RELACIÓN DE PROBLEMAS III. Funciones y Clases

Problemas Básicos

Problemas sobre funciones

1. Encuentre los errores de las siguientes funciones:

```
int ValorAbsoluto (int entero) {
   if (entero < 0)
      entero = -entero;
   else
      return entero;
}

void Cuadrado (int entero) {
   return entero*entero;
}
```

```
void Imprime(double valor) {
   double valor;

   cout << valor;
}

bool EsPositivo(int valor) {
   if (valor > 0)
      return true;
}
```

   Finalidad: Familiarizarnos con la definición de funciones, el paso de parámetros y el ámbito de las variables. Dificultad Baja.

2. Cread una función que calcule el máximo entre tres `double`. La cabecera de la función será la siguiente:

   ```
   double Max(double un_valor, double otro_valor, double el_tercero)
   ```

   Construid un programa principal que llame a dicha función con unos valores leídos desde teclado. Supongamos que dichos valores los leemos con `cin` dentro de la propia función, en vez de hacerlo en el `main`. El suspenso está garantizado ¿Por qué?

   Finalidad: Familiarizarnos con la definición de funciones, el paso de parámetros y el ámbito de las variables. Dificultad Baja.

3. Reescribid la solución del ejercicio 20 (factorial y potencia) de la Relación de Problemas II, modularizándola con funciones.

   Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros. Dificultad Baja.

4. Cread las siguientes funciones relacionadas con la progresión geométrica que se vio en el ejercicio 23 de la relación de problemas II. Analizad cuáles deben ser los parámetros a estas funciones.

a) Una función `SumaHasta` que calcule la suma de los primeros k valores de una progresión geométrica.

Cread un programa principal que llame a dicha función.

b) Cambiemos ahora la implementación de la anterior función `SumaHasta`. Para ello, en vez de usar un bucle aplicamos la siguiente fórmula que nos da la sumatoria aplicando únicamente cinco operaciones:

$$\sum_{i=1}^{i=k} a_i = a_1 \frac{r^k - 1}{r - 1}$$

Es muy importante remarcar que el programa `main` no cambia nada. Hemos cambiado la implementación de la función y lo hemos podido hacer sin cambiar el `main`, ya que éste no tenía acceso al código que hay dentro de la función. Esto es *ocultación de información* tal y como se describió en las clases de teoría.

Nota. Calculad la potencia (r^k) con la función `Potencia` definida en el ejercicio 3 de esta Relación de Problemas.

Hay que destacar que el cómputo de la potencia es una operación costosa, por lo que hasta podría ser más lenta la versión nueva que la antigua usando un bucle `for`. Probad distintos valores para ver si hay diferencias significativas. En cualquier caso, lo importante es que mientras no cambiemos la cabecera de la función `SumaHasta`, podemos cambiar su implementación sin tener que cambiar ni una línea de código del `main`.

c) Una función `SumaHastaInfinito` para calcular la suma hasta infinito, según la fórmula:

$$\sum_{i=1}^{i=\infty} a_i = \frac{a_1}{1 - r}$$

siempre que el valor absoluto de la razón sea menor o igual que 1.

d) Una función `ProductoHasta` para que multiplique los k primeros elementos de la progresión, aplicando la fórmula:

$$\prod_{i=1}^{i=k} a_1 r^i = \sqrt{(a_1 a_k)^k}$$

Finalidad: Enfatizar la importancia de la ocultación de información. Dificultad Baja.

5. Recuperad la solución del ejercicio 36 de la Relación de Problemas II (pasar de mayúscula a minúscula y viceversa usando un enumerado) Para que el tipo de dato enumerado sea accesible desde dentro de las funciones, debemos ponerlo antes de definir éstas, es decir, en un ámbito global a todo el fichero. Se pide definir las siguientes funciones y cread un programa principal de ejemplo que las llame:

a) `Capitalizacion` nos dice si un carácter pasado como parámetro es una minúscula, mayúscula u otro carácter. A dicho parámetro, llamadlo `una_letra`. La función devuelve un dato de tipo enumerado.

b) `Convierte_a_Mayuscula` comprueba si un carácter pasado como parámetro es minúscula (para ello, debe llamar a la función `Capitalizacion`), en cuyo caso lo transforma a mayúscula. Si el carácter no es minúscula debe dejar la letra igual. A dicho parámetro, llamadlo `caracter`.

