Ejercicios de Programación
Introducción al Lenguaje Ada
Programación de Sistemas de Telecomunicación - Informática II

GSyC
Departamento de Teoría de la Señal y Comunicaciones y Sistemas Telemáticos y Computación

7 de septiembre de 2016

NOTA: Al lado de cada ejercicio aparecen una o más cruces, en función de su dificultad y de la cantidad de tiempo que requiere. Cuantas más cruces tenga un ejercicio más difícil es o más tiempo requiere.

Ejercicio 1 +

¿Debería representarse el valor de e (2.71828) en un programa Ada como una constante o como una variable? ¿Por qué?

Ejercicio 2 +

¿Cuáles de los siguientes identificadores pueden utilizarse para nombrar una variable en Ada?:

`MiPrograma`, `programa2`, `prog#2`, `2Programa`, `procedure`, `end`, `"Puntuaciones"`, `Puntuaciones`

Ejercicio 3 +

¿Cuáles de las siguientes expresiones son sentencias de asignación válidas en Ada? ¿Por qué?

- `X = Y;`
- `A := B - C;`
- `P + Q := R;`
- `G := G;`
- `H := 3 + 4;`
- `H := 3 + K;`
- `T := S * T;`

Ejercicio 4 +

Corrige los errores sintácticos del siguiente programa y reescríbelo de acuerdo a las recomendaciones de estilo vistas en clase. Explica qué hace cada sentencia del programa una vez que los hayas corregido. ¿Qué se muestra en la salida estándar?

```
PROCEDURE SMALL;
  X:Float;
  Y:Foat;
  x:Float;
 BEGIN
15.0=Y;
Z=-Y + 3.5;
   Y+z=x;
PUT(X,y,Z)
end small;
```

Ejercicio 5 ++

Escribe un programa que convierta un conjunto de monedas en la cantidad de euros y céntimos de euro equivalente.

Se supondrá que las monedas pueden ser de 5, 10, 20 y 50 céntimos, y de 1 o 2 euros. El programa almacenará en una variable distinta la cantidad de monedas de cada tipo. La cantidad de monedas de cada tipo es siempre mayor o igual a 0.

Asegúrate de probar tu programa con diferentes valores.

Ejemplo:

```
Introduce a continuación el número de monedas de cada tipo.

Monedas de 1 céntimo: 1
Monedas de 2 céntimos: 2
Monedas de 5 céntimos: 0
Monedas de 10 céntimos: 10
Monedas de 20 céntimos: 0
Monedas de 50 céntimos: 5
Monedas de 1 euro: 100
Monedas de 2 euros: 0

Esas monedas son: 103 euros y 55 céntimos de euro.
```

Ejercicio 6 +

Evalúa las siguientes expresiones:

1. `22 / 7`
2. `7 / 22`
3. `22 rem 7`
4. `7 rem 22`
5. `15 / 16`
6. `16 / 15`
7. `15 rem 16`
8. `16 rem 15`
9. `3 / 23`
10. `23 / 3`
11. `3 rem 23`
12. `23 rem 3`
13. `4 / 16`
14. `16 / 4`
15. `4 rem 16`
16. `16 rem 4`

Ejercicio 7 +

Dadas las siguientes definiciones:

```
Pi    : constant Float   := 3.14159;
Max_I : constant Integer := 1000;

X : Float;
Y : Float := -1.0;
A : Integer := 3;
B : Integer := 4;
I : Integer;
```

Indica si las siguientes asignaciones son válidas, y en caso afirmativo cuál es el resultado.

1. `I := A rem B;`
2. `I := (990 - Max_I) / A`
3. `I := A rem Y;`
4. `X := Pi * Y;`

5. `I := A / B;`
6. `X := A / B;`
7. `X := A rem (A / B);`
8. `I := B / 0;`
9. `I := A rem (990 - Max_I);`
10. `I := (Max_I - 990) / A;`
11. `X := A / Y;`
12. `X := Pi ** 2;`
13. `X := Pi ** Y;`
14. `X := A / B;`
15. `I := (Max_I - 990) rem A;`
16. `I := A rem 0;`
17. `I := A rem (Max_I - 990);`

Ejercicio 8 +

Dadas las siguientes definiciones:

```
Pi    : constant Float   := 3.14159;
Max_I : constant Integer := 1000;

X : Float;
Y : Float := 2.0;
A : Integer := 5;
B : Integer := 2;
I : Integer;
```

Indica si las siguientes asignaciones son válidas, y en caso afirmativo cuál es el resultado.

1. `I := A rem B;`
2. `I := (990 - Max_I) / A`
3. `I := A rem Y;`
4. `X := Pi * Y;`
5. `I := A / B;`
6. `X := A / B;`
7. `X := A rem (A / B);`
8. `I := B / 0;`
9. `I := A rem (990 - Max_I);`

10. `I := (Max_I - 990) / A;`
11. `X := A / Y;`
12. `X := Pi ** 2;`
13. `X := Pi ** Y;`
14. `X := A / B;`
15. `I := (Max_I - 990) rem A;`
16. `I := A rem 0;`
17. `I := A rem (Max_I - 990);`

Ejercicio 9 +

Dadas las siguientes líneas de código:

```
Color, Lime, Straw, Yellow, Red, Orange : Integer;
Black, White, Green, Blue, Purple, Crayon : Float;
...
Color  := 2;
Black  := 2.5;
Crayon := -1.3;
Straw  := 1;
Red    := 3;
Purple := 0.3E1;
```

Evalúa las siguientes sentencias de asignación:

1. `White := Crayon * 2.5 / Purple;`
2. `Green := Black / Purple;`
3. `Orange := Color / Red;`
4. `Orange := (Color + Straw) / (2*Straw);`
5. `Lime := Red / Color + Red rem Color;`
6. `Purple := Straw / Red * Color;`

Ejercicio 10 +

Dadas las siguientes definiciones de variables:

```
I, J, K    : Integer;
A, B, C, X : Float;
```

Indica por qué cada una de las siguientes expresiones es inválida:

1. `X := 4.0 A * C;`
2. `A := AC;`
3. `I := 2 * -J;`
4. `K := 3(I + J);`
5. `X := 5A / BC;`
6. `I := 5J3;`

Corrige las expresiones.

Ejercicio 11 +

Escribe un programa que lea tres enteros y los almacene en las variables `X`, `Y` y `Z`, calculando luego su producto y su suma.

Ejercicio 12 +

Escribe un programa que lea un peso (en libras) de un objeto y que calcule y muestre el peso en kilogramos y gramos. Nota: $1libra = 0{,}453592kg$

Ejercicio 13 +

Escribe un programa que calcule la velocidad a la que corre un corredor medida en pies por segundo y km por segundo. Como entrada el programa dispondrá de los tiempos que tarda en recorrer el corredor 1 milla, medidos en minutos y segundos.

Nota: $1milla = 5280pies$. $1km = 3282pies$.

Ejercicio 14 +

Suponiendo que el corazón de un humano late una media de una vez al segundo durante 78 años, haz un programa que calcule cuántas veces late el corazón durante la vida de un humano. Supón que un año tiene 365.25 días.

Ejercicio 15 +

Escribe un programa que muestre en la salida estándar el área y la circunferencia de un círculo. La entrada del programa será el radio del círculo, leído de la entrada estándar.

Ejercicio 16 +

Escribe un programa que muestre el precio de Pizza por metro cuadrado. La entrada del programa deberá ser el precio de la Pizza y su diámetro.

Ejercicio 17 +

Dadas las siguientes definiciones:

```
type Day is (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);

Today : Day := Thursday;
```

Evalúa las siguientes expresiones:

1. `Day'Pos(Monday)`
2. `Day'Pos(Today)`
3. `Day'Val(6)`
4. `Today < Tuesday`
5. `Day'Succ(Sunday)`
6. `Day'Pred(Monday)`
7. `Day'Val(0)`
8. `Today >= Thursday`

Ejercicio 18 ++

Escribe un programa que muestre la fecha actual en el formato `dd/mm/yyyy`. Por ejemplo, si hoy es 11 de febrero de 2001, el programa mostrará `11/02/2001`.

Nota: la fecha actual se puede obtener utilizando el paquete `Ada.Calendar`

Ejercicio 19 ++

Escribe un programa que muestre la fecha actual en el formato `dd de MES de yyyy`, siendo `MES` el nombre del mes. Por ejemplo, si hoy es 11/02/2001, el programa mostrará `11 de febrero de 2001`.

Nota: la fecha actual se puede obtener utilizando el paquete `Ada.Calendar`

Ejercicio 20 +

Escribe un programa que reciba como entrada el día de la semana y muestre como salida el día siguiente y el día anterior al introducido en la entrada.

