Oinarrizko Programazioa – 5. Laborategiko informea

Izena: Eneko, Gontzal eta Markel **Data**: 2014-10-28

ikasitako kontzeptuak eta lortutako gaitasunak

Behin laborategia bukatu dudala, eskatu dizkidaten ariketan ondo ulertu ditudala kontsideratzen dut, eta beraz errepikatzeko gai izango nintzatekela esan dezaket. Erantzun hurrengo galdera-sorta erantzunez ea gai zaren (ala ez) hor galdetzen dena egiteko.

	Eneko	**BAI**	**EZ**
1	Jakin azpiprograma bat noiz den funtzio eta noiz den prozedura bat.	Bai	
2	Programa nagusia eta azpiprograma arteko informazio trakaketa (paso de parámetros por copia edo kopia bitarteko parametro trukaketa)	Bai	
3	Ulertu funtzioek programa nagusiari **informazioa** bueltatzeko erabiltzen duten *return* mekanismoa.	Bai	
4	Ulertu prozedurak programa nagusiari **informazioak** bueltatzeko erabiltzen duten erabiltzen duten OUT mekanismoa.	Bai	
5	Ulertu zergaitik hurrengo azpiprogramei egindako deiak txarto dauden. 1) ZifraKopurua funtzioa izanda zeinen burua hurrengoa den: *function zifraKopurua(num: in integer) return integer is* ---Sarrera: zenbaki oso bat ---Aurre: num:balor1 >10 ---Irteera: zenbaki oso bat ---Post: num-ek duen zifra kopurua bueltatuko du |nzifrak >1 programa nagusian dagoen deia hurrengoa da: ….. *zifraKopurua*(1234); **Gaizki dago. "Put(*zifraKopurua*(1234));" ipini behar da.**		

	2) hurrengo prozedura izanik: procedure ordenMotaKonprobatu(num: in integer; goranzko: out boolean; beheranzko: out boolean) is ---Sarrera: zenbaki oso bat ---Aurre: num:balor1 >10 ---Irteera: zenbaki oso bat ---Post: num-ek dituen zifrak goranzko ordena jarraitzen dutenean goranzko true balioko du, aldiz beheranzko ordena jarraitzen dutenean beheranzko true balioko du. Eta ez dutenean ez goranzko ezta beheranzko ordena jarraitzen, biak false izango dira. Programa nagusian dagoen deia hurrengoa da: ….. ordenMotaKonprobatu(1234); **Gaizki dago. Ezin dira bi balio "out" parentesian egon.**		
6	Hurrengo funtzioa edukita: *function zifraKopurua(num: in integer) return integer is* *kont:integer:=0;* *begin* *loop exit when num=0;* *kont:=kont+1;* *num:=num/10;* *end loop;* *return(kont);* *end zifraKopurua;* *Badakit kode hau zergaitik dagoen gaizki.*		
7	BESTELAKOAK: Argitu zeintzuk		

	Gontzal	**BAI**	**EZ**
1	Jakin azpiprograma bat noiz den funtzio eta noiz den prozedura bat.	Bai	
2	Programa nagusia eta azpiprograma arteko informazio trakaketa (paso de	Bai	

