

Programación en Pascal. Selección

Miguel Ortuño

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

Septiembre de 2022



© 2022 Miguel Angel Ortuño Pérez.
Algunos derechos reservados. Este documento se distribuye bajo la
licencia *Atribución-CompartirIgual 4.0 Internacional* de Creative
Commons, disponible en
<https://creativecommons.org/licenses/by-sa/4.0/deed.es>

1 Selección

- Problemas de selección
- if then else
- case
- Estructura de un programa
- Precondiciones y postcondiciones

Problemas de selección

Hasta ahora hemos visto problemas de solución directa:

- En el tema 2, la solución al problema era una expresión, numérica o booleana, construida como una secuencia operandos y operadores, donde los operandos eran o constantes o funciones predefinidas
- En el tema 3, aprendimos a usar funciones, pero la solución seguía siendo una única expresión, con la novedad de que los operadores podían ser no solo constantes y funciones predefinidas, sino funciones definidas por nosotros
- En este tema, veremos las *sentencias de control*, que permiten como seleccionar una expresión u otra, o ejecutar una acción u otra, a partir de cierta condición booleana.
En Pascal se hace mediante las setencias *if-then-else* y *case*

if then else

if *condición* then

sentencia/s a ejecutar si la condición es cierta

else

sentencia/s a ejecutar si la condición es falsa

- La condición puede ser cualquier expresión: una combinación de constantes, funciones, variables...
- En la rama *then* o en la rama *else* puede haber
 - Una única sentencias
 - Una lista de sentencias, dentro de un bloque *begin end*
- La sentencia *if then else* , como cualquier otra, se puede escribir escribir tanto en el cuerpo del programa principal como en el cuerpo de un subprograma

```
if mi_expression then
  writeln('Bla Bla') // Correcto
else
  writeln('Bla Bla Blá');
```

MUY IMPORTANTE

Es una única sentencia, en Pascal nunca se pone ';' antes del else. Este es un error de programación muy común (Pascal tiene pocas rarezas pero tal vez esta sea una)

```
if mi_expression then
  writeln('Bla Bla'); // ¡¡MAL!! (sobra el punto y coma)
else
  writeln('Bla Bla Blá');
```

Si en la *rama then* o en la *rama else*, o en ambas, es necesario ejecutar más de una sentencia, se declara un bloque *begin end*

```
if mi_expresion then begin
    write('Bla ');
    writeln('Bla Bla');
end    // Atención, antes del else, nunca se escribe punto y coma
else begin
    write('Bla Bla ');
    writeln('Blá');
end;
```

- Observa que nuestro convenio es que las sentencias tengan exactamente un nivel más de sangrado que *if*, *end* y *else*

En Pascal el punto y coma no se usa para finalizar sentencias, sino para delimitarlas. Por tanto, la última sentencia no necesita punto y coma. Aunque es más sencillo ponerla siempre, como haremos en esta asignatura. En este caso el compilador entiende que al final del bloque hay una sentencia nula

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
  
program if_then_else;  
  
const  
    Limite_fiebre = 37.5;  
    Temperatura = 37.8;  
  
begin  
    if Temperatura >= Limite_fiebre then  
        writeln('Fiebre')  
    else  
        writeln('Temperatura normal');  
end.
```

- En este curso, nuestro convenio será escribir la palabra reservada *if*, la condición y la palabra reservada *then* en una línea
- De esta forma, las sentencias a ejecutar tendrán exactamente 1 nivel de tabulación más que el *if* y el *else*

Una vez más: recuerda que el carácter ';' separa sentencias.

if-then-else es una única sentencia, no puede haber un ';' en medio¹

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program if_then_else_MAL;

const
    Limite_fiebre = 37.5;

var
    temperatura: real = 37.8;

begin
    if temperatura >= Limite_fiebre then
        writeln('Fiebre') ; // ¡¡MAL!! Sobra el ;
    else
        writeln('Temperatura normal');
    end.
```

¹Excepto dentro de un bloque begin-end, naturalmente

Es necesario que la tabulación sea consistente, pero también hay otros criterios posibles y habituales, p.e escribir *if*, *then* y *else* en la misma columna

