# Soluciones OIFem II Nivel 1

# Entrenamiento 2

## ¿Qué lado de la calle?

### Teoría

- if, else, else if

### Solución

Para leer todos los casos escribimos un `while (cin >> n)` para que vaya leyendo cada caso, y cuando `n` sea igual a 0 (que es la marca de final de secuencia), entonces debemos parar el código mediante la instrucción `break`.

Para cada caso, si leemos un número par, entonces estará al lado derecho, y si es impar, entonces estará al lado izquierdo.

Link al problema: ¿Qué lado de la calle?

### Código

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

int main()
{
  int n;
  while (cin >> n) {
    if (n == 0) {
       break;
    }
    if (n%2 == 0)
       cout << "DERECHA\n";
    else
       cout << "IZQUIERDA\n";
  }
}
```

## Fin de mes

## Teoría

- if, else, else if

## Solución

Primero, para leer la entrada de los casos, podemos hacer un `for` o un `while`. Entonces, llegaremos a fin de mes solamente cuando el saldo de la cuenta bancaria más el cambio (ingresos menos gastos) sea positivo o igual a cero. Por lo tanto, usamos la instrucción `if-else`.

Link al problema: Fin de mes

## Código

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

int main()
{
  int t;
  cin >> t;
  for (int i = 0; i < t; ++i) {
      int s, c;
      cin >> s >> c;
      if (s+c < 0)
        cout << "NO\n";
      else
        cout << "SI\n";
  }
}
```

# Triángulos

## Teoría

- if, else, else if

## Solución

Observamos que es imposible formar un trángulo con tres lados dados si no se cumple la Desigualdad Triangular, es decir, que para cada par de lados de triángulo, la suma es mayor que el lado restante (ver Desigualdad triangular para más información).

Entonces, para ver si el triángulo es acutángulo, rectángulo o obtusángulo, comprobamos si el Teorema de Pitágoras se cumple (entonces es rectángulo), si la cantidad de la suma de los catetos al cuadrado es menor que la hipotenusa al cuadrado (obtusángulo), o si la suma de los catetos al cuadrado es mayor que la hipotenusa al cuadrado (acutangulo). Ver Teorema de Pitágoras: Reconocimiento de triángulos para más información.

Link al problema: Triángulos

## Código

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

int main()
{
  int t;
  cin >> t;
  while(t--) {
      long long a, b, c;
      cin >> a >> b >> c;
      if (a < b+c and b < c+a and c < a+b) {
      // Comprobamos que las longitudes formen un triángulo

      // Hacemos un cambio de variables para guardar el valor máximo de los tres
      ↪  lados en a
        if (b > a) {
          b = b - a;
          a = a + b;
          b = a - b;
        }

        if (c > a) {
          c = c - a;
          a = a + c;
          c = a - c;
        }

        if (a*a > b*b + c*c)
          cout << "OBTUSANGULO\n";
        else if (a*a == b*b + c*c)
          cout << "RECTANGULO\n";
        else
          cout << "ACUTANGULO\n";

      }
      else {
        cout << "IMPOSIBLE\n";
      }
  }
}
```

# Yo soy tu...

## Teoría

- if, else, else if

## Solución

Como trabajamos con `string` debemos añadir el paquete `include <string>` porque sino, no compilará.

Entonces, como el lenguaje C++ reconoce los carácteres mayúscula y minúscula y los diferencia, usamos la instrucción `if` para ver si la combinación de palabras coinciden con “Luke” y “padre”.

Link al problema: Yo soy tu...

## Código

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

int main()
{
  int t;
  cin >> t;
  while(t--) {
      string personaje, parentesco;
      cin >> personaje >> parentesco;
      if (personaje != "Luke" or parentesco != "padre") {
        cout << personaje << ", yo soy tu " << parentesco << '\n';
      } else {
        cout << "TOP SECRET\n";
      }

  }
}
```

# ¡Hola mundo!

## Teoría

- while, for

## Solución

Leemos el número de veces que debemos imprimir ”Hola mundo” en pantalla y con un bucle ”for” (puesto que sabemos la cantidad) las imprimimos.

Link al problema: ¡Hola mundo!