Esta función hace lo mismo que la función `tolower` de la biblioteca `cctype`

Observad que el parámetro `una_letra` de la función `Capitalizacion` podría llamarse igual que el parámetro `caracter` de la función `Convierte_a_Mayuscula`. Esto es porque están en ámbitos distintos y para el compilador son dos variables distintas. Haced el cambio y comprobarlo.

c) `Convierte_a_Minuscula` análoga a la anterior pero convirtiendo a minúscula.

Observad que la constante de amplitud

```
const int AMPLITUD = 'a'-'A';
```

es necesaria declararla como constante local en ambas funciones. Para no repetir este código, ¿qué podemos hacer? Implemente la solución adoptada.

d) `CambiaMayusculaMinuscula`, a la que se le pase como parámetro un `char` y haga lo siguiente:

- si el argumento es una letra en mayúscula, devuelve su correspondiente letra en minúscula,
- si el argumento es una letra en minúscula, devuelve su correspondiente letra en mayúscula,
- si el argumento no es ni una letra mayúscula, ni una letra mayúscula, devuelve el carácter pasado como argumento.

Finalidad: Entender cómo se llaman las funciones entre sí. Dificultad Media.

6. En el ejercicio 4 de la relación de problemas I (página RP-I.2) se vio cómo obtener el valor de ordenada asignado por la función gaussiana, sabiendo el valor de abscisa x. Recordemos que esta función matemática dependía de dos parámetros μ (esperanza) y σ (desviación) y venía definida por:

$$\text{gaussiana}(x) = \frac{1}{\sigma\sqrt{2\pi}} \quad e^{\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}}$$

Cread un programa que lea un valor de esperanza y desviación y a continuación lea un número entero `n` que indique el número de abscisas que se van a procesar. Leed un total de `n` valores e imprimid en pantalla el valor de la función gaussiana en dichos valores. El cómputo de la gaussiana debe hacerse en una función.

Ahora estamos interesados en obtener el área que cubre la función en el intervalo $[-\infty, x]$. Dicho valor se conoce como la *distribución acumulada (cumulative distribution function)* en el punto x, abreviado $CDF(x)$. Matemáticamente se calcula realizando la integral:

$$CDF(x) = \int_{-\infty}^{x} \text{gaussiana}(t)dt$$

tal y como puede comprobarse ejecutando el applet disponible en:
http://dostat.stat.sc.edu/prototype/calculators/index.php3?dist=Normal

También pueden obtenerse un valor aproximado de $CDF(x)$, como por ejemplo, el dado por la siguiente fórmula (ver Wikipedia: Normal distribution)

$$CDF(x) = \text{Área hasta } (x) \approx 1 - \text{gaussiana}(x)(b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5)$$

dónde:

$$t = \frac{1}{1 + b_0 x} \qquad b_0 = 0{,}2316419 \;\; b_1 = 0{,}319381530 \;\; b_2 = -0{,}356563782$$

$$b_3 = 1{,}781477937 \;\; b_4 = -1{,}821255978 \;\; b_5 = 1{,}330274429$$

Cread otra función para calcular el área hasta un punto cualquiera x, es decir, $CDF(x)$, usando la anterior aproximación. Modificad el programa principal para que también imprima los valores de CDF correspondientes a los enteros leídos.

Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros. Entender la importancia de la ocultación de información. Dificultad Baja.

Problemas sobre clases

7. En este ejercicio se plantean varias modificaciones. Debe entregar un fichero `cpp` por cada uno de los apartados.

   Se desea implementar una clase `Recta` para representar una recta en el plano. Una recta viene determinada por tres coeficientes `A, B, C`, de forma que todos los puntos `(x,y)` que pertenecen a la recta verifican lo siguiente (*ecuación general de la recta*):

   $$Ax + By + C = 0$$

   a) *Definición de la clase y creación de objetos*

   Defina la clase `Recta`. En este apartado utilice únicamente datos miembro públicos. Cree un programa principal que haga lo siguiente:

   - Defina dos objetos de la clase `Recta`.
   - Lea seis reales desde teclado.
   - Le asigne los tres primeros a los coeficientes de una recta y los otros tres a la segunda recta.
   - Calcule e imprima la pendiente de cada recta aplicando la fórmula:

     ```
     pendiente = - A / B
     ```

   b) *Datos miembro privados*

   Cambie ahora los datos miembro públicos y póngalos privados. Tendrá que añadir métodos para asignar y ver los valores de los datos miembro. Añada un método para asignar un valor a cada uno de los tres datos miembro. Modifique el `main` para tener en cuenta estos cambios.