	parámetros por copia edo kopia bitarteko parametro trukaketa)		
3	Ulertu funtzioek programa nagusiari **informazioa** bueltatzeko erabiltzen duten *return* mekanismoa.	Bai	
4	Ulertu prozedurak programa nagusiari **informazioak** bueltatzeko erabiltzen duten erabiltzen duten OUT mekanismoa.	Bai	
5	Ulertu zergaitik hurrengo azpiprogramei egindako deiak txarto dauden. 2) ZifraKopurua funtzioa izanda zeinen burua hurrengoa den: *function zifraKopurua(num: in integer) return integer is* ---Sarrera: zenbaki oso bat ---Aurre: num:balor1 >10 ---Irteera: zenbaki oso bat ---Post: num-ek duen zifra kopurua bueltatuko du |nzifrak >1 programa nagusian dagoen deia hurrengoa da: ….. *zifraKopurua*(1234); **Gaizki dago. "Put(*zifraKopurua*(1234));" ipini behar da.** 3) hurrengo prozedura izanik: procedure ordenMotaKonprobatu(num: in integer; goranzko: out boolean; beheranzko: out boolean) is ---Sarrera: zenbaki oso bat ---Aurre: num:balor1 >10 ---Irteera: zenbaki oso bat ---Post: num-ek dituen zifrak goranzko ordena jarraitzen dutenean goranzko true balioko du, aldiz beheranzko ordena jarraitzen dutenean beheranzko true balioko du. Eta ez dutenean ez goranzko ezta beheranzko ordena jarraitzen, biak false izango dira. Programa nagusian dagoen deia hurrengoa da: ….. ordenMotaKonprobatu(1234); **Gaizki dago. Ezin dira bi balio "out" parentesian egon.**		
6	Hurrengo funtzioa edukita: *function zifraKopurua(num: in integer) return integer is* *kont:integer:=0;* *begin* *loop exit when num=0;* *kont:=kont+1;* *num:=num/10;* *end loop;*		

	return(kont); *end zifraKopurua;* *Badakit kode hau zergaitik dagoen gaizki.*		
7	BESTELAKOAK: Argitu zeintzuk		

	Markel	**BAI**	**EZ**
1	Jakin azpiprograma bat noiz den funtzio eta noiz den prozedura bat.	Bai	
2	Programa nagusia eta azpiprograma arteko informazio trakaketa (paso de parámetros por copia edo kopia bitarteko parametro trukaketa)	Bai	
3	Ulertu funtzioek programa nagusiari **informazioa** bueltatzeko erabiltzen duten *return* mekanismoa.	Bai	
4	Ulertu prozedurak programa nagusiari **informazioak** bueltatzeko erabiltzen duten erabiltzen duten OUT mekanismoa.	Bai	
5	Ulertu zergaitik hurrengo azpiprogramei egindako deiak txarto dauden. 3) ZifraKopurua funtzioa izanda zeinen burua hurrengoa den: *function zifraKopurua(num: in integer) return integer is* ---Sarrera: zenbaki oso bat ---Aurre: num:balor1 >10 ---Irteera: zenbaki oso bat ---Post: num-ek duen zifra kopurua bueltatuko du |nzifrak >1 programa nagusian dagoen deia hurrengoa da: ….. *zifraKopurua*(1234); **Gaizki dago. "Put(*zifraKopurua*(1234));" ipini behar da.** 4) hurrengo prozedura izanik: procedure ordenMotaKonprobatu(num: in integer; goranzko: out boolean; beheranzko: out boolean) is ---Sarrera: zenbaki oso bat ---Aurre: num:balor1 >10 ---Irteera: zenbaki oso bat ---Post: num-ek dituen zifrak goranzko ordena jarraitzen dutenean		

	goranzko true balioko du, aldiz beheranzko ordena jarraitzen dutenean beheranzko true balioko du. Eta ez dutenean ez goranzko ezta beheranzko ordena jarraitzen, biak false izango dira. Programa nagusian dagoen deia hurrengoa da: ….. ordenMotaKonprobatu(1234); **Gaizki dago. Ezin dira bi balio "out" parentesian egon.**		
6	Hurrengo funtzioa edukita: *function zifraKopurua(num: in integer) return integer is* *kont:integer:=0;* *begin* *loop exit when num=0;* *kont:=kont+1;* *num:=num/10;* *end loop;* *return(kont);* *end zifraKopurua;* *Badakit kode hau zergaitik dagoen gaizki.*		
7	BESTELAKOAK: Argitu zeintzuk		

1º ejercicio

Numero_de_0s_y_1s: Dado un número binario que comienza en 1 implementar un subprograma tal que nos indique si el número binario contiene la misma cantidad de 0s y 1s. Utilizad misma_cantidad_de_0s_y_unos.adb y prueba_misma_cantidad_de_0s_y_1s.adb. Primero leeré atentamente la especificación del (de los) subprograma(s) que me pidan implementar. Primera pregunta que os deberíais hacer, ¿el subprograma que he de hacer debe obtener/calcular una sola cosa o varias? Es decir, ¿será una función o un procedimiento?

