namespace std;
int main(void)
{int a,b,c,d;
 cin>>a>>b>>c;
 if (a>b){d=a;a=b;b=d;}
 if (b>c){d=b;b=c;c=d;}
 if (a>b){d=a;a=b;b=d;}
 if (a%3==0) cout<<a;
 else if (b%3==0) cout<<b;
 else if (c%3==0) cout<<c;
 else if ((a+b)%3==0) cout<<a<<b;
```

```
 else if ((a+c)%3==0) cout<<a<<c;
 else if ((b+c)%3==0) cout<<b<<c;
 else cout<<a<<a<<a;
 cout<<endl;
 return 0;
}
```

автор: Павлин Пеев

Задача D3. НАЙ-БЛИЗО

Напишете програма **prox**, която въвежда три цели положителни числа *a*, *b* и *c* (по-малки от 999) и намира най-близкото число до *c*, което може да се получи чрез едно от действията събиране или умножение, приложено към *a* и/или *b*. Ако съществува повече от едно такова най-близко число, да се изведе най-малкото.

Вход
На единствения ред на стандартния вход се въвеждат числата *a*, *b* и *c*, разделени с интервал.

Изход
На един ред на стандартния изход програмата трябва да изведе намереният най-близък резултат.

ПРИМЕР 1

Вход
```
3 5 11
```

Изход
```
10
```

ПРИМЕР 2

Вход
```
3 5 9
```

Изход
```
9
```

ПРИМЕР 3

Вход
```
3 5 7
```

Изход
```
6
```

Пояснение към пример 1: Всички възможни резултатите от действията, описани в условието на задачата, се получават от изразите 3+5, 3*5, 3+3, 5+5, 3*3, 5*5 и те са съответно равни на 8, 15, 6, 10, 9 и 25. Измежду тях най-близко до 11 е числото 10.

Решение: В елементите на масива *v* пресмятаме всичките 6 стойности, които могат да се получат от *a* и *b*, съгласно условието на задачата. След това с цикъл намираме най-малката дължината на разстоянието между някоя от тези стойности и *c*. Понеже е възможно тази най-малка дължина да се получи при две различни стойности от масива *v*, с още един цикъл откриваме по-малката от тях.

```
#include<cstdio>
#include<cstdlib>

int a,b,c;
int v[6];

int main()
{
  scanf("%d%d%d", &a, &b, &c);

  v[0]=a+b;
  v[1]=a*b;
  v[2]=a+a;
  v[3]=b+b;
  v[4]=a*a;
  v[5]=b*b;

  int d=abs(v[0]-c);
  for(int i=1;i<=5;i++)
   { int r=abs(v[i]-c);
     if(r<d) d=r;
   }

  int t=999999;
  for(int i=0;i<=5;i++)
  if(abs(v[i]-c)==d)
  if(t>v[i]) t=v[i];

  printf("%d\n",t);

}
```

автор: Емил Келеведжиев

Задача Е1. МРАВКА

Миналото лято мравката Здравка попаднала на цяло находище от *n* трохи. Първата се намирала на 1 см от нейния мравуняк, втората – на 2 см, третата – на 3 см, четвъртата – на 4 см, …, *n*-тата – на *n* см. Здравка излизала от мравуняка, вземала по една троха, носела я в мравуняка и след това продължавала по същия начин до събирането на всички трохи. Напишете програма **ant**, която намира дължината на пътя, който Здравка е изминала, за да прибере всички трохи в мравуняка.

Вход
От стандартния вход на един ред се въвежда стойността на *n* ($1 < n < 250$).

Изход
На един ред на стандартния изход програмата трябва да изведе дължината на изминатия от Здравка път в сантиметри.

ПРИМЕР

Вход
```
5
```

Изход
```
30
```

Решение: Изминатият от мравката път е равен на удвоения сбор на естествените числа от 1 до *n*.

```
#include<iostream>

using namespace std;

int main()
{
    int n, s = 0;
    cin >> n;
    for(int i=1; i<=n; i++)
      s = s + i;
    s = 2*s;
    cout << s << endl;
    return 0;
}
```

автор: Младен Манев

Задача Е2. ЧИСЛА

Иванчо знае само първата цифра a и последната цифра b на трицифреното число $\overline{a*b}$. Напишете програма **numbers**, която намира по колко различни начина той може да замени звездичката с цифра, така че полученото число да се дели на цялото число k.

Вход
На единствения ред на стандартния вход се въвеждат стойностите на a, b и k ($0 < a < 10$, $0 \le b < 10$, $1 < k < 100$).

Изход
На един ред на стандартния изход програмата трябва да изведе търсения брой начини.

ПРИМЕР

Вход
```
1 2 2
```

Изход
```
10
```

Решение: Да означим липсващата цифра с *i*. Тогава числото $\overline{a*b} = 100a + 10i + b$. Цифрата i може да бъде 0, 1, 2, 3, 4, 5, 6, 7, 8 или 9. За всяка от тези десет възможности проверяваме дали числото $\overline{a*b} = 100a + 10i + b$ се дели на *k*.

```
#include<iostream>

using namespace std;

int main()
{   int a, b, k, n, s=0;
    cin >> a >> b >> k;
    for(int i=0; i<=9; i++)
      { n = 100*a + 10*i + b;
        if (n%k ==0) s++;
      }
    cout << s << endl;
    return 0;
}
```

автор: Младен Манев

Задача Е3. ЗАЙЧЕ

Напишете програма **rabbit**, която отпечатва три вида фигурки на екрана в посочена последователност. Всяка фигурка се състои от по пет символа. Първо се въвежда „код" на фигурката.

Числото 1 е „код" на фигурката: `(\_/)`

Числото 2 е „код" на фигурката: `(o.o)`

Числото 3 е „код" на фигурката: `(_._)`
Понякога вашата програма ще изведе зайче, друг път нещо друго.

Вход
На стандартния вход се въвеждат три числа, всяко от които е 1, 2 или 3, като някои от числата могат да се повтарят.

Изход
На стандартния изход да се изведе получената последователност от три фигурки, всяка на нов ред.

ПРИМЕР 1

Вход
```
1 2 3
```

Изход
```
(\_/)
(o.o)
(_._)
```

ПРИМЕР 2

Вход
```
2 2 1
```

Изход
```
(o.o)
(o.o)
(\_/)
```

Решение:

```
#include <iostream>
using namespace std;
int main()
{ int a,b,c;
  cin>>a>>b>>c;
  if (a == 1) cout<<"(\\_/)\n";
  else if (a == 2) cout<<"(o.o)\n";
  else if (a == 3) cout<<"(_._)\n";
  if (b == 1) cout<<"(\\_/)\n";
  else if (b == 2) cout<<"(o.o)\n";
  else if (b == 3) cout<<"(_._)\n";
  if (c == 1) cout<<"(\\_/)\n";
  else if (c == 2) cout<<"(o.o)\n";
  else if (c == 3) cout<<"(_._)\n";
}
```

автор: Зорница Дженкова