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program if_then_else_v02;  
  
// Otro convenio posible, aunque no lo seguiremos aquí  
  
const  
    Limite_fiebre = 37.5;  
    Temperatura = 37.8;  
  
begin  
    if Temperatura >= Limite_fiebre  
    then  
        writeln('Fiebre')  
    else  
        writeln('Temperatura normal');  
end.
```

Otra posibilidad (discrepante con el convenio de este curso)
escribir bloques begin-end siempre, aunque no haga falta

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program if_then_else_v03;  
  
// Otra posibilidad más: escribir siempre begin-end  
  
const  
    Limite_fiebre = 37.5;  
    Temperatura = 37.8;  
  
begin  
    if Temperatura >= Limite_fiebre then begin  
        writeln('Fiebre');  
    end  
    else begin  
        writeln('Temperatura normal');  
    end; // El punto y coma tras el end puede omitirse  
end.
```

Si la rama del *then* o la rama del *else* tienen más de una sentencia, entonces sí será necesario el uso de *begin end*

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program if_then_else_v04;  
  
const  
    Limite_fiebre = 37.5;  
    Temperatura = 37.8;  
  
begin  
    if Temperatura >= Limite_fiebre then begin  
        writeln('Fiebre');  
        write('La temperatura es ');  
        write(Temperatura-Limite_fiebre:0:1);  
        writeln(' grados superior a lo normal');  
    end  
    else  
        writeln('Temperatura normal');  
end.
```

Ejecución:

Fiebre

La temperatura es 0.3 grados superior a lo normal

- Nuestro convenio es que si escribimos `begin`, irá en la misma línea de *if condición then*
- Observa que las sentencias de la rama *then* aparecen con una tabulación adicional, igual que las de la rama *else*

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program if_then_else_v05;
    // Tabulación discutible
const
    Limite_fiebre = 37.5;
    Temperatura = 37.8;
begin
    if Temperatura >= Limite_fiebre
    then
        begin
            writeln('Fiebre');
            write('La temperatura es ');
            write(Temperatura-Limite_fiebre:0:1);
            writeln(' grados superior a lo normal');
        end
    else
        writeln('Temperatura normal');
    end.
end.
```

Este enfoque tiene el inconveniente de añadir un nivel de tabulación innecesario, es preferible tabular como en if_then_else_v04

Números mágicos

- En un programa, una constante numérica literal (un número *tal cual*) solo puede estar en la definición de una constante, en ningún otro sitio
- Poner una constante numérica literal en mitad del código es una mala práctica denominada *número mágico*. El compilador no lo impedirá, pero es un código defectuosos, potencialmente problemático

Ejemplo incorrecto:

```
if Temperatura >= 37.5 then  
    writeln('Fiebre');
```

Ejemplo correcto:

```
const  
    Limite_fiebre = 37.5;  
begin  
    if Temperatura >= Limite_fiebre then  
        writeln('Fiebre');  
end
```


Pero hay excepciones. Si a juicio del desarrollador hay algún número cuyo valor es muy obvio, puede ser conveniente escribir un número *tal cual*, sin definir una constante

```
if n >= 0 then
  result := 'positivo'
else
  result := 'negativo';
```

Problema: omisión del begin-end

Es muy importante usar begin-end cuando una de las ramas tiene más de una sentencia

- Si me olvido del begin-end en la rama *then*, no es un error demasiado serio porque resulta un error de sintaxis, el *else* queda descolocado y el compilador me avisará con un error Fatal: Syntax error, ";" expected but "ELSE" found

