# Prog. Básica - Laboratorio 2 Instrucciones condicionales

**Nombre:** Oihan Irastorza  **Fecha**: 29/09/2020

Establece para cada uno de los siguientes enunciados:

a) si es ambiguo el enunciado.

b) la correspondiente especificación.

c) una lista de casos de prueba.

d) el diseño de un algoritmo para llevarlo a cabo.

e) indicar qué líneas del algoritmo se ejecutan

**1. Resultado de la ONCE.** Crea un algoritmo que, dados dos números de cuatro cifras (el número de cupón premiado y el número de cupón que hay que comprobar), obtenga el premio que le corresponde al cupón a comprobar, sabiendo que si tiene los cuatro dígitos iguales (y en el mismo orden) le corresponden 100000 euros, si coinciden los tres últimos dígitos le corresponden 50000 euros y si son los dos últimos el premio son 3 euros. El resto de casos no tienen premio asociado.

**2. Segundo anterior.** Crea un algoritmo que, dada una hora válida (*horas*, *minutos* y *segundos*), calcule qué hora era un segundo antes. Por ejemplo, la hora anterior a 10:30:40 es 10:30:39.

**3. Añadir duración a una tarea.** Crea un algoritmo que, dada la hora de comienzo de una tarea y la duración en segundos de ésta, calcule la hora en que finalizará la tarea. La duración de la tarea puede ser de varias horas, pero nunca superior a media jornada (14400s). Las tareas incompletas continúan en la siguiente jornada. La jornada laboral es de 8:00 a 16:00.

**4. Día anterior.** Crea un algoritmo que, dada una fecha (*día*, *mes* y *año*) calcule el día anterior. Comienza considerando que todos los años no son bisiestos. Luego añade la condición de que el año puede ser bisiesto. Los años bisiestos son los que son múltiplos de 4 y no ser múltiplos de 100. Los múltiplos de 400 son una excepción, ya que, aun siendo múltiplos de 100, son bisiestos.

**5. Estación del año.** Crea un algoritmo que dados dos números enteros día y mes, indique a qué estación del año corresponde dicha fecha. Se considera que la primavera va desde el 21 de marzo hasta el 20 de junio; el verano se extiende desde el 21 de junio al 20 de septiembre; el otoño dura desde el 21 de septiembre hasta el 20 de diciembre; y el invierno ocurre entre el 21 de diciembre y el 20 de marzo.

1. **Resultado de la ONCE**

Crea un algoritmo que, dados dos números de cuatro cifras (el número de cupón premiado y el número de cupón que hay que comprobar), obtenga el premio que le corresponde al cupón a comprobar, sabiendo que si tiene los cuatro dígitos iguales (y en el mismo orden) le corresponden 100000 euros, si coinciden los tres últimos dígitos le corresponden 50000 euros y si son los dos últimos el premio son 3 euros. El resto de casos no tienen premio asociado

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado?

Después, rellena la especificación:

|  |
| --- |
| Datos: cupon\_premiado y cupón, ambos números naturales.  Pre: 2 números naturales de 4 dígitos.  Resultado: premio, un numero natural. El premio correspondiente al cupón pasado por parámetro.  Post: si los 4 dígitos coinciden, el premio será de 100000. Si los 3 últimos dígitos coinciden, 50000. Los 2 últimos, 3. Para el resto de los casos, premio ha de ser 0. |

* 1. Casos de Prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Premiado | Cupón 2 | Descripción | Premio |
| 1 | 4540 | 4540 | Todos los números coinciden | 100000 |
| 2 | 4540 | 5540 | Últimos 3 números coinciden | 50000 |
| 3 | 4540 | 6640 | Últimos 2 números coinciden | 3 |
| 4 | 4540 | 7770 | Último número coincide | 0 |
| 5 | 4540 | 8888 | Ningún número coincide | 0 |

* 1. Algoritmo

-- Funcion que comprueba si el cupon introducido es premiado

FUNCTION resultado\_once(cupon\_premiado, cupon: Natural) return Natural IS

-- Datos: cupon\_premiado y cupon, ambos numeros naturales

-- Pre: 2 numeros naturales de 4 digitos

-- Resultado: premio, un numero natural. El premio correspondiente al cupon pasado por parametro

-- Post: si los 4 digitos coinciden, el premio sera de 100000. Si los 3 ultimos digitos coinciden, 50000. Los 2 ultimos 3. Para el resto de los casos, premio ha de ser 0.