## Código

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

int main () {
    int t;
    cin >> t;
    for (int i = 0; i < t; ++i) {
        cout << "Hola mundo.\n";
    }
```

```
}
```

# Numbers in an interval

## Teoría

- if, else, else if
- while, for

## Solución

Como sabemos la cantidad de números que hay en el intervalo, usamos el bucle "for".

Link al problema: Numbers in an interval

## Código

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

int main () {
        int a,b;
        cin >> a >> b;
        for (int i = a; i <= b; ++i) {
                cout << i;
                if (i != b) {
                        cout << ',';
                }
        }
        cout << '\n';
}
```

# Radares de tramo

## Teoría

- if, else, else if
- while, for

## Solución

Para cada entrada, separamos los cuatro posibles casos: que aparezca "ERROR", "OK", "MULTA" o "PUNTOS".

Link al problema: Radares de tramo

## Código

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

int main () {
        float distancia, limite, segundos;
        while (cin >> distancia >> limite >> segundos) {
                if (distancia == 0 and limite == 0 and segundos == 0) {
                        break;
                } else if (distancia <= 0 or limite <= 0 or segundos <= 0) {
                 ↪  //dejamos fuera los casos en que los valores son 0 o negativos
                        cout << "ERROR" << '\n';
                } else {
                        float velocidad = (distancia/segundos)*3.6; //la velocidad
                         ↪  = distancia/segundos está en m/s, y multiplicamos por
                         ↪  3.6 para pasarla a km/h, tal y como está el límite de
                         ↪  velocidad
                        if (velocidad <= limite) {
                                cout << "OK" << '\n';
                        } else if (velocidad < 1.2*limite) {
                                cout << "MULTA" << '\n';
                        } else if (velocidad >= 1.2*limite) {
                                cout << "PUNTOS" << '\n';
                        }
                }
        }
}
```

# Chess 2

Considere un tablero de ajedrez con $r$ filas y $c$ columnas, donde cada casilla contiene entre 0 y 9 monedas. Escribe un programa tal que, dado un tablero de ajedrez, calcule el número total de monedas que tiene en las **casillas blancas**.

Considere que la casilla de arriba a la derecha es siempre blanca.

**ENTRADA:** La entrada comienza con el número de filas $r$ y el número de columnas $c$. Se siguen $r$ líneas, cada una con $c$ caracteres entre 0 y 9.

**SALIDA:** Imprime el número total de monedas en las casillas blancas del tablero.

Link al problema: Chess 2

# Solución

Usamos el mismo razonamiento que en el problema Chess 1 (visto en clase) para leer las monedas de cada casilla. Ahora, solo debemos contar las monedas en las casillas blancas. Observamos que la paridad de la suma de los iteradores es 0 cuando señalan una casilla balnca y 1 cuando es negra. Por lo tanto, añadimos la condición de que módulo 2 deben sumar 0.

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

int main()
```

```
{
    // Contar el número de monedas en las casillas blancas del tablero.
    // B N B N ...
    // N B N B ...
    int r, c;
    int total = 0;
    cin >> r >> c;
    for (int i = 0; i < r; ++i) {
        for (int j = 0; j < c; ++j) {
            char k;
            cin >> k;
            if ((i - j) % 2 == 0)
                total += k - '0';
        }
    }
    cout << total << endl;
}
```

# Chess 3

Considere un tablero de ajedrez cuadrado con $n$ filas y $n$ columnas, donde cada casilla contiene entre 0 y 9 monedas. Escribe un programa tal que, dado un tablero de ajedrez, calcule el número total de monedas en sus dos diagonales.

**ENTRADA:** La entrada comienza con el tamaño $n$ del tablero. Se siguen $n$ líneas, cada una con $n$ caracteres entre 0 y 9.

**SALIDA:** Imprime el número total de monedas en las dos diagonales del tablero.

Link al problema: Chess 3

# Solución

Para leer las monedas del tablero usamos el mismo razonamiento que en el problema Chess 1 (visto en clase). Para contar solamente las monedas de las dos diagonales del tablero, observamos que el valor de las variables cuando señalan una diagonal se parecen: ya sea $i = j$ o $i = n - j - 1$. Por lo tanto, añadimos esta condición.