   A partir de ahora, todos los ejercicios deben resolverse utilizando únicamente datos miembro privados.

   

   c) *Política de acceso a los datos miembros*

   En vez de usar un método para asignar un valor a cada dato miembro, defina un único método para asignar los tres a la misma vez.

   Observad que los métodos permiten definir la política de acceso a los datos miembro. Si tengo previsto cambiar por separado los coeficientes de la recta, usaré métodos de asignación por separado. En caso contrario, usaré un único método.

   d) *Métodos públicos*

   En vez de calcular la pendiente en el programa principal, vamos a ponerlo como un método de la clase y así lo reutilizaremos todas las veces que necesitemos.

Añada un método para el cálculo de la pendiente y modificad el `main` para tener en cuenta este cambio.

¿Añadimos `pendiente` como dato miembro de la recta? La respuesta es que no ¿Por qué?

Añadir también los siguiente métodos:

- Obtener la ordenada (`y`) dado un valor de abscisa `x`, aplicando la fórmula:
  `(-C -xA) / B`
- Obtener la abscisa (`x`) dado un valor de ordenada `y`, aplicando la fórmula:
  `(-C -yB) / A`

e) *Constructor*

Modifique el programa principal del último apartado e imprima los valores de los datos miembros de una recta, **antes** de asignarles los coeficientes. Mostrará, obviamente, un valor indeterminado. Para evitar este problema, añada un constructor a la recta para que el objeto esté en un estado válido en el mismo momento de su definición. El constructor deberá tener como parámetros, obligatoriamente, los tres coeficientes de la recta. Tendrá que modificar convenientemente el `main` para tener en cuenta este cambio.

f) *Política de acceso a los datos miembro*

Suprima ahora los métodos que modificaban los datos miembro. De esta forma, una vez creado el objeto (pasándole los datos apropiados en el constructor) ya no podremos modificar los datos miembro. Esto es útil en aquellas situaciones en las que no queremos permitir que el estado del objeto cambie, una vez que se ha creado.

8. En el ejercicio 4 de esta relación de problemas se definieron varias funciones para operar sobre una progresión geométrica. Definid ahora una clase para representar una progresión geométrica.

   a) Diseñad la clase pensando cuáles serían los datos miembro *esenciales* que definen una progresión geométrica, así como el constructor de la clase.

   b) Definir un método `Termino` que devuelva el término k-ésimo.

   c) Definid los métodos `SumaHastaInfinito`, `SumaHasta`, `MultiplicaHasta`.

   d) Cread un programa principal que lea los datos miembro de una progresión, cree el objeto correspondiente y a continuación lea un entero `tope` e imprima los `tope` primeros términos de la progresión, así como la suma hasta `tope` de dichos términos.

   Finalidad: Comparar la ventaja de un diseño con clases a uno con funciones. Dificultad Baja.

9. Recuperad el ejercicio 6 de esta relación de problemas sobre la función gaussiana. En vez de trabajar con funciones, plantead la solución con una clase.

   Dificultad Media.

10. Se quiere construir una clase `DepositoSimulacion` para simular préstamos, ofreciendo la funcionalidad descrita en los ejercicios 15 (reinvierte capital e interés un número de años) y 16 (reinvierte capital e interés hasta obtener el doble de la cantidad inicial) de la relación de problemas II (página RP-II.7). Por tanto, la clase debe proporcionar, para un capital y unos intereses dados, métodos para:

    a) Calcular el capital que se obtendrá al cabo de un número de años,

    b) Calcular el número de años que deben pasar hasta obtener el doble de la cantidad inicial.