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program error_olvido_begin_end;
function comprueba_positivo(x: integer): boolean;
begin
    if x >= 0 then    // ¡¡Mal!!
        writeln('Entra en rama then');
        result := True ; // Olvidé el begin-end
                        // pero el compilador me avisa
    else
        result := False;
end;

begin
    writeln(comprueba_positivo(3));
end.
```

Recuerda que las funciones no deben tener efectos laterales. Pero en este caso el `writeln` puede ser aceptable, para una prueba, que borraremos

Pero si olvido el begin-end en la rama else, el error es más severo, porque la sintaxis es correcta

- Resultará un error lógico
- El compilador considerará que la primera sentencia pertenece a la rama else, pero que ahí acaba la rama
- El resto de sentencias aparentan ser correctas, se ejecutarán siempre, sin importar la condición del *if*
- La tabulación, que resultará incorrecta, hace que el error pueda ser más difícil de localizar

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program error_olvido_begin_end;  
  
function comprueba_positivo(x:integer):boolean;  
begin  
    if x >= 0 then  
        result := True  
    else  
        writeln('Entra en rama else, número negativo');  
        result := False; // ¡¡Mal!! Se ejecuta siempre, está  
                        // mal tabulado, olvidé el begin-end  
    end;  
  
begin  
    writeln(comprueba_positivo(3)); // Escribe FALSE  
end.
```

Omisión del else

El else no es obligatorio, si nuestro algoritmo no ejecuta nada en caso de incumplimiento de la condición, lo omitimos

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program if_then;  
  
const  
    Limite_fiebre = 37.5;  
    Temperatura = 37.8;  
  
begin  
    if Temperatura >= Limite_fiebre then  
        writeln('Fiebre');  
end.
```

If anidado

En la rama *then* puede haber otra sentencia *if-then-else*

Este ejemplo es correcto, aunque peligroso

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program anidacion_correcta;  
  
const  
    Mayoria_edad = 18;  
    Adolescencia = 13;  
    Edad = 14;  
  
begin  
    if Edad < Mayoria_edad then  
        if Edad < adolescencia then  
            writeln(Edad, ': Niño')  
        else  
            writeln(Edad, ': Adolescente');  
    end.  
end.
```

El *else* se corresponde con el *then* más próximo
Ejecución:

14: Adolescente

El problema es que no queda del todo claro, si no conocemos bien el criterio de Pascal, o si nos descuidamos, podemos pensar, erróneamente que el *else* se corresponde al primer *if*

El error anterior es aún más severo (más probable) si el programa está mal tabulado:

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }

program MALA_TABULACION;

const
    Mayoria_edad = 18;
    Adolescencia = 13;
    Edad = 14;

begin
    if Edad < Mayoria_edad then
        if Edad < Adolescencia then
            writeln(Edad, ': Niño')
        else // ;;Mal tabulado!!
            writeln(Edad, ': Adolescente');
end.
```

- En el ejemplo erróneo anterior, la tabulación nos haría pensar que se escribirá *adolescente* cuando $edad \geq \text{Mayoria_edad}$
- Como hemos dicho, lo que realmente hace el programa es escribir *adolescente* cuando $edad < \text{Mayoria_edad}$ y $edad \geq \text{adolescencia}$

- Para evitar este problema, una práctica recomendable que aquí estableceremos como convenio es que cuando en la rama *then* haya otra sentencia *if-then-else*, la *protegeremos* con un bloque *begin-end*

```
if condicion1 then begin
    if condicion2 then
        sentencia1
    else
        sentencia2
end
else
    sentencia3
```

Observa que escribimos el *if*, el *end* y el *else* en la misma columna

Hay una solución posiblemente mejor: evitar el anidamiento usando una única expresión

```
if condicion1 and condicion2 then  
    sentencia1  
if condicion1 and not condicion2  
    sentencia2  
if not condicion1 then  
    sentencia3
```

- Incluyendo las condiciones en una única expresión, evitamos niveles de anidamiento y resulta más legible
- Este enfoque no siempre es posible, en cada caso tendremos que evaluar qué resulta más claro

Cuando el segundo *if-then-else* esté en la rama *else*

- Tendremos una estructura muy habitual, llamada *if encadenados*
- Entonces no añadiremos el bloque *begin-end* (a menos que sea necesario porque haya más de una sentencia)
- Ya no se da el problema potencial que hemos descrito del *else* ambiguo

if encadenados

Los *if* encadenados, esto es, una sucesión de *else if*, *else if*, *else if*, son una estructura muy habitual

- *Si se da cierta condición haz esto, y si no, haz lo otro, y si tampoco, esto otro, y si tampoco ...*

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program if_encadenado;

const
    Mayoria_edad = 18;
    Adolescencia = 13;
    Edad = 10;

begin
    if Edad >= Mayoria_edad then
        writeln(Edad, ': Adulto')
    else if Edad > Adolescencia then
        writeln(Edad, ': Adolescente')
    else
        writeln(Edad, ': Niño');
end.
```