premio: Natural;

BEGIN

if cupon\_premiado < 10000 AND cupon < 10000 then

if (cupon = cupon\_premiado) then -- Si todos los numeros coinciden

premio := 100000;

elsif(cupon rem 1000 = cupon\_premiado rem 1000) then -- Si los 3 ultimos numeros coinciden

premio := 50000;

elsif(cupon rem 100 = cupon\_premiado rem 100) then -- Si los 2 ultimos numeros coinciden

premio := 3;

else

premio := 0;

end if;

else

raise constraint\_error with "Introduce un numero de cupon correcto";

end if;

return premio;

END resultado\_once;

1. **Segundo anterior**

Crea un algoritmo que, dada una hora válida (*horas*, *minutos* y *segundos*), calcule qué hora era un segundo antes. Por ejemplo, la hora anterior a 10:30:40 es 10:30:39.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? No

Después, rellena la especificación:

|  |
| --- |
| Datos: 3 parámetros de entrada: h(horas), m(minutos), s(segundos). Indican una hora determinada.  Pre: 3 números naturales (h, m, s), una hora valida: h > 0 y < 24. m >= 0 y < 60. s >= 0 y < 60.    -- Resultado: 3 parámetros de salida: h(horas), m(minutos), s(segundos). Indican la hora introducida, pero 1 segundo antes.  -- Post: Si s es 0, se convierte en 59. Sino, se le resta 1. En caso de s = 0, se realiza la misma comprobación con los minutos. En caso de m = 0, se hace lo mismo con h(horas). El resultado final tiene que constar de 3 números naturales. |

* 1. Casos de Prueba

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | hh | mm | ss | Descripción | hh | mm | ss |
| 1 | 10 | 30 | 10 | Caso general 1: cambia *ss* | 10 | 30 | 9 |
| 2 | 10 | 59 | 10 | Caso general 2: cambia *ss* | 10 | 59 | 9 |
| 3 | 10 | 59 | 0 | Cambian los ss en 0 | 10 | 58 | 59 |
| 4 | 10 | 0 | 0 | Cambian los minutos en 0 | 9 | 59 | 59 |
| 5 | 0 | 0 | 0 | Cambian las horas en 0 | 23 | 59 | 59 |

* 1. Algoritmo

PROCEDURE segundo\_anterior(h, m, s: in out Natural) IS

-- Datos: 3 parametros de entrada: h(horas), m(minutos), s(segundos). Indican una hora determinada.

-- Pre: 3 numeros naturales(h, m, s), una hora valida: h > 0 y < 24. m >= 0 y < 60. s >= 0 y < 60.

-- Resultado: 3 parametros de salida: h(horas), m(minutos), s(segundos). Indican la hora introducida pero 1 segundo antes.

-- Post: Si s es 0, se convierte en 59. Sino, se le resta 1. En caso de s = 0, se realiza la misma comprobacion con los minutos. En caso de m = 0, se hace lo mismo con h(horas). El resultado final tiene

--que constar de 3 numeros naturales.

BEGIN

if s=0 then

s:=59;

if m=0 then

m:=59;

if h=0 then

h:=23;

else

h:=h-1;

end if;

else

m:=m-1;

end if;

else

s:=s-1;

end if;

END segundo\_anterior;

* 1. Simulación

|  |  |
| --- | --- |
| hh:mm:ss | Simulación: indica qué líneas del algoritmo se ejecutan |
| 10:30:10 | PROCEDURE segundo\_anterior(h, m, s: in out Natural) IS  -- Datos: 3 parametros de entrada: h(horas), m(minutos), s(segundos). Indican una hora determinada.  -- Pre: 3 numeros naturales(h, m, s), una hora valida: h > 0 y < 24. m >= 0 y < 60. s >= 0 y < 60.    -- Resultado: 3 parametros de salida: h(horas), m(minutos), s(segundos). Indican la hora introducida pero 1 segundo antes.  -- Post: Si s es 0, se convierte en 59. Sino, se le resta 1. En caso de s = 0, se realiza la misma comprobacion con los minutos. En caso de m = 0, se hace lo mismo con h(horas). El resultado final tiene  --que constar de 3 numeros naturales.  BEGIN  if s=0 then  s:=59;  if m=0 then  m:=59;  if h=0 then  h:=23;  else  h:=h-1;  end if;  else  m:=m-1;  end if;  else  s:=s-1;  end if;  END segundo\_anterior; |