    A la hora de diseñar la clase, tendremos que analizar cuestiones como:

    - ¿Cuáles son sus datos miembro? Parece claro que el capital y el interés sí lo serán ya que cualquier operación que se nos ocurra hacer con un objeto de la clase `DepositoSimulacion` involucra a ambas cantidades. ¿Pero y el número de años?
    - ¿Qué constructor definimos?
    - ¿Queremos modificar el capital y el interés una vez creado el objeto?
    - ¿Queremos poder modificarlos de forma independiente?
    - ¿Hay alguna restricción a la hora de asignar un valor al capital e interés?
    - ¿Es mejor un método para calcular el número de años hasta obtener el doble de la cantidad inicial, o por el contrario es mejor un método para calcular el número de años hasta obtener una cantidad específica?

    Finalidad: Diseñar la interfaz de una clase. Dificultad Baja.

11. Recuperad la solución del ejercicio 9 (actualización de la retención fiscal) de la relación de problemas II. En este problema se leían caracteres de teclado (`'s'/'n'`) para saber si una persona era autónomo, pensionista, etc.

```
do{
   cout << "\n¿La persona es un trabajador autónomo? (s/n) ";
   cin >> opcion;
   opcion = toupper(opcion);
}while (opcion != 'S' && opcion != 'N');
```

    Este código era casi idéntico para la lectura del resto de los datos. Para evitarlo, definid una clase `LectorOpcion` que encapsule esta funcionalidad y cambiar el programa principal para que use esta clase.

12. Recuperad la solución del ejercicio 7 (recta) de esta relación de problemas. Se pide crear un programa principal que haga lo siguiente:

    - Se presentará al usuario un menú principal para salir del programa o para introducir los valores de los coeficientes `A, B, C` de la recta.
    - Una vez introducidos los coeficientes se presentará al usuario un segundo menú, para que elija alguna de las siguiente opciones:
      - Mostrar el valor de la pendiente de la recta.
      - Motrar la ordenada dada una abscisa (el programa tendrá que pedir la abscisa)
      - Mostrar la abscisa dada una ordenada (el programa tendrá que pedir la ordenada)
      - Volver al menú principal.

    Para resolver este problema, debe crear dos clases `MenuPrincipal` y `MenuOperaciones`.

    Finalidad: Trabajar con varias clases en un programa. Dificultad Media.

13. Se quiere construir una clase `Nomina` para realizar la funcionalidad descrita en el ejercicio 12 de la relación de problemas I sobre la nómina del fabricante y diseñador (página RP-I.5). Cread los siguientes programas (entregad un fichero por cada uno de los apartados):

    a) Suponed que sólo gestionamos la nómina de una empresa en la que hay un fabricante y tres diseñadores. Los salarios brutos se obtienen al repartir los ingresos de la empresa, de forma que el diseñador cobra el doble de cada fabricante.

    El programa leerá el valor de los ingresos totales y calculará los salarios brutos de los fabricantes y diseñador, llamando a los métodos oportunos de la clase `Nomina`.

    b) Supongamos que se aplica una retención fiscal y que ésta es la misma para los fabricantes y el diseñador. Una vez que se establezca la retención fiscal, ésta no cambia. Los salarios netos se obtienen al aplicar la retención fiscal a los salarios brutos (después de repartir los ingresos totales de la empresa):

    ```
    salario_neto = salario_bruto -
                   salario_bruto * retencion_fiscal / 100.0
    ```

    El programa leerá el valor de los ingresos totales y la retención fiscal a aplicar y calculará los salarios brutos y netos de los fabricantes y diseñador, llamando a los métodos oportunos de la clase `Nomina`.

    c) Supongamos que gestionamos las nóminas de varias sucursales de una empresa. En cada sucursal hay un único diseñador pero el número de fabricantes es distinto en cada sucursal. La forma de repartir el dinero es la siguiente: el diseñador se lleva una parte del total y el resto se reparte a partes iguales entre los

fabricantes. En los apartados anteriores, por ejemplo, la parte que se llevaba el diseñador era 2/5 y el resto (3/5) se repartía entre los tres fabricantes.

La parte que el diseñador se lleva puede ser distinta entre las distintas sucursales (2/5, 1/6, etc), pero no cambia nunca dentro de una misma sucursal. De la misma forma, siempre hay un único diseñador en cada sucursal, aunque el número de fabricantes puede variar entre distintas sucursales.

Suponed que las retenciones fiscales de los fabricantes y diseñador son distintas.