prueba_misma_cantidad_de_0s_y_1s.adb

```
with Ada.Integer_Text_IO,Ada.Text_IO;---use añade junto con

---las librerías de entrada y salida el subprograma misma_cantidad_de_0s_y_1s

use Ada.Integer_Text_IO,Ada.Text_IO;

with misma_cantidad_de_0s_y_1s;


procedure prueba_misma_cantidad_de_0s_y_1s  is


numBin:integer;


begin
   put_line("Primera prueba: 1");
   put("Tu programa debería devolver false indicando que el número no contiene la misma cantidad de 0s y 1s");
   new_line;
   numBin:=1;
   if misma_cantidad_de_0s_y_1s(numBin)/=true then
       put("Y devuelve true. Tu programa funciona correctamente");
       new_line;
   else
      put("Revisa tu programa, este no funciona correctamente");
       new_line;
   end if;
   put("Pulsa intro para continuar");
   Skip_Line;
   put_line("Segunda prueba: 10");
   put("Tu programa debería devolver true indicando que el número contiene la misma cantidad de 0s y 1s");
   new_line;
   numBin:=10;
```

```
    if misma_cantidad_de_0s_y_1s(numBin)=true then
        put("Y devuelve true. Tu programa funciona correctamente");
        new_line;
    else
       put("Revisa tu programa, este no funciona correctamente");
        new_line;
    end if;
    put("Pulsa intro para continuar");
    Skip_Line;
    put_line("Tercera prueba: 101010");
    put("Tu programa debería devolver true indicando que el número contiene la misma
cantidad de 0s y 1s");
    new_line;
    numBin:=101010;
    if misma_cantidad_de_0s_y_1s(numBin)=true then
        put("Y devuelve true. Tu programa funciona correctamente");
        new_line;
    else
       put("Revisa tu programa, este no funciona correctamente");
        new_line;
    end if;
    put("Pulsa intro para continuar");
    Skip_Line;
    put_line("Cuarta prueba: 101110");
    put("Tu programa debería devolver false indicando que el número no contiene la misma
cantidad de 0s y 1s");
    new_line;
    numBin:=101110;
    if misma_cantidad_de_0s_y_1s(numBin)/=true then
        put("Y devuelve false. Tu programa funciona correctamente");
        new_line;
    else
       put("Revisa tu programa, este no funciona correctamente");
        new_line;
    end if;
    put("Pulsa intro para continuar");
```

```
   Skip_Line;

end prueba_misma_cantidad_de_0s_y_1s;
```

misma_cantidad_de_0s_y_1s.adb

```
function misma_cantidad_de_0s_y_1s (n: in integer) return boolean is

   --- Entrada: 1 numero
   --- pre: el numero estara formado unicamente por 0s y 1s y comenzará con 1
   --- Salida: 1 booleano
   --- post: el resultado es true si el número de entrada contiene el mismo número de 1s que de 0s

   cont0s,cont1s: integer;
   emaitza: boolean;
   naux:integer;

   begin
      cont0s:=0;
      cont1s:=0;
      naux:=n;
      loop exit when naux=0;
         if naux rem 10 =0 then
            cont0s:=cont0s+1;
         else
            cont1s:=cont1s+1;
         end if;

         naux:=naux/10;
      end loop;

   if cont0s=cont1s then
      emaitza:=true;
   else
      emaitza:=false;
   end if;
```

```
        return emaitza;
end misma_cantidad_de_0s_y_1s;
```

2º ejercicio

Ordenar_tres_números: Dados tres números enteros positivos implementar un subprograma que ordene los tres números de menor a mayor. Utilizar para ellos ordenar_dos_numeros.adb, intercambiar.adb y prueba_ordenar_tres_numeros.adb. Primero leeré atentamente la especificación del (de los) subprograma(s) que me pidan implementar. Primera pregunta que os deberíais hacer, ¿el subprograma que he de hacer debe obtener/calcular una sola cosa o varias? Es decir, ¿será una función o un procedimiento?