El programa deberá usar el siguiente tipo:

```
type Días is (Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo);
```

Ejemplo de ejecución:

```
Introduce el nombre de un día de la semana: Domingo
Ayer fue Sábado
Hoy es Domingo
Mañana es Lunes
```

Ejercicio 21 +

Suponiendo que `X` es 15,0 y que `Y` es 25,0, ¿cuál es el valor de las siguientes expresiones booleanas?:

1. `X /= Y`
2. `X < X`
3. `X >= (Y - X)`
4. `X = (Y + X - Y)`

Ejercicio 22 +

Indica si cada uno de los siguientes fragmentos de código es válido o no. Indica respectivamente lo que aparecería en la pantalla o la razón por la que no es válido:

1. ```
   if 12 < 12 then
      Ada.Text_IO.Put("Never");
   else
      Ada.Text_IO.Put("Always");
   end if;
   ```
   ```

2. ```
   if 12 < 15.0 then
      Ada.Text_IO.Put("Never");
   else
      Ada.Text_IO.Put("Always");
   end if;
   ```

3. ```
   Var_1 := 15.0;
   Var_2 := 25.12;
   if (2*Var_1) > Var_2 then
      Ada.Text_IO.Put("OK);
   else
      Ada.Text_IO.Put("Not OK");
   end if;
   ```

4. ```
   Var_1 := 15.0;
   Var_2 := 25.12;
   if (2*Var_1) > Var_2 then
      Ada.Text_IO.Put("OK");
   end if;
   Ada.Text_IO.Put("Not OK");
   ```

Ejercicio 23 +

¿Qué valor se le asigna a `X` en cada uno de los siguientes fragmentos de código, suponiendo que `Y` es 15,0?:

1. ```
   X := 25.0;
   if Y /= (X - 10.0) then
      X := X - 10.0;
   else
      X := X / 2.0;
   end if;
   ```

2. ```
   if Y < 15.0 then
      X := 5 * Y;
   else
      X := 2 * Y;
   end if;
   ```
   ```

Ejercicio 24 +

Escribe un fragmento de código Ada que:

- Si `Item` no es cero, multiplique `Product` por `Item` y guarde el resultado en `Product`. Que muestre el valor de `Product`.
- Almacene en `Z` el valor absoluto de la diferencia entre `X` e `Y`, siendo el valor absoluto `(X - Y)` si este valor es positivo, y si no `(Y - X)`.
- Si `X` es cero, se incrementa en 1 `Zero_Count`. Si `X` es negativo, añadir `X` a `Minus_Sum`. Si `X` es mayor que cero, añadir `X` a `Plus_Sum`

Ejercicio 25 +

Escribe un fragmento de código Ada que intercambie los valores de las variables `X` e `Y` en caso de que `Y > X`

Ejercicio 26 ++

Escribe un programa que calcule el sueldo neto mensual de un empleado de una compañía. El sueldo bruto mensual se calcula como el número de horas trabajadas al mes multiplicado por el coste en euros por hora. El sueldo neto mensual se calcula restando al sueldo bruto la cotización a la Seguridad Social (20 % del sueldo bruto) y el IRPF (15 % de la parte del sueldo bruto que supere 600 euros mensuales).

El programa recibirá como entrada el número de horas trabajadas en un mes y el coste en euros por hora.

Ejemplo de ejecución:

```
Introduce las horas trabajadas en un mes: 160
Introduce el coste en euros por hora: 15

Sueldo bruto mensual: 2400 euros
Sueldo neto mensual : 1650 euros
```

Ejercicio 27 ++

Escribe un programa que calcule el sueldo neto mensual de un empleado de una compañía según el Ejercicio 26, pero pagando las horas que excedan las 160 mensuales un 50 % más que las normales.

Ejercicio 28 +

Escribe una única sentencia `if` que muestre en pantalla la nota a partir de la nota numérica según la siguiente tabla:

- $[0, 5)$: Suspenso
- $[5, 7)$: Aprobado
- $[7, 9)$: Notable
- $[9, 10]$: Sobresaliente

Ejercicio 29 ++

Escribe un programa que muestre en la salida estándar el mayor y el menor de tres números enteros que son introducidos en el teclado por el usuario.

Ejemplo de ejecución:

```
Introduce el primer número entero: -29
Introduce en segundo número entero: 574
Introduce el tercer número entero: 0

El número menor de los tres es el -29 y el mayor el 574
```

Ejercicio 30 ++

Escribe un programa que lea cuatro letras y que muestre la que aparece primero en el alfabeto.

Ejemplo de ejecución:

```
Introduce cuatro letras y luego pulsa <ENTER>: THEM
De las cuatro introducidas, E es la que primero aparece en el alfabeto
```

Ejercicio 31 ++

Escribe un programa que lea cuatro letras y que muestre la última que aparece en el alfabeto.

Ejemplo de ejecución:

```
Introduce tres letras y luego pulsa <ENTER>: THEM
De las tres introducidas, T es la última que aparece en el alfabeto
```

Ejercicio 32 ++

La compañía Telefonge tiene las siguientes tarifas de telefonía móvil:

- Las llamadas realizadas entre las 6:00PM y las 8:00AM tienen un descuento de 50 %
- Las llamadas entre las 8:00AM y las 6:00PM se cobran a la tarifa normal
- Todas las llamadas tienen un impuesto de 4 %
- La tarifa normal es de 0.15 céntimos de euro por minuto
- Las llamadas de más de 60 minutos tienen un descuento adicional de 15 % (antes de añadir el impuesto)

Escribe un programa que permita que un usuario introduzca la hora de comienzo y la duración de una llamada y que muestre el coste bruto (antes de aplicar descuentos y el impuesto), y el coste neto (tras aplicar descuentos e impuesto).

Ejercicio 33 ++

Escribe un paquete `Trafico` que exporte la siguiente función:

```
function Clasifica (Velocidad: Rango_Velocidad) return Clases;
```

Esta función devuelve la clase de velocidad que se le pasa como parámetro de acuerdo a la siguiente tabla:

- Lenta: 0 km/s < velocidad <= 60 km/s
- Moderada: 60 km/s < velocidad <= 70 km/s
- Correcta: 70 km/s < velocidad <= 90 km/s
- Rápida: 90 km/s < velocidad <= 100 km/s
- Punible: 100 km/s < velocidad

Realiza un programa principal que permita al usuario introducir una velocidad y que muestre la clase de tráfico haciendo uso de la función `Trafico.Clasifica`

Ejercicio 34 +

Escribe un programa que muestre la suma de los enteros de 1 a N. La entrada del programa es el número `Positive` N.

Ejemplo de ejecución:

```
Introduce el último número de la suma: 25
La suma de los enteros de 1 a 25 es 325
```

Ejercicio 35 +

Para cada uno de los siguientes programas indica si es legal o no, y caso afirmativo qué muestra.

- ```
  with Ada.Text_IO;
  procedure Loop_Test is
  begin
     for Count in 1..10 loop
        Ada.Text_IO.Put(Integer'Image(Count));
     end loop;
     Ada.Text_IO.New_Line;
     Ada.Text_IO.Put(Integer'Image(Count));
     Ada.Text_IO.New_Line;
  end Loop_Test;
  ```

- ```
  procedure Loop_Test is
     Count: Positive;
  begin
     Count := 537;
     for Count in 1..10 loop
        Ada.Text_IO.Put(Integer'Image(Count));
     end loop;
     Ada.Text_IO.New_Line;
     Ada.Text_IO.Put(Integer'Image(Count));
     Ada.Text_IO.New_Line;
  end Loop_Test;
  ```

Ejercicio 36 +

Escribe un fragmento de código que calcule la suma de los cuadrados de los enteros del 1 al 10.

Ejercicio 37 +

Escribe un bucle `for` que muestre la siguiente línea:

```
10 9 8 7 6 5 4 3 2 1
```

- utilizando `reverse` en el bucle `for`
- sin utilizar `reverse`

Ejercicio 38 ++

Escribe un programa que sume un conjunto indeterminado de números enteros.

Como entrada se proporcionará el número de enteros que se desean sumar y la lista de los mismos.

Ejemplo de ejecución:

```
¿Cuántos números vas a sumar (al menos 1)?: 3
Introduce el número  1: 23
Introduce el número  2: 12
Introduce el número  3: -3

La suma de los 3 números es: 32
```

Ejercicio 39 ++

Escribe un programa que muestre el mínimo, máximo y la media de una lista de tamaño indeterminado de números.

Como entrada se proporcionará el número de enteros y la lista de los mismos.

Ejemplo de ejecución:

```
¿Cuántos números vas a introducir (al menos 1)?: 3
Introduce el número  1: 23
Introduce el número  2: 12
Introduce el número  3: -3

El más pequeño es: -3
El mayor es: 23
La media es: 10
```

Ejercicio 40 ++

Modifica el programa del Ejercicio 27 para que calcule el sueldo bruto y neto de un conjunto de empleados.

Ejercicio 41 +++

Escribe un programa que muestre un triángulo isósceles. Como entrada se proporcionará la altura del mismo medida en líneas de texto.

Ejemplo de ejecución:

```
Introduce la altura medida en líneas de texto: 5
    *
   ***
  *****
 *******
*********
```

Ejercicio 42 +

¿Qué muestran en pantalla los siguientes fragmentos de código, suponiendo que `M` es 3 y `N` es 5?:

- ```
  for I in 1..N loop
      for J in 1..I loop
         Ada.Text_IO.Put('*');
      end loop;
      Ada.Text_IO.New_Line;
    end loop;
  ```

- ```
  for I in 1..N loop
      for J in 1..M loop
         Ada.Text_IO.Put('*');
      end loop;
      Ada.Text_IO.New_Line;
    end loop;
  ```

Ejercicio 43 +

¿Cuál es la salida que muestra el siguiente fragmento de código?:

```
for I in 1..2 loop
   Ada.Text_IO.Put("Outer");
   Ada.Text_IO.Put(Integer'Image(I));
   for J in 1..3 loop
      Ada.Text_IO.Put("Inner  ");
      Ada.Text_IO.Put(Integer'Image(I));
      Ada.Text_IO.Put(Integer'Image(J));
   end loop;
   for K in reverse 1..2 loop
      Ada.Text_IO.Put("Inner  ");
      Ada.Text_IO.Put(Integer'Image(I));
      Ada.Text_IO.Put(Integer'Image(K));
   end loop;
end loop;
```

Ejercicio 44 +

Escribe un programa que muestre la siguiente salida:

```
Introduce un número entre 1 y 9: 4
1
1 2
1 2 3
1 2 3 4
1 2 3
1 2
1

Introduce un número entre 1 y 9: 3
1
1 2
1 2 3
1 2
1
```

Ejercicio 45 ++

Escribe un programa que muestre una tabla de sumas para números entre 0 y 9:

```
Introduce un número entre 0 y 9: 3
+  0  1  2  3
0  0  1  2  3
1  1  2  3  4
2  2  3  4  5
3  3  4  5  6

Introduce un número entre 0 y 9: 4
+  0  1  2  3  4
0  0  1  2  3  4
1  1  2  3  4  5
2  2  3  4  5  6
3  3  4  5  6  7
4  4  5  6  7  8
```

Ejercicio 46 ++

Escribe un programa que pida al usuario un número N y a continuación muestre una tabla con la suma de 1..N y el factorial de N.