El programa leerá los siguientes datos desde un fichero externo:

- El número de sucursales.
- Los siguientes valores por cada una de las sucursales:
  - Ingresos totales a repartir
  - Número de fabricantes
  - Parte que se lleva el diseñador
  - Retención fiscal del diseñador
  - Retención fiscal de los fabricantes

Por ejemplo, el siguiente fichero indica que hay dos sucursales. La primera tiene unos ingresos de 300 euros, 3 fabricantes, el diseñador se lleva 1/6, la retención del diseñador es del 20 % y la de cada fabricante un 18 %. Los datos para la segunda son 400 euros, 5 fabricantes, 1/4, 22 % y 19 %.

```
2
300 3 6 20 18
400 5 4 22 19
```

El programa tendrá que imprimir el salario neto del diseñador y de los fabricantes por cada una de las sucursales, llamando a los métodos oportunos de la clase `Nomina`.

Finalidad: Diseño de una clase y trabajar con datos miembro constantes. *Dificultad Media.*

14. Se quiere construir una clase para representar la tracción de una bicicleta, es decir, el conjunto de estrella (engranaje delantero), cadena y piñón (engranaje trasero). Supondremos que la estrella tiene tres posiciones (numeradas de 1 a 3, siendo 1 la estrella más pequeña) y el piñón siete (numeradas de 1 a 7, siendo 1 el piñón más grande). La posición inicial de marcha es estrella = 1 y piñón = 1.

    La clase debe proporcionar métodos para cambiar la estrella y el piñón, sabiendo que la estrella avanza o retrocede de 1 en 1 y los piñones cambian a saltos de uno o de dos. Si ha llegado al límite superior (inferior) y se llama al método para subir (bajar) la estrella, la posición de ésta no variará. Lo mismo se aplica al piñón.

    Cread un programa principal que lea desde un fichero externo los movimientos realizados e imprima la situación final de la estrella y piñón. Los datos se leerán en el

siguiente formato: tipo de plato (pión o estrella) seguido del tipo de movimiento. Para codificar esta información se usarán las siguientes letras: `E` indica una estrella, `P` un piñón, `S` para subir una posición, `B` para bajar una posición, `T` para subir dos posiciones y `C` para bajar dos posiciones. `T` y `C` sólo se aplicarán sobre los piñones.

```
E S P S P S P S P C E S E B
```

En este ejemplo los movimientos serían: la estrella sube, el piñón sube en tres ocasiones sucesivas, el piñón baja dos posiciones de golpe, la estrella sube y vuelve a bajar. Supondremos siempre que la posición inicial de la estrella es 1 y la del piñón 1. Así pues, la posición final será Estrella=1 y Piñón=2.

Mejorad la clase para que no permita cambiar la marcha (con la estrella o el piñón) cuando haya riesgo de que se rompa la cadena. Este riesgo se produce cuando la marcha a la que queremos cambiar es de la siguiente forma:

- Estrella igual a 1 y piñón mayor o igual que 5
- Estrella igual a 2 y piñón o bien igual a 1 o bien igual a 7
- Estrella igual a 3 y piñón menor o igual que 3

Finalidad: Diseñar la interfaz de una clase. Dificultad Media.

Problemas de Ampliación

15. Definid funciones aisladas para implementar las soluciones a los siguientes ejercicios de la relación de problemas II: 25 (serie), 19 (narcisista), 33 (feliz).

    Dificultad Baja.

16. Implemente una clase para representar un número complejo. Un complejo se define como un par ordenado de números reales (a, b), donde a representa la parte real y b la parte imaginaria. Construya un programa principal que lea la parte real e imaginaria, cree el objeto e imprima el complejo en la forma `a + bi`.

    Por ahora no podemos implementar métodos para sumar, por ejemplo, dos complejos. Lo veremos en el último tema.