Resultado:

10: Niño

if anidado

En vez de encadenar

```
if then  
else if  
else if
```

tambien podríamos anidar, esto es, añadir un *begin-end* después del else y meter dentro el if-then-else

```
if then  
else begin  
    if ... then .. else  
end
```

Pero siempre es preferible encadenar, la estructura *else if* es muy habitual, muy clara y añadiendo otro *begin end*, solo lo complicamos

Varios if-then-else: resumen

Como resumen de todo lo anterior, recuerda:

Si una sentencia *if-then-else* tiene otro *if-then-else*...

- En la rama *then*, es potencialmente peligroso, así que siempre la anidaremos en un bloque *begin-end*
- En la rama *else*, es un if encadenado *else if, else if, else if*. No es peligroso, es muy común. No añadimos bloque *begin-end* (a menos, naturalmente, que sea imprescindible porque hay otras sentencias)

Anidamiento múltiple

¿Podemos escribir un if-then-else dentro de otro if-then-else que está dentro de un if-then-else que está un if-then-else que...?

- El lenguaje lo permite
- Pero si anidamos más de 2 o 3 niveles, muy probablemente estaremos escribiendo un programa de muy mala calidad, difícil de entender y propenso a errores
 - Esto es uno de los defectos peores y más típicos de quienes no saben programar
- El anidamiento excesivo casi siempre indica que el programador no ha sabido descomponer su problema en subproblemas (escribiendo nuevas funciones)

Recuerda que el encadenamiento múltiple (else if, else if, else if) es un caso diferente, que, bien hecho, no tiene por qué dar problemas

Case

Otra sentencia de control disponible en Pascal es `case`. Permite ejecutar diferentes acciones a partir de un valor discreto

- Es similar a `if`, la diferencia es que `if` considera un booleano y `case` puede considerar, además, tipos más complejos como `integer` o `char`
 - Pero nunca real: ha de ser un valor discreto
- Se parece mucho a los `if-then-else` encadenados. De hecho, cualquier cosa que se haga con `case`, también se puede hacer con `else-if`, `else-if`
 - Hay lenguajes que no tienen nada parecido a `case`, sus diseñadores consideraron que `else-if` es suficiente
 - La ventaja es que `case` puede resultar más claro que `else-if`

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program ejemplo_case;  
  
const  
    Edad = 14;  
begin  
    case Edad of  
        0 :  
            writeln('Bebé');  
        1..12 :  
            writeln('Niño');  
        13..17 :  
            writeln('Adolescente');  
        otherwise // sin dos puntos  
            writeln('Adulto');  
    end; // 'end' de case, único sin 'begin'  
end.
```

- El compilador evalúa la expresión que escribamos a continuación de la palabra reservada *case*. En este ejemplo, la constante *edad*
- Separamos todos los posibles valores, pudiendo usar rangos. Indicamos las acciones a realizar para cada conjunto de casos
- Si el valor no está dentro de ninguno de los rangos descritos, se ejecutan las sentencias a continuación de *otherwise*
- La rama *otherwise* se puede omitir, pero es preferible ponerla siempre

Ejemplo de case en una función

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }  
program case_en_funcion;  
function clasificacion_edad(edad: integer): string;  
begin  
    case edad of  
        0 :  
            result := 'Bebé';  
        1..12 :  
            result := 'Niño';  
        13..17 :  
            result := 'Adolescente';  
        otherwise // sin dos puntos  
            result := 'Adulto';  
    end; // 'end' de case, único sin 'begin'  
end;  
const  
    Edad_cliente = 14;  
begin  
    writeln( clasificacion_edad(Edad_cliente));  
end.
```