prueba_ordenar_tres_numeros.adb

```
-- Author: EUITI
with ADA.Text_IO,ADA.Integer_Text_IO;
use ADA.Text_IO,ADA.Integer_Text_IO;

with ordenar_tres_numeros;

procedure prueba_ordenar_tres_numeros is
n1,n2,n3: integer;

begin
   put_line("Primera prueba: n1=5, n2=3 y n3=1");
   n1:=5;
   n2:=3;
   n3:=1;
   put_line("el orden debería ser 1 3 5");
   put_line("Y tu programa dice:     ");
   ordenar_tres_numeros(n1,n2,n3);
   put(n1);
   put(n2);
   put(n3);

   new_line;
   put_line("Primera prueba: n1=5, n2=3 y n3=1");
   n1:=5;
   n2:=3;
   n3:=1;
   put_line("el orden debería ser 1 3 5");
```

```
put_line("Y tu programa dice:     ");
ordenar_tres_numeros(n1,n2,n3);
put(n1);
put(n2);
put(n3);

new_line;
put_line("Segunda prueba: n1=5, n2=1 y n3=3");
n1:=5;
n2:=1;
n3:=3;
put_line("el orden debería ser 1 3 5");
put_line("Y tu programa dice:     ");
ordenar_tres_numeros(n1,n2,n3);
put(n1);
put(n2);
put(n3);

new_line;
put_line("Tercera prueba: n1=3, n2=5 y n3=1");
n1:=3;
n2:=5;
n3:=1;
put_line("el orden debería ser 1 3 5");
put_line("Y tu programa dice:     ");
ordenar_tres_numeros(n1,n2,n3);
put(n1);
put(n2);
put(n3);

new_line;
put_line("Cuarta prueba: n1=3, n2=1 y n3=5");
n1:=3;
n2:=1;
n3:=5;
put_line("el orden debería ser 1 3 5");
```

```
   put_line("Y tu programa dice:     ");
   ordenar_tres_numeros(n1,n2,n3);
   put(n1);
   put(n2);
   put(n3);

   new_line;
   put_line("Quinta prueba: n1=1, n2=3 y n3=5");
   n1:=1;
   n2:=3;
   n3:=5;
   put_line("el orden debería ser 1 3 5");
   put_line("Y tu programa dice:     ");
   ordenar_tres_numeros(n1,n2,n3);
   put(n1);
   put(n2);
   put(n3);

   new_line;
   put_line("Sexta prueba: n1=1, n2=5 y n3=3");
   n1:=1;
   n2:=5;
   n3:=3;
   put_line("el orden debería ser 1 3 5");
   put_line("Y tu programa dice:     ");
   ordenar_tres_numeros(n1,n2,n3);
   put(n1);
   put(n2);
   put(n3);

end prueba_ordenar_tres_numeros;
```

ordenar_tres_numeros.adb

```
-- Author: EUITI
with ordenar_dos_numeros;
procedure ordenar_tres_numeros(n1,n2,n3: in out integer) is
---entrada: 3 numeros
   --Pre: no seran iguales entre si
---salida: 3 numeros
  ---Post: estaran ordenados de menor a mayor
begin
   ordenar_dos_numeros(n1,n2);
   ordenar_dos_numeros(n1,n3);
   ordenar_dos_numeros(n2,n3);
end ordenar_tres_numeros;
```

ordenar_dos_numeros.adb

```
-- Author: Aitziber
with intercambiar;
procedure ordenar_dos_numeros(n1,n2: in out integer) is

begin
   intercambiar(n1, n2);
end ordenar_dos_numeros;
```

intercambiar.adb

```
-- Author: Aitziber
procedure intercambiar(n1,n2: in out integer) is
   aux: integer;
begin

   if (n1 > n2) then
      aux:=n1;
      n1:=n2;
      n2:=aux;
   end if;

end intercambiar;
```

3º ejercicio

Numero_primo: Dado un número entero positivo implementar un subprograma que indique si dicho número es primo o no. Utilizad es_primo.adb y prueba_es_primo.adb. Primero leeré atentamente la especificación del (de los) subprograma(s) que me pidan implementar. Primera pregunta que os deberíais hacer, ¿el subprograma que he de hacer debe obtener/calcular una sola cosa o varias? Es decir, ¿será una función o un procedimiento?