1. **Añadir duración a una tarea**

Crea un algoritmo que, dada la hora de comienzo de una tarea y la duración en segundos de ésta, calcule la hora en que finalizará la tarea. La duración de la tarea puede ser de varias horas, pero nunca superior a media jornada (14400s). Las tareas incompletas continúan en la siguiente jornada. La jornada laboral es de 8:00 a 16:00.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? No

Después, rellena la especificación:

|  |
| --- |
| Datos: 3 parámetros, h(horas), m(minutos) y s(segundos), que son Natural. Y duración (de la tarea) que es Positive.  Pre: h, m, s son números Naturales (>=0) y duración es un numero positivo (>0).    Resultado: Resultado: 4 parámetros out. Hora de finalización de la tarea. 3 resultados, h(hora), m(minuto) y s(segundo), las 3 de tipo Natural. Y dia\_siguiente (parámetro booleano que muestra si la tarea se finaliza el día siguiente) de tipo Boolean.  Post: Post: si la jornada excede las 4 horas, el programa finaliza con error. Sino, si la duración de la tarea excede el fin de la jornada (teniendo en cuenta la hora inicial y que la jornada acaba a las 16:00) se acaba el día siguiente y dia\_siguiente es true. La tarea ha de finalizar entre las 8:00:01 y las 16:00:00 |

* 1. Casos de Prueba

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | hh:mm:ss | duración | Descripción | hh:mm:ss | día\_siguiente |
| 1 | 10:30:30 | 30 | medio minuto | 10:31:00 | false |
| 2 | 10:30:30 | 1800 | media hora | 11:00:30 | false |
| 3 | 10:40:30 | 3606 | una hora y 6 segundos | 11:40:36 | false |
| 4 | 15:50:10 | 3600 | cambiamos de jornada | 8:50:10 | true |
| 5 | 15:50:10 | 14400 | máxima duración de tarea | 11:50:10 | true |

* 1. Algoritmo

PROCEDURE duracion\_tarea(h, m, s: in out Natural; duracion: in Positive; dia\_siguiente: out Boolean) IS

-- Datos: 3 parametros, h(horas), m(minutos) y s(segundos), que son Natural. Y duracion(de la tarea) que es Positive

-- Pre: h, m, s son numeros Naturales(>=0) y duracion es un numero positivo (>0)

-- Resultado: 4 parametros out. Hora de finalizacion de la tarea. 3 resultados, h(hora), m(minuto) y s(segundo), las 3 de tipo Natural. Y dia\_siguiente

-- (parametro booleano que muestra si la tarea se finaliza el dia siguiente) de tipo Boolean.

-- Post: si la jornada excede las 4 horas, el programa finaliza con error. Sino, si la duracion de la tarea excede el fin de la jornada

-- (teniendo en cuenta la hora inicial y que la jornada acaba a las 16:00) se acaba el dia siguiente y dia\_siguiente es true. La tarea ha de finalizar entre las 8:00:01 y las 16:00:00

hora\_en\_segundos: Natural:= (h\*3600) + (m \* 60) + s;

duracion\_restante: Natural := duracion; -- La pasamos a una variable del programa, porque los parametros IN no se pueden sobreescribir

BEGIN

if duracion < 14401 then -- Si la jornada dura mas de 4 horas (14401s)

if (hora\_en\_segundos + duracion > 57600) then -- Si duracion + hora inicial excede las 16 horas (57600s)

dia\_siguiente := true;

duracion\_restante := ((duracion + hora\_en\_segundos) - 57600);

hora\_en\_segundos := 28800; -- Le sumamos a la duracion restante, las 8 de la mañana(28800)

end if;

h := (hora\_en\_segundos + duracion\_restante) / 3600;

m := ((hora\_en\_segundos + duracion\_restante) / 60) rem 60;

s := (hora\_en\_segundos + duracion\_restante) rem 60;

else

raise constraint\_error with "La duracion de la tarea no puede exceder la media jornada.";

end if;

END duracion\_tarea;