Adolescente

Ejemplo erróneo

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program case_erroneo;

function clasificacion_edad(edad: integer): string;
begin
    case edad of
        0 :
            result := 'Bebé';
        1..13 :
            result := 'Niño';
        13..17 :    // ¡Mal! El caso de 13 años está duplicado
                   // El compilador dará un error
            result := 'Adolescente';
        otherwise
            result := 'Adulto';
        end;
    end;
const
    Edad_cliente = 14;
begin
    writeln( clasificacion_edad(Edad_cliente));
end.
```

Otro ejemplo

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program case_hexa;
function valor_digito_hex(digito: char): integer;
begin
    case digito of
        '0':
            result := 0;
        '1'..'9':
            result := ord(digito)-ord('0');
        'A'..'F':
            result := 10+ord(digito)-ord('A');
        'a'..'f':
            result := 10+ord(digito)-ord('a');
        otherwise
            result := 0;
    end;
end;
const
    Digito = 'a' ;
begin
    writeln(digito, ': ', valor_digito_hex(Digito)); // Escribe a:10
end.
```


Case con múltiples sentencias

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program case_varias_sentencias;
const
    Edad = 14;
begin
    case Edad of
    0..1 : begin
        write('Menor de edad, ');
        writeln('bebé');
    end;
    2..12 : begin
        write('Menor de edad, ');
        writeln('niño');
    end;
    13..17 : begin
        write('Menor de edad, ');
        writeln('adolescente');
    end;
    otherwise      // sin dos puntos
        writeln('Adulto');
    end;          // 'end' de case, único sin 'begin'
end.
```

Ejemplo: fecha válida

https://gsyc.urjc.es/~mortuno/fpi/fecha_valida_v01.pas

https://gsyc.urjc.es/~mortuno/fpi/fecha_valida_v02.pas

https://gsyc.urjc.es/~mortuno/fpi/fecha_valida_v03.pas

Estructura de un programa

Como hemos visto,

- Los programas aceptan datos de entrada, los procesan mediante subprogramas (funciones o procedimientos) y producen datos de salida

Estos subprogramas se pueden clasificar de muchas formas, pero una estructura típica es preproceso, proceso principal y postproceso

- Subprogramas de preproceso
Toman los datos brutos de la entrada, comprueban que sean correctos y los preparan para el proceso principal, típicamente adaptando unidades o formatos
- Subprogramas de proceso principal
Realizan las tareas fundamentales del programa, a partir de datos *limpios*
- Subprogramas de postproceso, también llamados de formateado de salida
Los datos que salen del proceso principal, se adaptan al formato concreto necesario para su uso en la siguiente etapa

Estas ideas son universales en programación, aunque el vocabulario exacto puede cambiar

Ejemplo 1

Videojuego, módulo de movimiento de un personaje.

- Preproceso

Las órdenes del usuario pueden venir de un joystick, del teclado o del ratón. El preproceso convertirá todo esto a un único formato que especifique el movimiento

- Proceso principal

Recibe la órdenes de movimiento, la situación del personaje y el universo relevante, genera la nueva situación

- Postproceso

Esa situación se presenta en pantalla

Este es un ejemplo para ilustrar las ideas, un juego real es más complicado

Ejemplo 2

Módulo de un procesador de texto que inserta una imagen en un documento

- Preproceso

La imagen que facilita el usuario puede estar en muchos formatos: jpg, png, gif, bmp, etc. Internamente, el programa maneja el formato bmp, así que todos los ficheros que no estén en este formato, se convierten a bmp

- Proceso principal

La imagen se posiciona sobre la estructura interna que representa el documento

- Postproceso

A partir de la estructura interna del documento, se genera el pdf para imprimir o se compone el fragmento de página visible en ese momento

Este es un ejemplo para ilustrar las ideas, un procesador de texto real es más complicado

Ejemplo 3

Asistente virtual (Siri, Amazon Alexa, Google Assistant, Cortana)

- Preproceso
A partir del sonido de la voz, se extrae el texto y la identidad de quien habla
- Proceso principal
Mediante técnicas de inteligencia artificial, se hace el procesamiento del lenguaje natural
- Postproceso
Las órdenes se ejecutan: abrir una página web, leerla, reproducir una canción, encender una luz, ...

De nuevo, este ejemplo espera ser ilustrativo pero está muy simplificado

Precondiciones y postcondiciones

- Cada uno de estos conjuntos de subprogramas (preproceso, proceso y post-proceso) estará formado en general por varias funciones. O tal vez solo una, o incluso ninguna (nada que hacer)
- Para la correcta ejecución de una función de cualquier tipo es necesario que se den:
 - Precondiciones
Condiciones que han de cumplir los parámetros para que se pueda ejecutar la función.
Si no se cumplen, los parámetros recibidos serán erróneos y la función, si está bien programada, no se ejecutará
 - Postcondiciones
Condiciones que ha de cumplir el resultado de la función. Si no se cumplen, es que la función está mal programada

Ejemplos de precondición

- En una función que divida dos números, el divisor ha de ser no nulo
- En una función que calcule el factorial de un número, este ha de ser entero y positivo

Ejemplos de postcondición

- El factorial es un número ha de ser entero y positivo
- El seno de un ángulo ha de ser un número real entre -1 y 1

Cuando programemos una función, es muy importante comprobar las precondiciones. No hacerlo es:

- Muy peligroso y difícil de detectar: podemos hacer 1000 ejecuciones (con parámetros correctos) que provocarán resultados correctos. En la ejecución 1001 la función puede recibir parámetros erróneos. Si no los trata correctamente, normalmente se producirá un error severo
- Uno de los fallos de programación más habituales

Una función puede tener postcondiciones que sea conveniente comprobar, pero en la práctica esto no es tan habitual

En general, nuestras funciones deberán comprobar...

- Las precondiciones, casi siempre
- Las postcondiciones, algunas veces

¿Dónde comprobar precondiciones y postcondiciones?

- Si las precondiciones o postcondiciones son sencillas (tal vez una línea o dos), pueden hacerse en la misma función
- Si tienen más entidad, requerirán una nueva función

Ejemplo: inverso multiplicativo