Prueba_Primo.adb

```
with Ada.Text_Io;
use Ada.Text_Io;

with Es_Primo;

procedure Prueba_Primo is
   -- este programa hace llamadas a la funcion es_primo y es util
   -- para comprobar si su funcionamiento es correcto

   package Boolean_E_S is new Enumeration_Io(Boolean);
   use Boolean_E_S;
   -- esto sirve para leer y escribir valores de tipo Boolean

begin
   Put("Primera prueba: es_primo(1) debe ser false y el resultado es ");
   Put(Es_Primo(1));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;
   -- esta instrucción lee un caracter "return", es decir, sirve para
   -- esperar a que el usuario pulse return

   Put("Segunda prueba: es_primo(2) debe ser true y el resultado es ");
   Put(Es_Primo(2));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;
```

```
   Put("Tercera prueba: es_primo(3) debe ser true y el resultado es ");
   Put(Es_Primo(3));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;

   Put("Cuarta prueba: es_primo(49) debe ser false y el resultado es ");
   Put(Es_Primo(49));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;



   Put("Quinta prueba: es_primo(137) debe ser true y el resultado es ");
   Put(Es_Primo(137));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;

end Prueba_Primo;
```

es_primo.adb

```
with Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_IO, Ada.Integer_text_IO;



function es_primo (N: in integer) return boolean is
--Especificación
--Entrada: Un número entero
--Pre: el valor_entrada del número >=1
--Salida: un booleano
--Post: valor_salida1 será true si valor_entrada es primo y false si valor_entrada no es
primo

   posible_Divisor:Integer;
   primo:boolean;
```

```
begin
    Primo:=True;
    posible_divisor:=2;
    if (n = 1) then
        Primo:=False;
    end if;

    loop exit when  primo=false or posible_divisor=n;
       if n rem posible_divisor=0 then primo:=false;
       else
        posible_divisor:=posible_divisor+1;
       end if;

   end loop;

   return(primo);
end es_primo;
```

4º ejercicio

Numero_capicua: Dado un número entero positivo implementar un subprograma que indique si dicho número es capicúa o no. Utilizad es_capicua.adb y prueba_es_capicua.adb. Primero leeré atentamente la especificación del (de los) subprograma(s) que me pidan implementar. Primera pregunta que os deberíais hacer, ¿el subprograma que he de hacer debe obtener/calcular una sola cosa o varias? Es decir, ¿será una función o un procedimiento?

prueba_es_capicua.adb

```
with Ada.Text_Io;
use Ada.Text_Io;

with es_capicua;

procedure Prueba_es_capicua is
   -- este programa hace llamadas a la funcion capicua y es util
   -- para comprobar si su funcionamiento es correcto

   package Boolean_E_S is new Enumeration_Io(Boolean);
   use Boolean_E_S;
   -- esto sirve para leer y escribir valores de tipo Boolean

begin
   Put("Primera prueba: capicua(1) debe ser true y el resultado es ");
   Put(es_capicua(1));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;
   -- esta instrucción lee un caracter "return", es decir, sirve para
   -- esperar a que el usuario pulse return

   Put("Segunda prueba: capicua(232) debe ser true y el resultado es ");
   Put(es_capicua(232));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;
```

```
   Put("Tercera prueba: capicua(1234554321) debe ser true y el resultado es ");
   Put(es_capicua(1234554321));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;

   Put("Cuarta prueba: capicua(1234567) debe ser false y el resultado es ");
   Put(es_capicua(1234567));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;



   Put("Quinta prueba: capicua(12321) debe ser true y el resultado es ");
   Put(es_capicua(12321));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;

end Prueba_es_capicua;
```

es_capicua.adb

```
with inverso;

function es_capicua (n: in integer) return boolean is
   -- pre: N >= 1
   -- post: el resultado True si N es capicua y false si no

   N_Inverso: integer;
   capicua: boolean;



begin
    capicua:=False;
    N_Inverso:=inverso(n);
```

```
   if(n - N_Inverso = 0) then
      capicua:=True;
   end if;

   return(capicua);
end es_capicua;
```

inverso.adb

```
-- Author: EUITI

function inverso(n: in integer) return integer is
   n_aux,n_inverso,resto:integer;
begin
   n_aux:=n;
   n_inverso:=0;
   loop exit when n_aux=0;
    resto:=n_aux rem 10;
    n_inverso:=n_inverso*10+resto;
    n_aux:=n_aux/10;
   end loop;


   return n_inverso;
end inverso;
```