Ejemplo de ejecución:

```
Introduce un entero de 1 a 10: 9
N    Suma    Factorial
----------------------
1       1        1
2       3        2
3       6        6
4      10       24
5      15      120
6      21      720
7      28     5040
8      36    40320
9      45   362880
```

Ejercicio 47 ++

Añade un manejador de excepciones al programa 46.

Si se introduce un número que no esté en el rango 1..10 el manejador mostrará el siguiente mensaje: `número fuera del rango 1..`

Si se introduce un valor no entero el manejador mostrará el siguiente mensaje: `el valor introducido no es un entero`

Ejercicio 48 +

Para cada uno de los siguientes fragmentos de código indica cuántas veces se ejecuta cada sentencia `Put` y cuál es el último valor mostrado:

- ```
  for I in 1..10 loop
     for J in 1..5 loop
        Ada.Text_IO.Put(Integer'Image(I*J));
     end loop;
     Ada.Text_IO.New_Line;
  end loop;
  ```

- ```
  for I in 1..10 loop
     for J in 1..I loop
        Ada.Text_IO.Put(Integer'Image(I*J));
     end loop;
     Ada.Text_IO.New_Line;
  end loop;
  ```

- ```
  for Counter in 1..5 loop
     Ada.Text_IO.Put(Integer'Image(Counter));
  end loop;
  Ada.Text_IO.Put(Integer'Image(Counter));
  ```
  ```

Ejercicio 49 +

Escribe una sentencia `for` que recorra los valores de `'Z'` a `'A'` y muestre sólo las consonantes.

Nota: el operador `in` sirve para comprobar si un valor pertenece a un subtipo. Ejemplo: `if Variable in Subtipo then ...`

Ejercicio 50 ++

Escribe dos bucles `for` anidados que muestren las primeras seis letras del alfabeto en una línea, las siguientes 5 letras en una nueva línea, las siguientes 4 en la siguiente,... hasta llegar a una línea en la que sólo se muestre una letra.

Ejercicio 51 +

Escribe un programa que muestre una tabla con las primeras 15 potencias de 2, empezando por 2^0

Ejercicio 52 ++

Escribe un programa que lea 20 números enteros y muestre la cantidad de valores que son positivos (>= 0) y la cantidad de valores que son negativos. En función del resultado, el programa mostrará al final el texto `"más positivos"`, `"más negativos"` o `"igual número de positivos que de negativos"`.

Ejercicio 53 ++

Modifica el programa 16 para que permita calcular el precio por metro cuadrado de un número de pizzas. El número se leerá al comenzar el programa.

Ejercicio 54 ++

Escribe un programa que pida al usuario un mes y año inicial y un mes y año final. El programa escribirá en un **fichero de texto** una línea para cada día de ese periodo. Cada línea del fichero mostrará el día, mes y año.

Ejercicio 55 +

Escribe un programa que muestre los números impares comprendidos entre 1 y 39, ambos incluídos.

Escribe primero el programa usando un bucle `loop` y luego usando un bucle `while`.

Ejercicio 56 ++

Escribe un programa que muestre la tabla de equivalencia entre grados Celsius y grados Kelvin, de 10 en 10, para el rango comprendido entre 100 grados Celsius y -20 grados Celsius, en orden descendente.

Nota: $Fahrenheit = (1'8 * Celsius) + 32$

Ejemplo de ejecución:

```
Celsius    Fahrenheit

 100.0        212.0
  90.0        194.0
  80.0        176.0
   ...          ...
 -10.0         14.0
 -20.0         -4.0
```

Escribe primero el programa usando un bucle `loop` y luego usando un bucle `while`.

Ejercicio 57 ++

Escribe un programa que muestre la distancia a la que se va encontrando un objeto de la pared. Cada vez que el objeto se mueve lo hace en una cantidad igual a su tamaño.

El tamaño del objeto es de 8.5 cm. La distancia inicial la introduce el usuario.

Ejemplo de ejecución:

```
Distancia inicial del objeto a la pared: 27

La distancia es     27.00
La distancia es     18.50
La distancia es     10.00
...
Distancia final entre el objeto y la pared:     1.50
```

Escribe primero el programa usando un bucle `loop` y luego usando un bucle `while`.

Ejercicio 58 +

¿Cuántas veces se ejecuta el cuerpo del siguiente bucle? ¿Qué se muestra en cada vuelta del bucle?

```
X := 3;
Count := 0;
loop
   X := X * X;
   Ada.Text_IO.Put(Integer'Image(X));
   Count := Count + 1;
   exit when Count >= 3;
end loop;
```

Ejercicio 59 +

¿Cuántas veces se ejecuta el cuerpo del siguiente bucle? ¿Qué se muestra en cada vuelta del bucle?

```
X := 3;
Count := 0;
loop
   X := X * X;
   Ada.Text_IO.Put(Integer'Image(X));
   Count := Count + 2;
   exit when Count >= 3;
end loop;
```

Ejercicio 60 +

¿Cuántas veces se ejecuta el cuerpo del siguiente bucle? ¿Qué se muestra en cada vuelta del bucle?

```
X := 3;
Count := 0;
loop
   X := X * X;
   Ada.Text_IO.Put(Integer'Image(X));
   exit when Count >= 3;
end loop;
```

Ejercicio 61 ++

En un momento dado viven en una ciudad 9870 personas. Suponiendo que la población crece un 10 % anual, escribe un bucle que calcule cuántos años se necesitan para que la población supere los 30000 habitantes.

Ejercicio 62 +

Escribe un bucle que muestre una tabla con n y 2^n mientras que 2^n sea menor de 10000.

Ejercicio 63 ++

Escribe un programa que calcule la suma de los números naturales introducidos por el teclado. En cada iteración se preguntará si se quiere seguir introduciendo valores o no.

Ejemplo de ejecución:

```
Introduce el siguiente número: 33
¿Más datos? S/N: S
Introduce el siguiente número: 55
¿Más datos? S/N: S
Introduce el siguiente número: 77
¿Más datos? S/N: N

La suma total es 165
```

Ejercicio 64 ++

Escribe un programa que calcule la suma de los números naturales introducidos por el teclado hasta que se introduzca el número -1.

Ejemplo de ejecución:

```
Introduce el siguiente número: 33
Introduce el siguiente número: 55
Introduce el siguiente número: 77
Introduce el siguiente número: -1

La suma total es 165
```

Ejercicio 65 ++

Escribe un programa que calcule el producto de una colección de valores enteros no nulos. Si hay varias ocurrencias consecutivas de un mismo número sólo se incluirá en el producto una ocurrencia. Por ejemplo, el producto de los números 10, 5, 5, 5 y 10 debe calcularse como $10 * 5 * 10 = 500$

Se irán leyendo números enteros hasta que se introduzca el valor 0.

Ejemplo de ejecución:

```
Introduce 0 para parar.
Introduce el primer    número: 10
Introduce el siguiente número: 5
Introduce el siguiente número: 5
Introduce el siguiente número: 5
Introduce el siguiente número: 10
Introduce el siguiente número: 0

El producto es 500
```

Escribe primero el programa usando un bucle `loop` y luego usando un bucle `while`.

Ejercicio 66 +

Indica qué valores muestra el siguiente fragmento de código cuando se introduce por el teclado el valor 5:

```
Ada.Text_IO.Put("Introduce un entero: ``);
X := Integer'Value(Ada.Text_IO.Get_Line);
Producto := X;
Contador := 0;
loop
   Ada.Text_IO.Put(Integer'Image(Producto));
   Producto := Producto * X;
   Contador := Contador + 1;
   exit when Contador >= 4;
end loop;
```

Ejercicio 67 +

Indica qué valores muestra el siguiente fragmento de código cuando se introduce por el teclado el valor 5:

```
Ada.Text_IO.Put("Introduce un entero: ``);
X := Integer'Value(Ada.Text_IO.Get_Line);
Producto := X;
Contador := 0;
loop
   Producto := Producto * X;
   Contador := Contador + 1;
   Ada.Text_IO.Put(Integer'Image(Producto));
   exit when Contador >= 4;
end loop;
```

Ejercicio 68 +

Escribe un programa que calcule $1 + 2 + 3 + ... + (N - 1) + N$, siendo N un valor leído del teclado.

Ejercicio 69 ++

Reescribe el siguiente código añadiéndole un manejador de excepciones para que cuando en el bucle no se introduzca un valor válido el bucle siga ejecutándose sin que pare el programa.

```
with Ada.Text_IO;
procedure Exception_Loop is
   Min_Val : constant Integer := -10;
   Max_Val : constant Integer :=  10;
   subtype Small_Int is Integer range Min_Val .. Max_Val;
   Input_Value : Small_Int;
   Sum :         Integer;
begin
   Sum := 0;

   for Contador in 1..5 loop
      Ada.Text_IO.Put("Introduce un entero entre ");
      Ada.Text_IO.Put (Integer'Image(Small_Int'First));
      Ada.Text_IO.Put(" y ");
      Ada.Text_IO.Put (Integer'Image(Small_Int'Last));
      Ada.Text_IO.Put(" : ");
      Input_Value := Integer'Value(Ada.Text_IO.Get_Line);

      Sum := Sum + Input_Value;
   end loop;

   Ada.Text_IO.Put ("La suma es ");
   Ada.Text_IO.Put(Integer'Image(Sum));
   Ada.Text_IO.New_Line;
end Exception_Loop;
```

Ejemplo de ejecución:

```
Introduce un entero entre -10 y 10 : 20
Valor fuera de rango. Por favor, inténtalo de nuevo.
Introduce un entero entre -10 y 10 : -11
Valor fuera de rango. Por favor, inténtalo de nuevo.
Introduce un entero entre -10 y 10 : x
El valor no es un entero. Por favor, inténtalo de nuevo.
Introduce un entero entre -10 y 10 : 0
Introduce un entero entre -10 y 10 : -5
Introduce un entero entre -10 y 10 : y
El valor no es un entero. Por favor, inténtalo de nuevo.
Introduce un entero entre -10 y 10 : 3
Introduce un entero entre -10 y 10 : 4
Introduce un entero entre -10 y 10 : -7
La suma es -5
```

Ejercicio 70 ++

Escribe un programa que muestre un menú de mandatos en pantalla y tras leer el mandato seleccionado por el usuario imprima un mensaje explicando la opción del menú elegida.

Ejemplo de ejecución:

```
Menú de mandatos

+ : Subir el volumen
- : Bajar el volumen
s : Silenciar

Introduzca un mandato: +
  Subiendo el volumen
Introduzca un mandago: *
  Mandato no reconocido
Introduzca un mandato: -
  Bajando el volumen
```

Ejercicio 71 ++

Modifica el programa 70 para que el usuario sólo tenga tres intentos para introducir un mandato correcto. Si introduce 3 mandatos seguidos incorrectos el programa deberá terminar. Ejemplo de ejecución:

```
Menú de mandatos

+ : Subir el volumen
- : Bajar el volumen
s : Silenciar

Introduzca un mandato: +
  Subiendo el volumen
Introduzca un mandago: a
  Mandato no reconocido
Introduzca un mandago: b
  Mandato no reconocido
Introduzca un mandago: c
  Mandato no reconocido

3 mandatos incorrectos: fin del programa
```

Ejercicio 72 +

Escribe un subprograma que tenga dos parámetros formales de tipo entero. El subprograma deberá devolver ordenados los dos parámetros, el menor en el primer parámetro formal y el mayor en el segundo.

Ejercicio 73 +

Usando el subprograma del Ejercicio 72 escribe un programa que lea 3 valores enteros y los escriba ordenados de mayor a menor.

Ejemplo de ejecución:

```
Introduzca el primer valor entero: 12
Introduzca el segundo valor entero: 2
Introduzca el tercer valor entero: -7
Los tres números ordenados son: -7 2 12
```

Ejercicio 74 ++

Escribe un programa que calcule el producto de una colección de valores enteros. El programa debe terminar cuando se lea el valor 0.

Ejercicio 75 ++

Escribe un programa que lea un entero N y calcule `Slow = 1 + 2 + 3 + ... + N`. Luego calculará `Fast = (N*(N+1))/2` y comparará los valores de Slow y Fast. El programa mostrará en pantalla los dos valores e indicará si son o no iguales.

Ejercicio 76 ++

Escribe un programa que lea una lista de valores enteros hasta que se introduzca el valor 0, y que muestre en pantalla la posición que ocupa la primera y última ocurrencias del valor 12. El programa debe mostrar el valor 0 si el número 12 no estaba en la colección de valores leídos. Si el número 12 sólo aparece una vez, la primera y última ocurrencias serán iguales.

Ejercicio 77 ++

Escribe un programa que lea una colección de notas de examen entre 0 y 10. El programa debe contar y mostrar el número de sobresalientes (>= 9 y <= 10), el número de notables (>= 7 y <= 9), el número de aprobados (>= 5 y < 7) y el número de suspensos (< 5).

El programa leerá notas hasta que la nota introducida esté fuera del rango. Para cada nota mostrará al lado su calificación (suspenso, aprobado, notable, sobresaliente). Al final mostrará el número de notas de cada calificación y la media de todas las notas.

Ejercicio 78 +++

Escribe un programa que permita gestionar el inventario de un distribuidor de bebidas.

Existen 4 tipos de bebidas: Cocacola (1), Pepsi (2), Trinaranjus (3), KongaCola (4) y sus precios respectivamente son 10, 20, 30 y 40 céntimos de euro.

El programa deberá mostrar el siguiente menú:

```
C: Comprar un número de botellas de una bebida
V: Vender  un número de botellas de una bebida
M: Mostrar el inventario
D: Mostrar el dinero en caja en euros
T: Terminar el programa
```

La cantidad inicial de botellas de cada marca y la cantidad de dinero inicial se leerán de un **fichero de texto**.

Al terminar el programa se actualizará el fichero con el inventario y la cantidad de dinero existentes en el momento de terminar el programa.

Ejercicio 79 +++

La raíz cuadrada de un número N puede calcularse por aproximaciones sucesivas con la siguiente fórmula:

`Estimación_Nueva = 0.5 (Estimación_Anterior + (N / Estimación_Anterior))`

Escribe una función `Sqrt` que calcule por aproximaciones sucesivas la raíz cuadrada de un número `Float` positivo.

La función deberá devolver un valor cuando la diferencia entre los dos últimos valores estimados sea manor de 0,005

Para el valor estimado inicial utiliza el valor 1,0

Ejercicio 80 ++

Escribe un programa que lea una fecha en el formato día, mes, año de forma robusta. Utiliza un tipo enumerado para los nombres de los meses.

La fecha introducida ha de ser una fecha válida (p.ej. no existe el 31 de febrero de ningún año). Para comprobar la validez de las fechas consulta el paquete `Ada.Calendar`.

Ejercicio 81 +++

Escribe un programa que muestre los dígitos de un número `Positive` en orden inverso.

Ejemplo de ejecución:

```
Introduce un entero positivo: 9752013
Los dígitos en orden inverso son: 3 1 0 2 5 7 9
```

Ejercicio 82 +++

Escribe un programa que muestre la hora actual en el formato `hh:mm:ss`.

Ejemplo de ejecución:

```
La hora actual: 20:22:05
```

Ejercicio 83 +++

Escribe un programa que compruebe si un entero positivo es un número primo. El programa mostrará un mensaje diciendo si el número es primo, o mostrará su divisor más pequeño si no es primo.

Ejercicio 84 ++

Escribe un programa que lea una colección de caracteres hasta que se introduzca un punto. El programa mostrará el número de espacios en blanco introducidos.

Ejemplo de ejecución:

```
Introduzca una secuencia de caracteres en la que el último carácter sea un punto:
La vaca que ríe.
El número de espacios es 3
```

Ejercicio 85 ++

Escribe una función que reciba como argumento un carácter y devuelva un carácter. Si el argumento es una letra minúscula deberá devolver la misma letra pero en mayúscula. Si el argumento ya es una letra mayúscula deberá devolverla sin modificar.

Nota: todas las letras minúsculas están juntas, así como las mayúsculas. Utiliza los atributos `'Pos` y `'Val` del tipo `Character`.

Ejercicio 86 +

Evalúa las siguientes expresiones teniendo en cuenta que todas las letras minúsculas están juntas en el tipo Character, así como las mayúsculas.

1. `Character'Pos('D') - Character'Pos('A')`
2. `Character'Pos('d') - Character'Pos('a')`
3. `Character'Succ(Character'Pred('a'))`
4. `Character'Val(Character'Pos('C'))`
5. `Character'Val(Character'Pos('C') - Character'Pos('A')+Character'Pos('a'))`
6. `Character'Pos('7') - Character'Pos('6')`
7. `Character'Pos('9') - Character'Pos('0')`
8. `Character'Succ(Character'Succ(Character'Succ('d')))`
9. `Character'Val(Character'Pos('A') + 5)`

Ejercicio 87 +

Dadas las siguientes declaraciones:

```
type Mes is (Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto,
             Septiembre, Octubre, Noviembre, Diciembre);
```

Y las variables `Este_Año`, con valores en el rango 1901..2099, `Dias_En_Mes` de tipo `Positive` y `Este_Mes` de tipo `Mes`, escribe una sentencia `case` que almacene en `Dias_En_Mes` el número de días en `Este_Mes` teniendo en cuenta el año `Este_Año`. Ten en cuenta los años bisiestos.

Ejercicio 88 +

Escribe una sentencia `if` que sea equivalente a la siguiente sentencia `case`:

```
case X > Y is
   when True =>
      Ada.Text_IO.Put(Item => "X mayor");
   when False =>
      Ada.Text_IO.Put(Item => "Y mayor o igual");
end case;
```

Ejercicio 89 +

Dada esta definición de tipo:

```
type Color is (Red, Green, Blue, Brown, Yellow);
```

escribe una sentencia `case` que asigne un valor a la variable `Eyes`, de tipo `Color`, suponiendo que las dos primeras letras del color están almacenadas en dos variables de tipo `Character`, `Letra_1, Letra_2`.

Ejercicio 90 ++

Escribe una sentencia `case` que muestre un mensaje indicando si `Next_Character`, de tipo `Character` es uno de los siguientes caracteres: un *operador* (`+, -, *, =, <, >, /`), un *signo de puntuación* (punto, dos puntos, coma, punto y coma, paréntesis, llave, corchete) o un *dígito*. El código debe mostrar la categoría (operador, signo de puntuación o dígito).

Reescríbelo con un `if`.

Ejercicio 91 +

Evalúa la siguiente expresión:

```
True and ((30 rem 10) = 0)
```

Ejercicio 92 +

Evalúa la siguiente expresión:

```
False and (((30 rem 10) / 0) = 0)
```

¿Qué ocurriría si en lugar de `and` se usa `and then`?

Ejercicio 93 +

Evalúa las siguientes expresiones:

```
Character'Val(Character'Pos('a'))
Character'Val(Character'Pos('a') + 3)
Character'Val(Character'Pos('z') - 26)
Character'Val(Character'Pos('z') - 32)
```

Ejercicio 94 +++

Un entero N es divisible por 9 si la suma de sus dígitos es divisible por 9. Escribe un programa que compruebe utilizando este algoritmo si un número introducido por el usuario es divisible por 9 o no.

Ejercicio 95 +++

El valor de e^x puede calcularse utilizando la siguiente serie:

$1 + x + x^2/2! + x^3/3! + ... + x^n/n! + ...$

Escribe un subprograma `My_Exp` para calcular e imprimir el valor de esta serie para cualquier x y n.

Escribe un programa que permita al usuario introducir un valor x y que calcule utilizando el subprograma anterior el valor de e^x. El programa deberá comprobar si el resultado es correcto comparándolo con el resultado de la función `Ada.Numerics.Generic_Elementary_Functions.Exp`. Será correcto cuando la diferencia entre el valor calculado por `My_Exp` y el devuelto por `Exp` sea menor de 0.001.

Ejercicio 96 ++

Dada la siguiente definición:

```
type Point is record
   X : Float;
   Y : Float;
end record;
```

Escribe un programa que lea dos puntos y calcule d, su distancia Euclídea:

$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

Ejemplo de ejecución:

```
Introduce las coordenadas X e Y del primer Punto:
  X: 3
  Y: 4
Introduce las coordenadas X e Y del segundo Punto:
  X: 9
  Y: 12

Distancia entre los puntos: 10.00
```

Ejercicio 97 ++

Escribe un programa que lea una colección de 8 valores `Float` del terminal, calcule la media de los valores, y muestre la tabla de diferencias entre los valores leídos y la media.

Ejemplo de ejecución:

```
Introduce 8 valores de tipo Float:
16 12 6 8 2.5 12 14 -54.5

La media es: 2.00
Tabla de diferencias
   I      X(I)    Diferencia
   1     16.00      14.00
   2     12.00      10.00
   3      6.00       4.00
   4      8.00       6.00
   5      2.50       0.50
   6     12.00      10.00
   7     14.00      12.00
   8    -54.50     -56.50
```

Ejercicio 98 +

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Dados los contenidos del array `X` de valores de tipo `Float` mostrado en la figura, indica qué ocurre cuando se ejecutan las siguientes sentencias y cuál es elcontenido final del array cuando se han ejecutado todas las sentencias:

```
I := 3;

X(I) := X(I) + 10.0;

X(I - 1) := X(2 * I - 1);

X(I + 1) := X(2 * I) + X(2 * I + 1);

for I in 5..7 loop
   X(I) := X(I + 1);
end loop;

for I in reverse 1..3 loop
   X(I + 1) := X(I);
end loop;
```

Ejercicio 99 +

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Dados los contenidos del array `X` de valores de tipo `Float` mostrado en la figura, escribe sentencias que permitan realizar las siguientes acciones sobre el Array:

1. Substituir el 3er elemento por 7.0
2. Copiar el 5º elemento en el primero
3. Restarle al 4º elemento el 1º y almacenar el resultado en el 5º
4. Sumar 2 al 6º elemento
5. Calcular la suma de los primeros 5 elementos
6. Multiplicar cada uno de los 6 primeros elementos por 2 y almacenar el resultado en el elemento correspondiente de otro array `Array_De_Respuesta`
7. Mostrar los contenidos de los elementos pares del array `X`

Ejercicio 100 +

Dadas las siguientes declaraciones:

```
A,B,C,D : array(1..5) of Integer;

type List is array (1..5) of Integer;
E,F,G,H: List;
```

Explica si las siguientes sentencias son válidas o no, y por qué:

1. A := B;
2. C := D;
3. E := F;
4. G := H;

Ejercicio 101 +

Declara tipos array que sirvan para almacenar la información que se especifica a continuación. Define subtipos adecuados para representar los índices de los arrays.

Ejemplo: Array para almacenar el nombre de una persona, con un máximo de 10 caracteres:

```
subtype Longitud_Nombre is Positive range 1..10;
type Nombre is array(Longitud_Nombre) of Character;
```

1. Array para almacenar la conversión a temperaturas en grados Fahrenheit de un conjunto de temperaturas en grados Celsius entre -10 y 10.
2. Array para almacenar el número de veces que aparece cada letra mayúscula en un texto.
3. Array para almacenar flags (valores booleanos) que indiquen si una letra (mayúscula o minúscula) aparece o no en un texto.
4. Array para almacenar si las respuestas a las preguntas de un examen son correctas o no, con un máximo de 100 preguntas.
5. Las áreas de cada habitación de una casa (comedor, cocina,...)
6. El número de estudiantes en cada curso de un colegio (desde educación primaria hasta bachillerato)

Ejercicio 102 +++

Un número expresado en notación científica se representa con una mantisa (la parte fraccionaria) y un exponente. Ejemplo: `-0.1234E20` es el número $-0{,}1234 * 10^{20}$

Escribe un procedimiento que lea un conjunto de caracteres representando un número en notación científica y que devuelva un registro de tipo `Mantisa_Exponente` de dos campos en el que se almacenen la mantisa y el exponente.

Escribe una función que devuelva el número de tipo `Float` correspondiente al valor del registro de tipo `Mantisa_Exponente` que recibe como argumento.

Escribe funciones que reciban 2 argumentos de tipo `Mantisa_Exponente` y devuelvan un valor de ese mismo tipo para sumar, y multiplicar valores números en notación científica.

Escribe un paquete que exporte estos subprogramas y un programa principal que permita comprobar su correcto funcionamiento.

Ejercicio 103 +++

Escribe un programa que lea dos números `Positive` con una longitud máxima de 9 dígitos decimales, y los sume.

Cada número se almacenará en un array, ocupando cada dígito decimal una posición del array.

La suma deberá implementarse sobre los números representados en los arrays.

La suma se almacenará en un tercer array y el resultado final se mostrará en la pantalla como un número `Positive`.

Ejercicio 104 +++

Escribe un programa que muestre la cantidad de veces que aparece una letra en un texto almacenado en un **fichero de texto** que se pasa como argumento al programa (usar `Ada.Command_Line` para obtener el nombre del fichero). Las ocurrencias de una letra minúscula y la misma letra en mayúscula se sumarán.

Ejemplo de ejecución:

```
Letra   Contador
    A          3
    B         15
    C         32
  ...
    Z         12
```

Ejercicio 105 +++

Si no existe un número primo menor que N que lo divida, N es un número primo. Por tanto se puede determinar si N es primo dividiéndolo sólo entre los números primos menores que él.

Haciendo uso de esta afirmación, escribe un programa que calcule los 100 primeros números primos y los vaya almacenando en un array según los vaya calculando. Al terminar deberá mostrarlos en pantalla.

Ejercicio 106 +++

Escribe una función `Reverse` que reciba como argumento un `String` y devuelva el mismo `String` dado la vuelta. Ejemplo: si se le pasa como argumento `"hola a todos"` devolverá el `String` `"sodot a aloh"`.

Escribe un programa que haciendo uso de la función `Reverse` compruebe si un texto introducido por el usuario es un palíndromo. Un palíndromo es una cadena de texto que se lee igual del derecho que del revés. Por ejemplo la palabra `"reconocer"` es un palíndromo.

Ejercicio 107 +++

Escribe un programa que lea cada línea de un **fichero de texto** y la muestre junto a su número de línea en caso de que la longitud de la línea exceda el tamaño máximo de línea.

El nombre del fichero y el tamaño máximo de la línea se introducirán como argumentos del programa (usar `Ada.Command_Line` para obtenerlos).

Al terminar se mostrará el tamaño de la línea más larga del fichero.

Nota: La función `Ada.Text_IO.End_Of_File (Fichero)` devuelve `True` si no hay más caracteres que leer en el fichero de texto.

Ejercicio 108 +++

Escribe un programa que cuente las ocurrencias de las longitudes de las palabras de un fichero de texto. El nombre del fichero se pasará como argumento en la línea de mandatos al arrancar el programa.

El programa mostrará cuántas palabras de 1 letra hay en el fichero, cuántas de 2 letras, etc.

Se considerará una palabra como una secuencia de caracteres que no contenga espacios en blanco.

Ejercicio 109 ++

Escribe una función que encuentre el máximo de un array de `Float` que se le pase como parámetro. Deberá podérsele pasar un array de `Float` de cualquier tamaño y devolverá el valor `Float` del máximo.

Haz un programa que pruebe la función con diferentes arrays. Este programa no recibe su entrada interactivamente ni de un fichero, sino que los arrays de prueba pueden estar declarados como variables del programa.

Ejercicio 110 +

Dadas las siguientes declaraciones:

```
type Matrix_Type is array (1..5, 1..4) of Float;
Matrix : Matrix_Type;
```

Responde a las siguientes preguntas:

1. ¿Cuántos elementos hay en `Matrix`?
2. Escribe una sentencia que muestre el elemento de la fila 3, columna 4 de `Matrix`
3. Escribe un bucle que calcule la suma de los elementos de la fila 5
4. Escribe un bucle que calcule la suma de los elementos de la columna 4
5. Escribe un bucle que calcule la suma de todos los elementos de `Matrix`
6. Escribe un bucle que muestre en pantalla los contenidos de `Matrix` rotados 90^{o}. La primera fila que se muestre en pantalla debe estar formada por los elementos de la columna 4 de `Matrix`, etc.

Ejercicio 111 +++

Escribe un subprograma que determine si un jugador ha ganado el juego de las 3 en raya. Se supondrá que el tablero es una matriz de 3x3, y que cada elemento puede tener uno de los siguientes valores: `Jugador_1, Jugador_2`.

El subprograma aceptará como parámetro al menos el tablero, y deberá devolver la información necesaria para saber si alguno de los dos jugadores ha ganado o si la partida sigue en marcha.

Ejercicio 112 +++

Utilizando el subprograma del Ejercicio 111 escribe un programa interactivo que permita a un jugador competir con tu programa para jugar a las 3 en raya. El programa deberá detectar quién ha ganado.

Ejercicio 113 +++

Escribe un paquete que exporte subprogramas para sumar, restar y multiplicar matrices. Los subprogramas deben validar los parámetros comprobando las dimensiones de las matrices.

Escribe un programa que utilice el paquete para testear su correcto funcionamiento.

Ejercicio 114 ++

Escribe un programa que lea interactivamente 5 cartas de póker y las almacene en un array de dos dimensiones (colores y figuras).

El programa deberá mostrar la jugada correspondiente a las 5 cartas. Nota: ver `http://www.poquer.com.es/ranking.html`

Ejercicio 115 ++

Escribe un programa que lea interactivamente 5 cartas de póker y almacene cada una en un registro con dos campos, color y figura. Las 5 cartas se almacenarán en un array de 5 registros.

El programa deberá mostrar la jugada correspondiente a las 5 cartas. Nota: ver `http://www.poquer.com.es/ranking.html`

Ejercicio 116 +++

Escribe una función recursiva que multiplique sus dos parámetros devolviendo el resultado.

Se puede realizar una multiplicación de manera recursiva apoyándose en la suma. Por ejemplo, se puede multiplicar recursivamente M por N así:

1. Multiplicar recursivamente M por N-1
2. Añadir M al resultado del paso anterior

Ejercicio 117 +++

Escribe un procedimiento que muestre en pantalla la sucesión de Fibonacci hasta su término N, siendo N el parámetro de la función.

La sucesión de Fibonacci se define recursivamente así:

- $Fib_1 = 1$
- $Fib_2 = 1$
- $Fib_n = Fib_{n-2} + Fib_{n-1}$, para n > 2

Escribe ahora el mismo procedimiento de manera iterativa.

Ejercicio 118 +++

El máximo común divisor (MCD) de dos enteros positivos es el mayor número entero que es un divisor de ambos.

El algoritmo de Euclides para encontrar el MCD se define recursivamente:

- $MCD(M, N)$ es N si $N <= M$ y N es un divisor de M
- $MCD(M, N)$ es $MCD(N, M)$ si $M < N$
- $MCD(M, N)$ es $MCD(N$, resto de dividir M entre $N)$ en el resto de casos

Escribe una función que devuelva el `MCD` de sus dos parámetros, M y N.

Ejercicio 119 +++

Escribe una función `Power` que reciba dos argumentos, `Base` y `Exponente` y que calcule $Base^{exponente}$ de manera recursiva.

Nota:

- $M^0 = 1$
- $M^N = M * M^{N-1}, para N > 0$

Ejercicio 120 +

¿Cuál es la salida del siguiente programa? ¿Qué calcula la función `Strange`?

