Problema 1

Descripción:

La Olimpiada Internacional de Informática (IOI) explica en su temario que es necesario tener conocimientos básicos de matemáticas para ser capaz de competir en la competencia. En este primer problema empezaremos por evaluar tus habilidades matemáticas. Dados dos enteros positivos y un signo que indica la operación a realizar, tu tarea es imprimir el resultado de aplicar dicha operación a los números dados.

Entrada:

Una línea con dos enteros positivos a y b separados por un espacio. En la siguiente línea, un caracter "+", "−", "∗" o "/" indicando la operación a realizar: suma, resta, multiplicación o división, respectivamente.

Salida:

Un entero que represente el resultado de aplicar la operación a ambos enteros.

Ejemplo:

entrada	salida
1 1 +	2
2 1 -	1

Notas:

- Ambos enteros positivos dados son menores o iguales a 1000.
- En el caso de las operaciones de resta y división, debes restar (o dividir) el segundo número al primero (como en el ejemplo).
- Puedes asumir que el resultado de la operación siempre será un entero.

Solución:
Es necesario saber leer entrada, el uso de condicionales e imprimir salida correctamente.

```
#include<iostream>
using namespace std;
int main ()
{
  int a;
  int b;
  char c;
  cin >> a;
  cin >> b;
  cin >> c;

  if (c == '+')
  {
    cout << a + b;
  }
  else if (c == '-')
```

```
  {
    cout << a - b;
  }
  else if (c == '*')
  {
    cout << a * b;
  }
  else if (c == '/')
  {
    cout << a / b;
  }
}
```

Problema 2

Descripción

Una cadena de paréntesis es *balanceada* si cada paréntesis de apertura tiene un paréntesis de cierre correspondiente. Dada una cadena de paréntesis, decidir si es balanceada o no.

Entrada

Cadena de paréntesis.

Salida

La salida debe constar de dos líneas. La primera línea debe ser la cantidad de paréntesis de apertura en la cadena. La segunda línea debe indicar "Balanceada" en caso de que la cadena sea balanceada, "No balanceada" en caso contrario.

Ejemplo

entrada	salida
(()())()	4 Balanceada
)(()))()	3 No balanceada

Solución:
Podemos contar los paréntesis de apertura y de cierre. Si en algún momento hay más paréntesis de cierre, significa que la cadena ya no es balanceada ya que no puede existir uno de apertura para ese paréntesis de cierre. Al final de procesar la cadena, si tenemos el mismo número de paréntesis de apertura y de cierre, podemos concluir que la cadena es balanceada.

```
#include<iostream>
using namespace std;

#include<iostream>
using namespace std;
int main ()
{
  string str;
  cin >> str;
  int apertura = 0;
```

```
  int cierre = 0;
  bool noBalanceada = false;

  for (int i = 0; i < str.length(); i++)
  {
    if (str.at(i) == '(')
    {
      apertura++;
    }
    else
    {
      cierre++;
    }

    if (cierre > apertura)
    {
      noBalanceada = true;
    }
  }

  cout << apertura << endl;

  if (noBalanceada == true)
  {
    cout << "No balanceada";
  }
  else if (apertura == cierre)
  {
    cout << "Balanceada";
  }
  else
  {
    cout << "No balanceada";
  }
}
```

Problema 3

Descripción

Tus amigos Ana y Beto deciden organizar un torneo de n rondas basado en el juego *Piedra, Papel o Tijera*. Ellos te convencieron de que seas el juez del torneo por ser imparcial. Ya que además de imparcial eres bastante perezoso, decides escribir un programa que haga tu trabajo.

Entrada

- El número n de rondas.
- Una cadena con los caracteres R, P y T que representan piedra, papel o tijera respectivamente. El primer caracter de cada ronda corresponde a Ana.

Salida

El ganador de cada ronda y el ganador del torneo. Se debe usar el formato mostrado en el ejemplo.