5º ejercicio

Numero_Omirp: Dado un número entero positivo implementar un subprograma que calcule el primer número capicúa y primo a partir del inicial. Utilizar es_omirp.adb y siguiente_omirp.adb y prueba_siguiente_omirp.adb. Primero leeré atentamente la especificación del (de los) subprograma(s) que me pidan implementar. Primera pregunta que os deberíais hacer, ¿el subprograma que he de hacer debe obtener/calcular una sola cosa o varias? Es decir, ¿será una función o un procedimiento?

prueba_siguiente_omirp.adb

```
-- Author: EUITI
with Ada.Text_IO,Ada.Integer_Text_IO;
use Ada.Text_IO,Ada.Integer_Text_IO;

with siguiente_omirp;

procedure prueba_siguiente_omirp is
--algunos ejempos de numeros omirp
---  107/701 - 113/311 -
---  149/941 - 157/751 - 167/761 - 179/971
---  199/991 - 337/733 - 347/743 - 359/953 - 389/983
---  709/907 - 739/937 - 769/967
num_omirp:integer;
zen1:integer;

begin
   put_line("Sartu zenbaki oso bat, eta programa honek hurrengo zenbaki omirp topatuko du: ");
   get(zen1);
   num_omirp:=siguiente_omirp(zen1);
   put("Hurrengo zenbaki omirp: ");
   put(num_omirp);
   put_line(" da.");
end prueba_siguiente_omirp;
```

siguiente_omirp.adb

```
-- Author: Aitziber
with es_omirp;
function siguiente_omirp(n: in integer) return integer is
n_aux:integer;
begin
   n_aux:=n+1;
   loop exit when es_omirp(n_aux)=true;
      n_aux:=n_aux+1;
   end loop;
  return n_aux;
end siguiente_omirp;
```

es_omirp.adb

```
-- Author: EUITI
with es_primo;
with inverso;

function es_omirp (n: in integer) return boolean is
   -- pre: N >= 1
   -- post: el resultado True si N es capicua y false si no

   N_Inverso: integer;
   omirp:boolean:=false;
   begin

   if es_primo(n) then

      N_Inverso:=inverso(n);
      if N_inverso/=n then
         if es_primo(n_inverso) then
            omirp:=true;
         end if;
      end if;

   end if;
```

```
   return omirp;
end es_omirp;
```

es_primo.adb

```
with Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_IO, Ada.Integer_text_IO;



function es_primo (N: in integer) return boolean is
--Especificación
--Entrada: Un número entero
--Pre: el valor_entrada del número >=1
--Salida: un booleano
--Post: valor_salida1 será true si valor_entrada es primo y false si valor_entrada no es
primo

  posible_Divisor:Integer;
  primo:boolean;

begin
   Primo:=True;
   posible_divisor:=2;
   if (n = 1) then
      Primo:=False;
   end if;

   loop exit when  primo=false or posible_divisor=n;
     if n rem posible_divisor=0 then primo:=false;
     else
      posible_divisor:=posible_divisor+1;
     end if;

  end loop;

  return(primo);
```

```
end es_primo;
```

inverso.adb

```
-- Author: EUITI

function inverso(n: in integer) return integer is
   n_aux,n_inverso,resto:integer;
begin
   n_aux:=n;
   n_inverso:=0;
   loop exit when n_aux=0;
    resto:=n_aux rem 10;
    n_inverso:=n_inverso*10+resto;
    n_aux:=n_aux/10;
   end loop;


   return n_inverso;
end inverso;
```