```
with Ada.Text_IO;

procedure Test_Strange is
   function Strange (N : Integer) return Integer is
   begin
     if N = 1 then
        Result := 0;
     else
        Result := 1 + Strange (N / 2);
     end if;
   end Strange;

begin
   Ada.Text_IO.Put(Integer'Image(Strange (8)));
   Ada.Text_IO.New_Line;
end Test_Strange;
```

Ejercicio 121 ++

Escribe una función recursiva, `Find_Sum` que calcule la suma de los enteros que comienzan en 1 y acaban en N.

Ejercicio 122 ++

Escribe un procedimiento que muestre el contenido del array que se le pasa como argumento. Deberá implementarse el procedimiento de manera recursiva, no pudiéndose recorrer con un bucle.

Ejercicio 123 +++

Escribe una función `Reverse` que reciba como argumento un `String` y devuelva el mismo `String` dado la vuelta. Ejemplo: si se le pasa como argumento `"hola a todos"` devolverá el `String` `"sodot a aloh"`. Esta función deberá implementarse de manera recursiva, no pudiéndose recorrer el string con un bucle.

Escribe un programa que haciendo uso de la función `Reverse` compruebe si un texto introducido por el usuario es un palíndromo. Un palíndromo es una cadena de texto que se lee igual del derecho que del revés. Por ejemplo la palabra `"reconocer"` es un palíndromo.

Ejercicio 124 +++

Dadas las siguientes declaraciones:

```
subtype Name_Type is String (1..10);
subtype GPA_Type is Float range 0.0 .. 4.0;
type Student_Rec is
   record
      First_Name    : Name_Type;
      Last_Name     : Name_Type;
      ID            : Positive;
      GPA           : GPA_Type;
      Current_Hours : Natural;
      Total_Hours   : Natural;
   end record;

type List_Type is array (1 .. 100) of Student_Rec;
type List_Ptr is access List_Type;
Student_List : List_Ptr;
```

Se supondrá que las variables de tipo `Access` ocupan 6 bytes en memoria, las de tipo `Integer` 4 bytes, las de tipo `Float` 4 bytes, las de tipo `Character` 1 byte. Además se supondrá que los campos de un registro están almacenados contiguamente en memoria, sin dejar huecos entre ellos.

1. ¿Cuánta memoria medida en número de bytes ocupa la variable `Student_List`?
2. ¿Cuánta memoria medida en número de bytes se obtiene al ejecutarse la siguiente sentencia?:
   `Student_List := new List_Type;`
3. Escribe una sentencia de asignación que asigne 1000 al campo `ID` del primer estudiante del array obtenido en el apartado anterior.
4. Escribe un bucle que muestre el valor del campo `ID` de todos los estudiantes de la lista.
5. Escribe un bucle en el que para cada estudiante se sume su campo `Current_Hours` al valor de su campo `Total_Hours`, y se inicialice a cero `Current_Hours`

Ejercicio 125 ++

Dadas las siguientes declaraciones:

```
type Frequency_Array is array (Character range <>) of Natural;

type Frequency_Ptr is access Frequency_Array;

A : Frequency_Ptr;
B : Frequency_Ptr;
C : Frequency_Ptr;
```

1. Escribe una sentencia que cree espacio en memoria dinámica para un array de tipo `Frequency_Array` con un índice de rango `'a'..'g'` y almacene su dirección en la variable A.
2. Escribe una sentencia que cree espacio en memoria dinámica para un array de tipo `Frequency_Array` con un índice de rango `'h'..'n'` y almacene su dirección en la variable B. Cada elemento del array debería tener como valor inicial cero.
3. Escribe un bucle que asigne el valor 100 a todos los elementos del array apuntado por `B`.
4. Escribe una sentencia que cree espacio en memoria dinámica para un array de tipo `Frequency_Array` con un índice de rango `'a'..'n'` y almacene su dirección en la variable `C`. Cada elemento del array debería tener como valor inicial el que tenga la posición correspondiente en los arrays apuntados por `A` y `B`.
5. Escribe el código necesario para instanciar un procedimiento llamado `Free` que sirva para liberar la memoria de un objeto apuntado por una variable de tipo `Frequency_Ptr`.
6. Usa el procedimiento `Free` del apartado anterior para liberar la memoria que ocupa el array apuntado por la variable `C`.
7. Suponiendo que `A`, `B` y `C` apuntan a tres arrays distintos, describe qué ocurriría cuando se ejecutase cada uno de los siguientes fragmentos de código. Realiza gráficos que representen la memoria ocupada por las variable `A, B, C` y por los arrays apuntados por ellas, antes y después de que se ejecute cada sentencia.

   a)
   ```
   Free(A);
   Free(A);
   ```

   b)
   ```
   Free(A);
   A.all('c') := 52;
   ```

   c)
   ```
   A := null;
   ```

   d)
   ```
   A := null;
   A.all('c') := 52;
   ```

   e)
   ```
   A := B;
   B.all('f') := 96;
   ```

   f)
   ```
   A := B;
   Free(B);
   A := null;
   ```

   g)
   ```
   A.all := C.all('a'..'g');
   ```

   h)
   ```
   A.all := C.all;
   ```

   i)
   ```
   A := null;
   B := A;
   ```

Ejercicio 126 +++

Dadas las siguientes declaraciones:

```
type My_Unbounded_String_Type is access String;

function Length (Source : in My_Unbounded_String_Type) return Natural is
begin
   return Source.all'Length;
end Length

function To_String (Source : in My_Unbounded_String_Type) return String is
begin
   return Source.all;
end To_String;

function "<" (Left  : in My_Unbounded_String_Type;
              Right : in My_Unbounded_String_Type) return Boolean is
begin
   return Left.all < Right.all;
end "<";

function "&" (Left  : in My_Unbounded_String_Type;
              Right : in String)  return My_Unbounded_String_Type is
   Result : My_Unbounded_String_Type;
begin
   Result := new String'(Left.all & Right);
   return Result;
end "&";

procedure Free is new
  Ada.Unchecked_Deallocation (Object => String,
                              Name   => My_Unbounded_String_Type);

W : My_Unbounded_String_Type := new String (1..20);
X : My_Unbounded_String_Type := new String'("Hola");
Y : My_Unbounded_String_Type := new String'(1..10 => 'J');
Z : My_Unbounded_String_Type := new String'(X.all & " " & "Pepe");
```

1. Haz un gráfico en el que se muestre la memoria ocupada por las variables `W, X, Y, Z` y por las zonas de memoria apuntadas por ellas.

2. ¿Qué ocurriría si se ejecutase la siguiente sentencia?: `W := X;`. Realiza un gráfico que ilustre el estado de la memoria antes y después de ejecutar la sentencia.

3. ¿Qué ocurriría si se ejecutasen las siguientes sentencias?:

   ```
   Free (W);
   W := new String'(X.all);
   ```

   Realiza un gráfico que ilustre el estado de la memoria antes y después de ejecutar cada una de las sentencias.

4. ¿Qué ocurriría si se ejecutase la siguiente sentencia?:

   ```
   X := X & " Jaime";
   ```

   Realiza un gráfico que ilustre el estado de la memoria antes y después de ejecutar cada una de las sentencias.

5. ¿Qué ocurriría si se ejecutasen las siguientes sentencias?:

   ```
   Free (W);
   W := X;
   X := X & " Jaime";
   Free (W);
   ```

   Realiza un gráfico que ilustre el estado de la memoria antes y después de ejecutar cada una de las sentencias.

Ejercicio 127 +++

Dada la siguiente declaración:

```
type My_Unbounded_String_Type is access String;
```

Escribe el cuerpo de la siguiente función:

```
-- Si Pattern es una subcadena de Source, Index devuelve la posición que ocupa
--   el primer carácter de Pattern en Source.
-- Si Pattern no es una subcadena Index devuelve 0
function Index (Source  : in My_Unbounded_String_Type;
                Pattern : in My_Unbounded_String_Type) return Natural;
```

Ejercicio 128 +++

Sea la siguiente declaración de una lista enlazada:

```
with Ada.Strings.Unbounded;
...
    package ASU renames Ada.Strings.Unbounded;

    type Cell;
    type Cell_A is access Cell;

    type Cell is record
       Name  : ASU.Unbounded_String := ASU.Null_Unbounded_String;
       Value : Integer := 0;
       Next  : Cell_A;
    end record;