* 1. Simulación

|  |  |
| --- | --- |
| hh:mm:ss | Simulación: indica qué líneas del algoritmo se ejecutan |
| 10:30:10  14400s | PROCEDURE duracion\_tarea(h, m, s: in out Natural; duracion: in Positive; dia\_siguiente: out Boolean) IS  -- Datos: 3 parametros, h(horas), m(minutos) y s(segundos), que son Natural. Y duracion(de la tarea) que es Positive  -- Pre: h, m, s son numeros Naturales(>=0) y duracion es un numero positivo (>0)    -- Resultado: 4 parametros out. Hora de finalizacion de la tarea. 3 resultados, h(hora), m(minuto) y s(segundo), las 3 de tipo Natural. Y dia\_siguiente  -- (parametro booleano que muestra si la tarea se finaliza el dia siguiente) de tipo Boolean.  -- Post: si la jornada excede las 4 horas, el programa finaliza con error. Sino, si la duracion de la tarea excede el fin de la jornada  -- (teniendo en cuenta la hora inicial y que la jornada acaba a las 16:00) se acaba el dia siguiente y dia\_siguiente es true. La tarea ha de finalizar entre las 8:00:01 y las 16:00:00  hora\_en\_segundos: Natural:= (h\*3600) + (m \* 60) + s;  duracion\_restante: Natural := duracion; -- La pasamos a una variable del programa, porque los parametros IN no se pueden sobreescribir  BEGIN  if duracion < 14401 then -- Si la jornada dura mas de 4 horas (14401s)  if (hora\_en\_segundos + duracion > 57600) then -- Si duracion + hora inicial excede las 16 horas (57600s)  dia\_siguiente := true;  duracion\_restante := ((duracion + hora\_en\_segundos) - 57600);  hora\_en\_segundos := 28800; -- Le sumamos a la duracion restante, las 8 de la mañana(28800)  end if;  h := (hora\_en\_segundos + duracion\_restante) / 3600;  m := ((hora\_en\_segundos + duracion\_restante) / 60) rem 60;  s := (hora\_en\_segundos + duracion\_restante) rem 60;  else  raise constraint\_error with "La duracion de la tarea no puede exceder la media jornada.";  end if;  END duracion\_tarea; |

1. **Día anterior**

Crea un algoritmo que, dada una fecha (*día*, *mes* y *año*) calcule el día anterior. Comienza considerando que todos los años no son bisiestos. Luego añade la condición de que el año puede ser bisiesto. Los años bisiestos son los que son múltiplos de 4 y no ser múltiplos de 100. Los múltiplos de 400 son una excepción, ya que, aun siendo múltiplos de 100, son bisiestos.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? No

Después, rellena la especificación:

|  |
| --- |
| Datos: 3 parámetros, yy(año), mm(mes) y dd(día). yy es Natural, y mm y dd son Positive  Pre: yy >= 0 (implícito al tiparlo como Natural), mm y dd > 0 (implícito al tiparlos como Positive)  Resultados: 3 parámetros de salida que muestra el día anterior: yy(año), mm(mes) y dd(día). yy Natural, y mm y dd Positive.  Post: si dd es 1, entonces restamos 1 a mm y día pasa a ser el último día de ese mes (el día será 28, 29, 30 o 31 según el mes y si el año es bisiesto). Sino restamos 1 a dd. Si el mes es igual a 0, restamos 1 al año y mes será 12. Sino restamos 1 a mm. Y si año es 0 se muestra un error para indicar que ese es el primer día admitido por el programa. Sino, se resta 1 al año. |

* 1. Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| # | dd-mm-aaaa | Descripción | dd-mm-aaaa |
| 1 | 01-01-2006 | día 1 tras mes de 31 días | 31-12-2005 |
| 2 | 01-03-2006 | día 1 tras febrero de año no bisiesto | 28-02-2006 |
| 3 | 01-03-2000 | día 1 tras febrero de año bisiesto | 29-02-2000 |
| 4 | 01-05-2006 | día 1 tras mes de 30 días (1) | 30-04-2006 |
| 5 | 01-10-2006 | día 1 tras mes de 30 días (2) | 30-09-2006 |
| 6 | 13-03-2006 | un día cualquiera de un mes cualquiera | 12-03-2006 |

* 1. Algoritmo

PROCEDURE dia\_anterior(yy: in out Natural; mm, dd: in out Positive) IS

-- Datos: 3 parametros, yy(año), mm(mes) y dd(dia). yy es Natural, y mm y dd son Positive

-- Pre: yy >= 0 (implicito al tiparlo como Natural), mm y dd > 0 (implicito al tiparlos como Positive)

-- Resultado: 3 parametros de salida que muestra el dia anterior: yy(año), mm(mes) y dd(dia). yy Natural, y mm y dd Positive.