6º ejercicio

Calcular el día anterior: Dada una fecha, calcular cuál es el día anterior. Para ello se precisa implementar los siguientes subprogramas: ultimo_dia.adb y es_bisiesto.adb. (Se tendrán en cuenta los días bisiestos. Utilizar las plantillas de ultimo_dia.adb y es_bisiesto.adb, y día_anterior.adb y utilizar para probar el código prueba_dia_anterior.adb. Donde se encontrarán todos los casos de prueba.

prueba_dia_anterior.adb

```
with ADA.Text_IO,ADA.Integer_Text_IO,dia_anterior;
use ADA.Text_IO,ADA.Integer_Text_IO;

procedure prueba_dia_anterior is



dia,mes,anno:integer;

begin

   put("Primer caso de prueba: 1 1 2010");
   new_line;
   put("el resultado deberia ser 31 12 2009");
   new_line;
   put("y según tu programa el resultado es:");
   new_line;
   dia:=1;
   mes:=1;
   anno:=2010;
   ---inicializaciones de dia mes y anno con el caso de prueba
   ---llamada a tu subprograma
   dia_anterior(dia, mes, anno);
   put(dia,width=>3);
   put(mes,width=>3);
   put(anno,width=>5);
   new_line;
   new_line;
```

```
put("Segundo caso de prueba: 1 4 2010");
new_line;
put("el resultado deberia ser 31 3 2010");
new_line;
put("y según tu programa el resultado es:");
new_line;
dia:=1;
mes:=4;
anno:=2010;
---inicializaciones de dia mes y anno con el caso de prueba
---llamada a tu subprograma
dia_anterior(dia, mes, anno);
put(dia,width=>3);
put(mes,width=>3);
put(anno,width=>5);
new_line;
new_line;

put("Tercer caso de prueba: 1 10 2010");
new_line;
put("el resultado deberia ser 30 9 2010");
new_line;
put("y según tu programa el resultado es:");
new_line;
dia:=1;
mes:=10;
anno:=2010;

---inicializaciones de dia mes y anno con el caso de prueba
---llamada a tu subprograma
dia_anterior(dia, mes, anno);
put(dia,width=>3);
put(mes,width=>3);
put(anno,width=>5);
new_line;
new_line;
```

```
put("Cuarto caso de prueba: 1 3 2010");
new_line;
put("el resultado deberia ser 28 2 2010");
new_line;
put("y según tu programa el resultado es:");
new_line;
dia:=1;
mes:=3;
anno:=2010;

---inicializaciones de dia mes y anno con el caso de prueba
---llamada a tu subprograma
dia_anterior(dia, mes, anno);
put(dia,width=>3);
put(mes,width=>3);
put(anno,width=>5);
new_line;
new_line;

put("Quinto caso de prueba: 1 3 2004");
new_line;
put("el resultado deberia ser 29 2 2004");
new_line;
put("y según tu programa el resultado es:");
new_line;
dia:=1;
mes:=3;
anno:=2004;

---inicializaciones de dia mes y anno con el caso de prueba
---llamada a tu subprograma
dia_anterior(dia, mes, anno);
put(dia,width=>3);
put(mes,width=>3);
put(anno,width=>5);
```

```
   new_line;
   new_line;

   put("Sexto caso de prueba: 3 1 2010");
   new_line;
   put("el resultado deberia ser 2 1 2010");
   new_line;
   put("y según tu programa el resultado es:");
   new_line;
   dia:=3;
   mes:=1;
   anno:=2010;

   ---inicializaciones de dia mes y anno con el caso de prueba
   ---llamada a tu subprograma
   dia_anterior(dia, mes, anno);
   put(dia,width=>3);
   put(mes,width=>3);
   put(anno,width=>5);
   new_line;
   new_line;

end prueba_dia_anterior;
```