    P_First : Cell_A;
    P_Elem : Cell_A;
```

Supón que en un instante dado el contenido de la lista es el que se muestra en la figura 1.

Escribe el código que permita contar cuántos elementos tiene la lista.

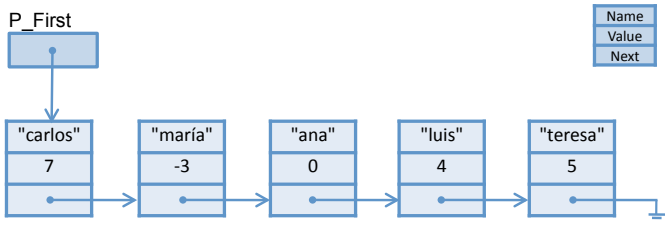


Figura 1: Lista enlazada 1

Ejercicio 129 +++

Sea la siguiente declaración de una lista enlazada:

```
with Ada.Strings.Unbounded;
...
    package ASU renames Ada.Strings.Unbounded;

    type Cell;
    type Cell_A is access Cell;

    type Cell is record
       Name  : ASU.Unbounded_String := ASU.Null_Unbounded_String;
       Value : Integer := 0;
       Next  : Cell_A;
    end record;

    P_First : Cell_A;
    P_Elem : Cell_A;
```

Supón que en un instante dado el contenido de la lista es el que se muestra en la figura 2, y que el puntero `P_Elem` apunta al tercer elemento de la misma.

Escribe el código que eliminaría de la lista ese elemento apuntado por `P_Elem`.

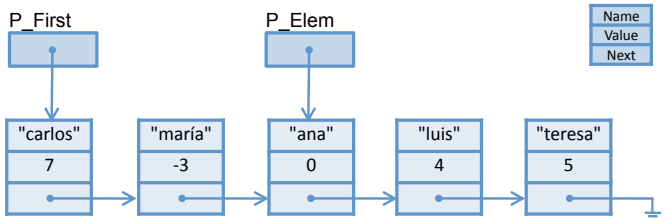


Figura 2: Lista enlazada 2

Ejercicio 130 +++

El problema $3n + 1$ (Problema utilizado en el concurso de Programación San Teleco 2012).

El siguiente algoritmo genera una secuencia de números:

> Se comienza con un número entero n
>
> Repetir hasta que n sea 1:
>
> - Si n es par, se divide entre 2
> - Si n es impar, se multiplica por 3 y se le suma 1

Por ejemplo, la siguiente secuencia se genera cuando el número inicial $n = 22$:

```
22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

Para un número inicial n la *longitud de ciclo* de n es la cantidad de números generados, incluyendo el 1. En el ejemplo anterior, la *longitud de ciclo* para $n = 22$ es 16.

Dados dos números i y j, tienes que determinar la máxima *longitud de ciclo* para todos los números entre i y j, incluyendo a ambos números.

Entrada

La entrada consiste en una serie de parejas de enteros, i y j, una pareja de números por línea. Todos los enteros serán menores que 1,000,000 y mayores que 0.

Salida

Para cada pareja de enteros de la entrada, i y j, la salida será i y j, en el mismo orden en el que aparecieron en la entrada, y después, en la misma línea, la máxima *longitud de ciclo* para los enteros entre i y j, incluyendo ambos valores. Estos tres números deben separarse por un espacio, con los tres números en la misma línea, y una línea por cada línea de la entrada.

Ejemplos

Ejemplo de entrada

```
1 10
100 200
201 210
900 1000
```

Ejemplo de salida

```
1 10 20
100 200 125
210 210 89
900 1000 174
```

Ejercicio 131 +++

Buscaminas (Problema utilizado en el concurso de Programación San Teleco 2012).

El objetivo del juego del buscaminas es encontrar dónde están las minas en un campo de minas de MxN posiciones.

El juego muestra el campo de minas con un número en cada posición que indica cuántas minas hay adyacentes a dich posición. Cada posición tiene un máximo de ocho posiciones adyacentes. El campo de $4x4$ posiciones que se muestra a continuación contiene dos minas, cada una representada por el carácter `"*"`. Si se representa el mismo campo de minas con los números descritos antes, el campo resultante sería el de la derecha:

```
*...        *100
....        2210
.*..        1*10
....        1110
```

Entrada

La entrada consta de un número arbitrario de campos de minas. La primera línea de cada campo de minas contiene dos enteros n y m $(0 < n, m \leq 100)$ que son el número de filas y columnas del campo de minas respectivamente. Cada una de las siguientes n líneas contiene exactamente m caracteres, representando el campo de minas.

Las posiciones seguras se representan con el carácter `"."` y las posiciones en las que hay minas con el carácter `"*"`, ambos sin las comillas. Una primera línea de un campo de minas en la que $n = m = 0$ indica el final de la entrada, no debiéndose procesar esta línea.

Salida

Para cada campo de minas se mostrará el mensaje `Campo #x:` en una línea, en la que `x` es el número de campo de minas, empezando por 1. Las siguientes n líneas deben contener el campo de minas con los caracteres `"."` substituidos por el número de minas adyacentes a esa posición. Debe haber una línea en blanco sseparando la salida de los campos de minas.

Ejemplos

Ejemplo de entrada	Ejemplo de salida
4 4 *... .... .*.. .... 3 5 **... ..... .*... 0 0	Campo #1: *100 2210 1*10 1110 Campo #2: **100 33200 1*100

Ejercicio 132 +++

Pantalla 7 segmentos (Problema utilizado en el concurso de Programación San Teleco 2012).

Escribe un programa que muestre dígitos representados con segmentos. Cada dígito se representará con segmentos verticales y horizontales. El número 8 por ejemplo se representa con los siguientes 7 segmentos (en este caso cada segmento se dibuja con un único carácter):

```
 -
| |
 -
| |
 -
```

El resto de dígitos se representa con un subconjunto de estos 7 segmentos utilizados para representar el dígito 8.

Entrada

El fichero de entrada contiene varias líneas, una para cada número que se quiere mostrar. Cada línea contiene dos enteros s y n, siendo n el número a mostrar ($0 \leq n \leq 99{,}999{,}999$) y s el tamaño de cada segmento que se debe utilizar para representar los números ($1 \leq s \leq 10$). La entrada terminará con una línea que contiene dos ceros, que no deben ser procesados.

Salida

Se muestran los números especificados en el fichero de entrada representados con s caracteres `"-"` para los segmentos horizontales y s caracteres `"|"` para los segmentos verticales. Cada dígito ocupa exactamente $s+2$ columnas y $2s+3$ filas. Asegúrate de llenar con espacios el espacio en blanco de cada dígito, incluyendo el último dígito. Debe haber exactamente una columna de espacios en blanco entre cada dos dígitos.

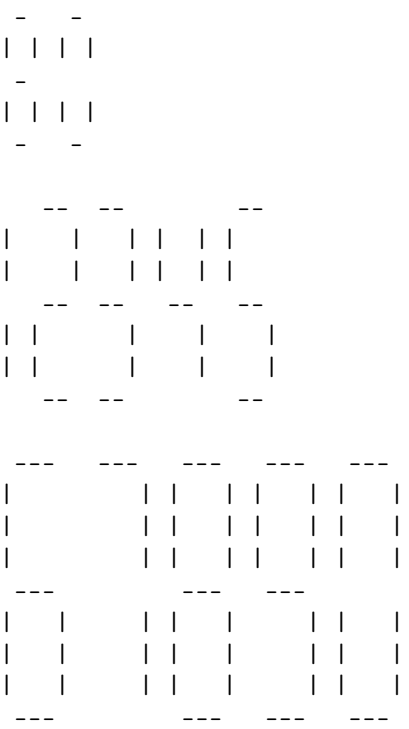
Deja una línea en blanco detrás de cada dígito.

Ejemplos

Ejemplo de entrada

```
1 80
2 12345
3 67890
0 0
```

Ejemplo de salida



Ejercicio 133 +++

Jaque (Problema utilizado en el concurso de Programación San Teleco 2012).

Escribe un programa que lea la configuración de un tablero de ajedrez e identifique si la casilla de alguno de los reyes está amenazada (si les están dando jaque).

Se está dando jaque a un rey cuando la casilla en la que está puede ser ocupada por el oponente en su próximo movimiento.

Las piezas blancas se representarán con letras mayúsculas, y las negras con letras minúsculas. El lado de las piezas blancas será siempre la parte inferior del tablero según aparece en pantalla.

Para los que no conozcan las reglas del ajedrez, estos son los movimientos de cada pieza:

Peón (p o P): sólo puede moverse hacia adelante una casilla en cada movimiento. Sin embargo, puede come piezas del contrario que estén a una casilla de distancia en las diagonales, avanzando hacia el lado del contrario, y eso es lo que importa en este problema.

Caballo (c o C): se mueve en L tal como se muestra más abajo. Es la única pieza que puede saltar por encima de otras piezas.

Alfil (a o A): puede moverse cualquier número de casillas diagonalmente, hacia adelante o hacia atrás.

Torre (t o T): puede moverse cualquier número de casillas verticalmente u horizontalmente, hacia adelante o hacia atrás.

Dama (d o D): puede moverse cualquier número de casillas en cualquier direción (diagonal, horizontal o verticalmente), hacia adelante o hacia atrás.

Rey (r o R): sólo puede moverse una casilla en cada movimiento, en cualquier dirección (diagonal, horizontal o verticalmente), hacia adelante o hacia atrás.

Debajo se muestran ejemplos de cómo se pueden mover las piezas, donde `"*"` indica las posiciones a en las que una pieza puede capturar a otra:

```
Peón         Torre        Alfil        Dama         Rey          Caballo
........     ...*....     .......*     ...*...*     ........     ........
........     ...*....     *.....*.     *..*..*.     ........     ........
........     ...*....     .*...*..     .*.*.*..     ........     ..*.*...
........     ...*....     ..*.*...     ..***...     ..***...     .*...*..
...p....     ***t****     ...a....     ***d****     ..*r*...     ...c....
..*.*...     ...*....     ..*.*...     ..***...     ..***...     .*...*..
........     ...*....     .*...*..     .*.*.*..     ........     ..*.*...
........     ...*....     *.....*.     *..*..*.     ........     ........
```

Recuerda que el caballo es la única pieza que puede saltar por encima de otras piezas. El movimiento del peón depende del lado en el que esté. Si es un peón blanco, sólo puede moverse una casilla en diagonal hacia capturar a otra pieza. Si es un peón blanco, sólo puede moverse una casilla en diagonal hacia arriba para capturar otra pieza. El ejemplo de peón que se muestra más arriba corresponde a un peón negro (letra `"p"` minúscula). En este desafío utilizamos la palabra *movimiento* para referirnos a las casillas a las que puede moverse una pieza para capturar otra.