Ejemplo

entrada	salida
2 PTRP	Beto gana Beto gana Beto gana el torneo
4 RTPPPTPR	Ana gana Empate Beto gana Ana gana Ana gana el torneo

Notas

- Si no estas familiarizado con el juego: Piedra gana a Tijera, Papel gana a Piedra y Tijera gana a Papel. En el problema R representa Piedra, P representa Papel y T representa Tijera.
- Las rondas son representadas por cada par de caracteres consecutivos: El primer par de caracteres representa la primera ronda, el segundo par representa la segunda ronda, y así sucesivamente.
- Puedes asumir que siempre habrá un ganador al final del torneo.

Solución:
La solución consiste en procesar cada ronda, siguiendo las reglas del juego ya descritas para decidir quién gana cada ronda. Debemos llevar un contador de rondas ganadas para cada jugador. Al final, el jugador con contador más grande gana el torneo.

```
#include<stdio.h>

int ganador(char a, char b)
{
    if(a == b)
        return 0;
    if((a == 'P' && b == 'R') || (a == 'R' && b == 'T') || (a == 'T' && b == 'P'))
        return 1;
    else
        return 2;
}

int main()
{
    int n;
    scanf("%d", &n);
    getchar();

    int a_cont = 0;
    int b_cont = 0;

    while(n--)
    {
        char a;
```

```
        char b;
        scanf("%c %c", &a, &b);

        if(ganador(a, b) == 1){
            printf("Ana gana\n");
            a_cont++;
        }
        else if(ganador(a, b) == 2){
            printf("Beto gana\n");
            b_cont++;
        }
        else
            printf("Empate\n");
    }

    if(a_cont > b_cont)
        printf("Ana gana el torneo\n");
    else
        printf("Beto gana el torneo\n");
}
```

Problema 4

Descripción

El juego de los unos es un juego cooperativo de 3 jugadores y un juez que consiste en lo siguiente:

- El primer jugador escoge un entero positivo compuesto por N dígitos 1 exclusivamente, por ejemplo: 1111.
- El segundo jugador escoge un dígito d entre 0 y 8 (ambos inclusive).
- El tercer jugador escoge un entero positivo K, sin restricciones.

Lo interesante del juego es que no existe comunicación entre los jugadores por lo que sus elecciones de números no pueden depender de los demás.

El juez da el veredicto sobre si los jugadores ganaron el juego o no, decidiendo de la siguiente forma:

- Los jugadores ganan el juego si es posible sumarle el dígito d a alguno de los N dígitos del primer número y así obtener un número divisible para K.

Tu tarea es implementar un programa que pueda actuar como juez, decidiendo si los jugadores ganaron o perdieron, dependiendo de qué números escogieron inicialmente.

Entrada

Un entero positivo N, un dígito d (entre 0 y 8 inclusive) y un entero positivo K.

Salida

Si los jugadores ganan el juego, cada línea de la salida debe constar de cada una de las posiciones de dígitos a los que se les puede sumar el dígito d para conseguir un número divisible para K. La posición 1 corresponde a la posición del dígito de las unidades, la posición 2 al dígito de las decenas, etc. Las líneas deben estar ordenadas de forma ascendente. En caso de que los jugadores no puedan ganar el juego, imprimir la cadena "Derrota".

Ejemplo

entrada	salida
2 1 3	1 2
2 3 5	Derrota

Para el primer ejemplo, se pueden obtener los números 12 y 21, ambos múltiplos de 3, sumándole el dígito 1 al dígito de las unidades (posición 1) y al de las decenas (posición 2) respectivamente.

En el segundo ejemplo, ningunos de los posibles números 14 o 41 son múltiplos de 5.

Límites

$N \leq 9$

$0 \leq d \leq 8$

$K < 2^{30}$

Solución:
Se puede crear el número de la forma $11\ldots1$ con multiplicación iterativa con potencias sucesivas de 10. Luego se puede iterar para cada una de las posiciones de dígitos, crear el número de la forma $11\ldots d\ldots1$ sumándole $d \times 10^i$ y evaluar el residuo en la división para K.

```
#include<stdio.h>

int crearN(int n){
  int N = 0;

  for(int c = 0, i = 1; c < n; c++, i*=10)
    N += i;

  return N;
}

int main(){
  int n, d, k, N;
  int esPosible = 0;

  scanf("%d", &n);
  scanf("%d", &d);
  scanf("%d", &k);

  N = crearN(n);

  for(int c = 1, i = 1; c <= n; c++, i *= 10){
    if((N + d * i) % k == 0){
      esPosible = 1;
      printf("%d\n", c);
    }
  }

  if(!esPosible)
    printf("Derrota");
```