17. Una empresa quiere gestionar las nóminas de sus empleados. El cómputo de la nómina se realiza en base a los siguientes criterios:

    a) Hay cuatro tipos de categorías laborales: Operario, Base, Administrativo y Directivo.

    b) Se parte de un salario base que depende de la antigüedad del trabajador y de su categoría laboral. Para la categoría Operario, el salario base es de 900 euros, 1100 el puesto Base, 1200 los Administrativos y 2000 los Directivos. Dicho salario base se incrementa con un tanto por ciento igual al número de años trabajados.

    c) Los trabajadores tienen complementos en su nómina por el número de horas extraordinarias trabajadas. La hora se paga distinta según la categoría: 16 euros por hora para los operarios, 23 para el puesto Base, 25 los Administrativos y 30 los Directivos. Además, al complemento que sale al computar el número de horas extraordinarias, se le aplica una subida con un tanto por ciento igual al número de años trabajados.

    Se pide diseñar la interfaz de una clase (también hay que incluir los datos miembro privados) para poder trabajar con esta información. No se pide implementar la clase, únicamente determinar la interfaz.

    Finalidad: Diseñar la interfaz de una clase. *Dificultad Media.*

18. Implementad la clase del ejercicio 17 de esta relación de problemas. *Dificultad Media.*

19. La sonda Mars Climate Orbiter fue lanzada por la NASA en 1998 y llegó a Marte el 23 de septiembre de 1999. Lamentablemente se estrelló contra el planeta ya que se acercó demasiado. El error principal fue que los equipos que desarrollaron los distintos módulos de la sonda usaron sistemas de medida distintos (el anglosajón y el métrico). Cuando un componente software mandaba unos datos en millas (o libras), otro componente software los interpretaba como si fuesen kilómetros (o Newtons). El problema se habría arreglado si todos hubiesen acordado usar el mismo sistema. En

cualquier caso, cada equipo se encuentra más a gusto trabajando en su propio sistema de medida. Por tanto, la solución podría haber pasado por que todos utilizasen una misma clase para representar distancias (idem para fuerzas, presión, etc), utilizando los métodos que les resultasen más cómodos.

Para ello, se pide construir la clase `Distancia` que contendrá métodos como `SetKilometros`, `SetMillas`, etc. Internamente se usará un único dato miembro privado llamado `kilometros` al que se le asignará un valor a través de los métodos anteriores, realizando la conversión oportuna (una milla es $1{,}609344$ kilómetros). La clase también proporcionará métodos como `GetKilometros` y `GetMillas` para lo que tendrá que realizar la conversión oportuna (un kilómetro es $0{,}621371192$ millas).

Observad que la implementación de la clase podría haber utilizado como dato miembro privado, una variable `millas`, en vez de `kilómetros`. Esto se oculta a los usuarios de la clase, que sólo ven los métodos `SetKilometros`, `SetMillas`, `GetKilometros` y `GetMillas`.

Nota. Otro de los fallos del proyecto fue que no se hicieron suficientes pruebas del software antes de su puesta en marcha, lo que podría haber detectado el error. Esto pone de manifiesto la importancia de realizar una batería de pruebas para comprobar el correcto funcionamiento del software en todas las situaciones posibles. Esta parte en el desarrollo de un proyecto se le conoce como *pruebas de unidad (unit testing)*

Finalidad: Trabajar con una clase como una abstracción de un concepto. *Dificultad Baja.*

20. Construid una clase llamada `MedidaAngulo` que represente una medida de un ángulo. Al igual que se hizo en el ejercicio 19, la clase aceptará datos que vengan de alguna de las siguientes formas: número de grados con decimales (real); número de radianes (entero); número de segundos (entero); número de grados, minutos y segundos (en un struct que represente estos tres valores)

    Dificultad Baja.

21. Cread un `struct` llamado `CoordenadasPunto2D` para representar un par de valores reales correspondientes a un punto en $\mathbb{R}^2$. Cread ahora una clase llamada `Circunferencia`. Para establecer el centro, se usará un valor del tipo `CoordenadasPunto2D`. Añadid métodos para obtener la longitud de la circunferencia y el área del círculo interior. Añadid también un método para saber si la circunferencia contiene a un punto. Recordemos que un punto (x_1, y_1) está dentro de una circunferencia con centro (x_0, y_0) y radio r si se verifica que:

$$(x_0 - x_1)^2 + (y_0 - y_1)^2 <= r^2$$

    Observad que el valor de π debe ser constante, y el mismo para todos los objetos de la clase `Circunferencia`.

    Finalidad: Trabajar con constantes estáticas de tipo `double`. *Dificultad Baja.*