Entrada

Habrá un número de configuraciones de tableros en la entrada, constando cada uno de ellos de ocho líneas de ocho caracteres cada una. Un `"."` representa una casilla vacía, siendo las letras minúsculas y mayúsculas las piezas tal como se definieron antes. No pueden aparecer otros caracteres. No puede haber configuraciones en las que se esté

dando jaque a ambos reyes. Debe leerse la entrada hasta encontrar un tablero vacío (compuesto únicamente por caracteres "."), que no debe ser procesado. Habrá una línea en blanco entre cada par de configuraciones de tableros. Todos los tableros, excepto el vacío, contendrán exactamente un rey blanco y uno negro.

Salida

Para cada configuración de tablero que se lea debe mostrarse en la salida una de las siguientes salidas:

```
Juego #d: jaque al rey blanco
Juego #d: jaque al rey negro
Juego #d: no se da jaque a ningún rey
```

donde `d` es el número de juego, comenzando por 1.

Ejemplos

Ejemplo de entrada

```
..r.....
ppp.pppp
........
.T...A..
........
........
PPPPPPPP
R.......

tcadr.ct


ppp..ppp
....p...
...p....
.aPP....
.....C..
PP..PPPP
TCADRA.T

........
........
........
........
........
........
........
........
```

Ejemplo de salida

```
Juego #1: jaque al rey negro
Juego #2: jaque al rey blanco
```

Ejercicio 134 +++

Democracia australiana (Problema utilizado en el concurso de Programación San Teleco 2012).

¿Sabías que en 1856 se votó por primera vez en el mundo de manera secreta en los estados de Victoria y Tasmania de Australia?

En este país los votantes tienen que ordenar a todos los candidatos en orden de preferencia. En un primer recuento de las papeletas, si uno de los candidatos aparece como preferido en más del 50 % de los votos, éste resulta elegido.

Si por el contrario ningún candidato aparece como preferido en más del 50 % de los votos:

1. Se elimina el candidato que haya recibido el menor número de votos. Si hay varios candidatos empatados en el menor número de votos, se eliminan todos ellos.
2. A continuación se rehace el recuento: para cada papeleta, se le suma el voto de cada candidato eliminado al candidato preferido no eliminado de esa papeleta.

Este proceso de eliminación de los candidatos menos votados y de recuento de sus votos en favor del candidato preferido continúa hasta que un candidato recibe más del 50 % de los votos, o hasta que todos los candidatos aún no eliminados están empatados.

Entrada

La entrada comienza con una línea en la que hay un entero positivo que indica el número de procesos electorales distintos que se van a procesar. Después de esta primera línea viene una línea en blanco. También hay una línea en blanco entre cada dos procesos electorales distintos.

La primera línea de cada proceso electoral es un entero $n \leq 20$ que indica el número de candidatos. Las siguientes n líneas contienen los nombres de los n candidatos, cada uno de los cuales puede tener un máximo de 80 caracteres. En las siguientes líneas, hasta un máximo de 1000, aparecen las papeletas. Cada papeleta contiene los números de los candidatos, de 1 a n, ordenados según la preferencia de cada votante: el primer número es el candidato preferido, el segundo número es su segunda opción, etc.

Salida

La salida muestra el/los candidatos elegidos para cada una de las votaciones. Después de cada proceso electoral se debe generar una línea en blanco.

Los resultados de cada votación constan de una línea con el nombre del candidato elegido, o de varias líneas con los nombres de los candidatos que han empatado.

Ejemplos

Ejemplo de entrada

```
1

3
Groucho Marx
Charles Chaplin
Gila
1 2 3
2 1 3
2 3 1
1 2 3
3 1 2
```

Ejemplo de salida

```
Groucho Marx
```

Ejercicio 135 +++

Descifrado con texto en claro conocido (Problema utilizado en el concurso de Programación San Teleco 2012).

Un método sencillo e inseguro de cifrar texto es permutar las letras del alfabeto: cada letra del alfabeto se substituye siempre que aparece en el texto por otra letra del alfabeto. Para que el proceso sea reversible, cada letra ha de ser substituida por una letra distinta.

Si se posee un texto en claro no cifrado, y su correspondiente texto cifrado, se puede averiguar qué substituciones se hicieron para cifrar los mensajes. A este ataque se le denomina descifrado con texto en claro conocido.

Tu tarea consiste en descifrar varias líneas de texto cifradas, suponiendo que cada una de ellas utiliza el mismo conjunto de substituciones, y que una de las líneas es el mensaje cifrado del siguiente texto en claro conocido: `the quick brown fox jumps over the lazy dog`

Entrada

La entrada comienza con una línea en la que únicamente hay un entero positivo que indica el número de casos a tratar, seguido de una línea en blanco. También hay una línea en blanco entre cada dos casos consecutivos.

Salida

Para cada caso hay que descifrar cada línea e imprimirla en la salida. Si no se puede descifrar una frase se mostrará la siguiente salida:

`No hay solución`

Ejemplos

Ejemplo de entrada

```
1

vtz ud xnm xugm itr pyy jttk gmv xt otgm xt xnm puk ti xnm fprxq
xnm ceuob lrtzv ita hegfd tsmr xnm ypwq ktj
frtjrpgguvj otvxmdxd prm iev prmvx xnmq
```

Ejemplo de salida

```
now is the time for all good men to come to the aid of the party
the quick brown fox jumps over the lazy dog
programming contests are fun arent they
```

Ejercicio 136 +++

Descifrado con diccionario (Problema utilizado en el concurso de Programación San Teleco 2012).

Un método sencillo e inseguro de cifrar texto es permutar las letras del alfabeto: cada letra del alfabeto se substituye siempre que aparece en el texto por otra letra del alfabeto. Para que el proceso sea reversible, cada letra ha de ser substituida por una letra distinta.

Tu tarea es descifrar varias líneas de texto cifradas, suponiendo que cada línea usa un conjunto de substituciones distinto, y que todas las palabras del texto descifrado pertenecen a un diccionario de palabras conocidas.

Entrada

La entrada consta de una línea con un número entero n, seguido de n palabras escritas con letras minúsculas, una por línea, en orden alfabético. Estas n palabras componen el diccionario de palabras que pueden aparecer en el texto descifrado. Después del diccionario apareen varias líneas de mensajes cifrados según lo explicado más arriba.

No puede haber más de 1000 palabras en el diccionario. Las palabras tienen 16 caracteres como máximo. Las líneas con los mensajes cifrados contienen sólo letras minúsculas y espacios, y tienen una longitud máxima de 80 caracteres.

Salida

En la salida aparecen los mensajes descifrados, uno por línea. Si hay varias soluciones se puede imprimir una cualquiera. Si no hay solución se imprime una línea en la que cada carácter se substituye por un asterisco.

Ejemplo de entrada

```
6
llora
jaime
la
vota
juan
pelota
kbjnf mmpsb kvbñ wpub mb qfmpub mmpsb kvbñ qfmpub
xxxxx yyyyy aaaa eeee zz pppppp cassn pppe pixxxx
```

Ejemplo de salida

```
jaime llora juan vota la pelota llora juan pelota
***** ***** **** **** ** ****** ***** **** ******
```

Ejercicio 137 +++

Piedra, papel, o tijera (Problema utilizado en el concurso de Programación San Teleco 2012).

Piedra, papel o tijera es un juego infantil conocido también como cachipún, jankenpón, yan ken po, chis bun papas, hakembó, chin-chan-pu o kokepon. Es un juego de manos en el cual existen tres elementos:

- La piedra, que vence a la tijera rompiéndola
- La tijera, que vence al papel cortándolo
- El papel, que vence a la piedra envolviéndola

Este juego es muy utilizado para decidir quien de dos personas hará algo, tal y como a veces se hace usando una moneda, o para dirimir algún asunto. `http://es.wikipedia.org/wiki/Piedra,_papel_o_tijera`

Escribe un programa que determine los ganadores de una serie de juegos de piedra, papel o tijera. En cada juego pueden participar dos jugadores.

Entrada

En la entrada habrá una serie de juegos, separados por una línea en blanco. Cada juego consta de dos líneas, en cada una de las cuáles figura el nombre propio del jugador y su elección: `"R"` para piedRa, `"P"` para Papel y `"T"` para tiJera. El nombre es una sóla palabra.

Salida

En la salida aparecerá una línea con el nombre del ganador de cada juego.

Ejemplos

Ejemplo de entrada

```
Jaime P
Carmen T

Juan T
Carlos R
```

Ejemplo de salida

```
Carmen
Carlos
```

Ejercicio 138 +++

Anagramas (Problema utilizado en el concurso de Programación San Teleco 2012).

Un anagrama es una palabra formada por la trasposición de las letras de otra palabra. Por ejemplo `"carpa"` es un anagrama de `"parca"`.

Escribe un programa que dado un conjunto de palabras las agrupe por anagramas.

Entrada

Cada conjunto de palabras a agrupar aparecerá en una serie de líneas, una palabra por línea. Para separar conjuntos diferentes de palabras a clasificar se utilizará una línea en blanco.

Salida

Cada grupo de anagramas distinto aparecerá en una línea. Para separar los grupos de diferentes conjuntos se usará una línea en blanco.

Ejemplos

Ejemplo de entrada

```
amor
roldan
roma
ladron
omar
mora
ramo
armo

monja
lamina
esponja
jamon
animal
japones
mojan
```

Ejemplo de salida

```
amor roma omar mora ramo armo
roldan ladron

monja jamon mojan
lamina animal
esponja japones
```