}

Problema 5

Descripción

¿Recuerdas cuando en la escuela calculabas áreas y perímetros de figuras geométricas? Calcular estos valores sabiendo los lados de la figura seguramente te resultaba bastante trivial. ¿Qué tan complejo sería hacer lo contrario? En este problema nos enfocaremos en el proceso inverso: Es decir, dados el perímetro y área de un triángulo, tu tarea será encontrar todos los triángulos con lados enteros positivos que tengan dicho perímetro y área.

Entrada

2 números (pueden contener decimales) separados por una línea. El primer número representa el perímetro y el segundo el área.

Salida

En caso de que existan triángulos que cumplan las condiciones del perímetro y área, cada línea debe constar de 3 enteros positivos (los lados del triángulo) separados por un espacio y ordenados ascendentemente. En caso de existir múltiples triángulos, las líneas de la salida deben estar ordenadas lexicográficamente entre sí.

En caso de no existir ningún triángulo que cumpla, imprimir la cadena "no hay".

Ejemplo

entrada	salida
1 1	no hay
70 210	17 25 28 20 21 29

Notas

- Recuerda que un triángulo es válido solamente si sus lados cumplen con las desigualdades triangulares: Para valores de sus lados a, b, c se debe cumplir que $c < a + b$, $a < b + c$, $b < a + c$.
- Los valores del perímetro y área dados serán igual o menores a 10000.
- 2 números de tipo float se consideran iguales si difieren en menos de 0.00001.

Solución:
Como un primer intento de solución a fuerza bruta, podemos hacer una búsqueda con un ciclo triple para los lados, respetando las desigualdades triangulares. Luego calculamos el perímetro y el área y verificamos si son iguales al perímetro y área dada por la entrada del problema. Para calcular el área en base a los 3 lados de un triángulo, podemos utilizar la fórmula de Herón. Para un triángulo con lados l_1, l_2, l_3 su área A está dada por la fórmula:

$$A = \sqrt{s(s - l_1)(s - l_2)(s - l_3)}$$

donde $s = \frac{l_1+l_2+l_3}{2}$ es el semiperímetro de la figura.

Esta solución no es del todo eficiente, ya que tiene un ciclo de más. Ya que tenemos el perímetro dado, bastaría hacer una búsqueda para dos de los lados del triángulo con un doble ciclo. El tercer lado lo podemos deducir restándole ambos lados considerados al perímetro. Incluso se puede acotar la búsqueda usando la desigualdad

triangular para quitar casos imposibles, pero esta última optimización no afecta a la complejidad de tiempo de la solución.

Aquí mostramos la solución simple con ciclo triple, pues es suficiente para resolver los casos del problema. Supongamos sin pérdida de generalidad que $l_1 \geq l_2 \geq l_3$. La desigualdad triangular implica que $2l_1 < l_2 + l_3 + l_1$ o que $l_1 < \frac{p}{2}$ donde p es el perímetro del triángulo. También implica que $l_2 + l_3 > l_1$, de donde $l_3 \geq l_1 - l_2 + 1$. Los límites de l_1, l_2 y l_3 están entonces dados por:

$$1 \leq l_1 < \frac{p}{2}$$
$$1 \leq l_2 \leq l_1$$
$$l1 - l2 + 1 \leq l_3 \leq l_2.$$

Con estos límites, podemos armar los tres ciclos correctamente:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double p;
    double a;
    cin >> p;
    cin >> a;
    bool hay = 0;
    double sp = p / 2;

    for (int l1 = 1; l1 <= p/2; l1++)
    {
        for (int l2 = 1; l2 <= l1; l2++)
        {
            for (int l3 = l1-l2+1; l3 <= l2; l3++)
            {
                if (l1 + l2 + l3 == p
                    && abs(sqrt(sp * (sp-l1) * (sp-l2) * (sp-l3)) - a) < 0.00001)
                {
                    hay = true;
                    cout << l3 << " " << l2 << " " << l1 << endl;
                }
            }
        }
    }

    if (!hay)
    {
        cout << "no hay";
    }
}
```