-- Post: si dd es 1, entonces restamos 1 a mm y dia pasa a ser el ultimo dia de ese mes (el dia sera 28, 29, 30 o 31 segun el mes y si el año es bisiesto)

-- Sino restamos 1 a dd. Si el mes es igual a 0, restamos 1 al año y mes sera 12. Sino restamos 1 a mm. Y si año es 0 se muestra un error para indicar que ese es el primer dia admitido por el programa.

-- Sino, se resta 1 al año

es\_bisiesto: Integer range 0..1;

BEGIN

if (dd = 1) then

if (mm = 1) then

mm := 12;

if (yy = 0) then

raise constraint\_error with "Ha llegado al ultimo dia a.C.";

else

yy := yy - 1;

end if;

else

mm := mm - 1;

end if;

-- Comprobar si el año es bisiesto

es\_bisiesto := (if (yy rem 4 = 0) and ((yy rem 100 /= 0) or (yy rem 400 = 0)) then 1 else 0);

-- Formula para calcular dias de cada mes, idea tomada de:

-- http://www.dispersiondesign.com/articles/time/number\_of\_days\_in\_a\_month

dd := 31 - (if mm = 2 then (3 - es\_bisiesto) else (mm - 1) rem 7 rem 2);

else

dd := dd - 1;

end if;

END dia\_anterior;dispersiondesign.com/articles/time/number\_of\_days\_in\_a\_month

dd := 31 - (if mm = 2 then (3 - es\_bisiesto) else (mm - 1) rem 7 rem 2);

else

dd := dd - 1;

end if;

END dia\_anterior;

* 1. Simulación

|  |  |
| --- | --- |
| dd-mm-aaaa | Simulación: indica qué líneas del algoritmo se ejecutan |
| 24-5-2020 | PROCEDURE dia\_anterior(yy: in out Natural; mm, dd: in out Positive) IS  -- Datos: 3 parametros, yy(año), mm(mes) y dd(dia). yy es Natural, y mm y dd son Positive  -- Pre: yy >= 0 (implicito al tiparlo como Natural), mm y dd > 0 (implicito al tiparlos como Positive)    -- Resultado: 3 parametros de salida que muestra el dia anterior: yy(año), mm(mes) y dd(dia). yy Natural, y mm y dd Positive.  -- Post: si dd es 1, entonces restamos 1 a mm y dia pasa a ser el ultimo dia de ese mes (el dia sera 28, 29, 30 o 31 segun el mes y si el año es bisiesto)  -- Sino restamos 1 a dd. Si el mes es igual a 0, restamos 1 al año y mes sera 12. Sino restamos 1 a mm. Y si año es 0 se muestra un error para indicar que ese es el primer dia admitido por el programa.  -- Sino, se resta 1 al año  es\_bisiesto: Integer range 0..1;  BEGIN  if (dd = 1) then  if (mm = 1) then  mm := 12;  if (yy = 0) then  raise constraint\_error with "Ha llegado al ultimo dia a.C.";  else  yy := yy - 1;  end if;  else  mm := mm - 1;  end if;  -- Comprobar si el año es bisiesto  es\_bisiesto := (if (yy rem 4 = 0) and ((yy rem 100 /= 0) or (yy rem 400 = 0)) then 1 else 0);  -- Formula para calcular dias de cada mes, idea tomada de:  -- http://www.dispersiondesign.com/articles/time/number\_of\_days\_in\_a\_month  dd := 31 - (if mm = 2 then (3 - es\_bisiesto) else (mm - 1) rem 7 rem 2);  else  dd := dd - 1;  end if;  END dia\_anterior; |

1. **Estación del año**

Crea un algoritmo que dados dos números enteros *día* y *mes,* indique a qué estación del año corresponde dicha fecha. Se considera que la primavera va desde el 21 de marzo hasta el 20 de junio; el verano se extiende desde el 21 de junio al 20 de septiembre; el otoño dura desde el 21 de septiembre hasta el 20 de diciembre; y el invierno ocurre entre el 21 de diciembre y el 20 de marzo.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado?