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

int main()
{
    // Contar el número de monedas en la diagonales de un tablero.
    int n;
    int total = 0;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            char k;
            cin >> k;
            if (i == j or i == n - j - 1)
                total += k - '0';
        }
```

```
    }
    cout << total << endl;
}
```

# Constante de Kaprekar

## Teoría

- if, else, else if
- while, for

## Solución

Para cada iteración de la "rutina de Kaprekar" descomponemos el número en sus cuatro cifras, y vamos comparando y ordenando las cifras construyendo el mayor número. El menor número se construye girando las cifras del número mayor.

Link al problema: Constante de Kaprekar

## Código

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

int main () {
 int t, n, total;
 cin >> t;
 for (int i = 0; i < t; ++i) {
  total = 0;
  cin >> n;
  while (n != 6174) {
   if (total == 8) {
    break;
   }
   int a, b, c, d, n1, n2;
   a = n%10; //unidad
   b = (n/10)%10; //decena
   c = (n/100)%10; //centena
   d = n/1000; //unidad de mil
   if (max(a,b) > max(c,d)) { //n1 = max(a,b) ...
    if (max(c,d) > min(a,b)) { //n1 = max(a,b) max(c,d) ...
     if (min(a,b) > min(c,d)) {
          n1 = 1000*max(a,b) + 100*max(c,d) + 10*min(a,b) + min(c,d);
          n2 = 1000*min(c,d) + 100*min(a,b) + 10*max(c,d) + max(a,b); //n2 es el
           ↪  número más pequeño formado por las cifras de n, y por lo tanto es
           ↪  exactamente el opuesto de n1
         }
     else {
      n1 = 1000*max(a,b) + 100*max(c,d) + 10*min(c,d) + min(a,b);
```

```
        n2 = 1000*min(a,b) + 100*min(c,d) + 10*max(c,d) + max(a,b);
           }
          }
          else { //n1 = max(a,b) min(a,b) ...
           n1 = 1000*max(a,b) + 100*min(a,b) + 10*max(c,d) + min(c,d);
           n2 = 1000*min(c,d) + 100*max(c,d) + 10*min(a,b) + max(a,b);
          }
     }
     else { //n1 = max(c,d) ...
      if (max(a,b) > min(c,d)) { //n1 = max(c,d) max(a,b) ...
           if (min(a,b) > min(c,d)) {
            n1 = 1000*max(c,d) + 100*max(a,b) + 10*min(a,b) + min(c,d);
            n2 = 1000*min(c,d) + 100*min(a,b) + 10*max(a,b) + max(c,d);
           }
           else {
        n1 = 1000*max(c,d) + 100*max(a,b) + 10*min(c,d) + min(a,b);
            n2 = 1000*min(a,b) + 100*min(c,d) + 10*max(a,b) + max(c,d);
           }
          }
          else { //n1 = max(c,d) min(c,d) ...
           n1 = 1000*max(c,d) + 100*min(c,d) + 10*max(a,b) + min(a,b);
           n2 = 1000*min(a,b) + 100*max(a,b) + 10*min(c,d) + max(c,d);
          }
     }
    n = n1 - n2;
    total++;
   }
   cout << total << '\n';
   }
 }
```

# El tablero en forma de donut

## Teoría

- if, else, else if
- while, for

## Solución

Calculamos la nueva posición horizontal y vertical después de desplazarse. Entonces, sumamos o restamos la cantidad del ancho y largo del tablero, respectivamente, para acabar en una posición dentro del tablero inicial.

Link al problema: El tablero en forma de donut

## Código

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

```

```cpp
int main() {
        int w, h, x, y, r, s;
        cin >> w >> h >> x >> y >> r >> s;
        r +=x;
        s += y;
        while (r < 0 or r >= w) {
                if (r < 0) {
                        r += w;
                } else {
                        r -= w;
                }
        }
        while (s < 0 or s >= h) {
                if (s < 0) {
                        s += h;
                } else {
                        s -= h;
                }
        }
        cout << r << ' ' << s << '\n';
}
```

# Otros problemas para practicar

Aquí proponemos una lista de problemas de jueces online para ir practicando este tema de `if`, `else if` y `else`:

- Link al problema: Gregorio XIII
- Link al problema: El día de Navidad
- Link al problema: Lletra del NIF