```
{ $mode objfpc } { $H- } { $R+ } { $T+ } { $Q+ } { $V+ } { $D+ } { $X- } { $warnings on }
program inverso_multiplicativo;

function inverso(x: real): real;
begin
    result := 1/x;
end;

const
    A = 4.3;

begin
    if A <> 0 then begin // Precondición
        write('El inverso de ', A:0:3 );
        writeln(' es ', inverso(A):0:3);
    end
    else
        writeln('Error: el 0 no tiene inverso');
    end.
end.
```

Ejemplo: cambio de divisas

Supongamos una aplicación de comercio electrónico. Internamente trabaja siempre con dólares, pero los clientes pueden pagar en libras o euros, que el sistema se encarga de convertir en dólares

- Preproceso
Convertir la divisa de entrada en dólares, si es necesario
- Proceso principal
(no haremos)
- Postproceso
(no haremos)

Algunos conceptos sobre el dominio del problema

- Cada divisa tiene un código ISO 4271, formado por 3 letras mayúsculas:
USD: Dólar norteamericano
EUR: Euro
GBP: Libra esterlina
etc
- El precio de una divisa frente a otra se indica concatenando los códigos ISO 4217

Ejemplo: $\text{EURUSD} = 1.13$ significa que necesitamos 1.13 USD para comprar 1 EUR

- A la primera divisa (EUR en este caso) se la denomina *divisa base*, es la divisa que compramos
- La segunda divisa (USD) es la *divisa cotizada*, la divisa que usamos para pagar

Si necesitamos el par inverso, podemos dividir entre 1

$$\text{USDEUR} = 1 / 1.13 = 0.88$$

(aunque en la vida real puede haber pequeñas diferencias por las comisiones)

Nuestro programa recibirá:

- Un importe, un número real
- El código ISO 4271 de la divisa, una cadena

La función principal, *a_dolar*, convierte la divisa de entrada en dólares

- La precondición es que la divisa sea EUR, GBP o USD. Si no se cumple, no se invocará a la función
- La función *a_dolar* nunca recibirá una divisa distinta a estas tres, pero *por si acaso*, hacemos una comprobación redundante. Si algo falla y no se detectó en la comprobación de precondiciones, el programa se detiene abruptamente

<https://gsyc.urjc.es/~mortuno/fpi/divisas.pas>