Después, rellena la especificación:

|  |
| --- |
| Datos: 2 parámetros, dd(día) y mm(mes), ambos Positive  Pre: 2 números positivos (>0). Esto será así siempre puesto que están tipados como Positive.    Resultados: se retorna la estación, un String de 9 caracteres de largo.  Post: se le resta 1 al mes si el día esta entre 1 y 20. El programa retornara una estación u otra comprobando que mes empieza y que mes acaba. El resultado del programa tiene que ser un string de 9 caracteres de largo, que es el espacio mínimo para escribir cualquiera de las 4 estaciones (primavera es la mas larga, 9 caracteres). |

* 1. Casos de Prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | día | mes | Descripción | estación |
| 1 | 2 | 10 | Fecha contenida en otoño | Otoño |
| 2 | 14 | 2 | Fecha contenida en inverno | Invierno |
| 3 | 20 | 4 | Fecha contenida en primavera | Primavera |
| 4 | 23 | 8 | Fecha contenida en verano | Verano |

* 1. Algoritmo

FUNCTION estacion\_ano(dd, mm: Positive) return String IS

dias\_en\_mes: Integer := 31 - (if mm = 2 then 2 else (mm - 1) rem 7 rem 2);

mes\_estacion: Integer := mm; -- Hay que crear una nueva variable puesto que no se puede editar un parametro in

estacion: string(1..9);

-- Datos: 2 parametros, dd(dia) y mm(mes), ambos Positive.

-- Pre: 2 numeros positivos (>0). Esto sera asi siempre puesto que estan tipados como Positive.

-- Resultado: se retorna la estacion, un String de 9 caracteres de largo.

-- Post: se le resta 1 al mes si el dia esta entre 1 y 20. El programa retornara una estacion u otra comprobando que mes empieza y que mes acaba. El resultado del programa tiene que ser

-- un string de 9 caracteres de largo, que es el espacio minimo para escribir cualquiera de las 4 estaciones (primavera es la mas larga, 9 caracteres).

BEGIN

if dd in 1..20 then

if mes\_estacion = 1 then

mes\_estacion := 12;

else

mes\_estacion := mes\_estacion - 1;

end if;

end if;

if (dd > 0 and dd <= dias\_en\_mes) and (mm > 0 and mm <= 12) then

if mes\_estacion = 12 or mes\_estacion in 1..2 then

estacion := "invierno ";

elsif mes\_estacion in 3..5 then

estacion := "primavera";

elsif mes\_estacion in 6..8 then

estacion := " verano ";

elsif mes\_estacion in 9..11 then

estacion := " otono ";

end if;

else

raise constraint\_error with "Introduzca un dia y mes correctos";

end if;

return estacion;

END estacion\_ano;

return estacion;

END estacion\_ano;

* 1. Simulación

|  |  |  |
| --- | --- | --- |
| dd | mes | Simulación: indica qué líneas del algoritmo se ejecutan |
| 12 | 1 | FUNCTION estacion\_ano(dd, mm: Positive) return String IS  dias\_en\_mes: Integer := 31 - (if mm = 2 then 2 else (mm - 1) rem 7 rem 2);  mes\_estacion: Integer := mm; -- Hay que crear una nueva variable puesto que no se puede editar un parametro in  estacion: string(1..9);  -- Datos: 2 parametros, dd(dia) y mm(mes), ambos Positive.  -- Pre: 2 numeros positivos (>0). Esto sera asi siempre puesto que estan tipados como Positive.    -- Resultado: se retorna la estacion, un String de 9 caracteres de largo.  -- Post: se le resta 1 al mes si el dia esta entre 1 y 20. El programa retornara una estacion u otra comprobando que mes empieza y que mes acaba. El resultado del programa tiene que ser  -- un string de 9 caracteres de largo, que es el espacio minimo para escribir cualquiera de las 4 estaciones (primavera es la mas larga, 9 caracteres).    BEGIN  if dd in 1..20 then  if mes\_estacion = 1 then  mes\_estacion := 12;  else  mes\_estacion := mes\_estacion - 1;  end if;  end if;  if (dd > 0 and dd <= dias\_en\_mes) and (mm > 0 and mm <= 12) then  if mes\_estacion = 12 or mes\_estacion in 1..2 then  estacion := "invierno ";  elsif mes\_estacion in 3..5 then  estacion := "primavera";  elsif mes\_estacion in 6..8 then  estacion := " verano ";  elsif mes\_estacion in 9..11 then  estacion := " otono ";  end if;  else  raise constraint\_error with "Introduzca un dia y mes correctos";  end if;  return estacion;  END estacion\_ano; |