dia_anterior.adb

```
with ADA.Text_IO,ADA.Integer_Text_IO,ultimo_dia;
use ADA.Text_IO,ADA.Integer_Text_IO;

procedure dia_anterior (N1, N2, N3: in out integer) is

  --- N1 = Eguna | N2 = Hilabetea | N3 = Urtea
  --- Entrada: 3 numeros
  --- pre:  1<= dia:valor1 <=30 |mes:valor2 {04,06,09,11}
  ---  1<= dia:valor1 <=31 |mes:valor2 {01,03,05,07,08,10,12}
  ---  1<= dia:valor1 <=29 |mes:valor2=01 ^ año bisiesto
  ---  1<= dia:valor1 <=28 |mes:valor2=01 ^ año no bisiesto
```

```
  ---  <=1 mes:valor2 <=12
  ---  1<= anno:valor3 <=2020
  --- Salida: 3 numeros
  --- post: dia será el dia anterior
  ---  1<= dia:valor1 <=30 |mes:valor2 {04,06,09,11}
  ---  1<= dia:valor1 <=31 |mes:valor2 {01,03,05,07,08,10,12}
  ---  1<= dia:valor1 <=29 |mes:valor2=01 ^ año bisiesto
  ---  1<= dia:valor1 <=28 |mes:valor2=01 ^ año no bisiesto
  ---  <=1 mes:valor2 <=12
  ---  1<= anno:valor3 <=2020

   azken_eguna:integer;


begin

   N1:=N1 - 1;

   if(N1 < 1) then
      N2:=N2 - 1;
      azken_eguna:=ultimo_dia(N2, N3);
      N1:=azken_eguna;
   end if;
   if(N2 < 1) then
      N3:= N3 - 1;
      azken_eguna:=ultimo_dia(N2, N3);
      N2:=12;
   end if;

end dia_anterior;
```

ultimo_dia.adb

```
with ADA.Text_IO,ADA.Integer_Text_IO,es_bisiesto;
use ADA.Text_IO,ADA.Integer_Text_IO;

function ultimo_dia (N1, N2: in integer) return integer is

  --- Entrada: 2 numeros   N1= hilabetea / N2= urtea
  --- pre:1 mes:valor1 <=12
  ---     1<= anno:valor2 <=2020
  --- Salida: 1 numero
  --- post: dia:valor3
  ---  1<= ult_dia:valor1 <=30 |mes:valor2 {04,06,09,11}
  ---  1<= ult_dia:valor1 <=31 |mes:valor2 {01,03,05,07,08,10,12}
  ---  1<= ult_dia:valor1 <=29 |mes:valor2=01 ^ año bisiesto
  ---  1<= ult_dia:valor1 <=28 |mes:valor2=01 ^ año no bisiesto

  ult_dia:integer;

begin
   if(es_bisiesto(N2) = true) then
      if(N1 = 2) then
         ult_dia:=29;
      end if;
   else
      ult_dia:=28;
   end if;

   if((N1 = 1) or (N1 = 3) or (N1 = 5) or (N1 = 7) or (N1 = 8) or (N1 = 10) or (N1 = 12))
then
      ult_dia:=31;
   end if;
   if((N1 = 4) or (N1 = 6) or (N1 = 9) or (N1 = 11)) then
      ult_dia:=30;
   end if;

   return(ult_dia);
```

```
end ultimo_dia;
```

es_bisiesto.adb

```
function es_bisiesto (N: in integer) return boolean is

   --- Entrada: 1 numero
   --- pre: 1<= anno:valor1 <=2020
   --- post: el resultado es true si el anno es bisiesto |
   --- bisiesto (((divisible entre 4)y(no divisible entre 100))o (divisible entre 400))

   result_bisies: boolean:=false;

begin

    if( ( (N rem 4 = 0) and not( N rem 10 = 0) ) or (N rem 400 = 0) ) then
        result_bisies:= True;
    end if;

    return(result_bisies);
end es_bisiesto;
```

7º ejercicio

Número medio: Dado un entero, decir si es medio o no. Utilizar para ello los ficheros es_medio.adb, suma_de_inferiores.adb, suma_de_superiores.adb y prueba_es_medio.adb.

Un número es medio si: La suma de los números inferiores a él es igual a la suma de números superiores a él.

prueba_es_medio.adb

```
-- Author: Aitziber
with Ada.Text_IO,Ada.Integer_Text_IO;
use Ada.Text_IO,Ada.Integer_Text_IO;

with es_medio;

procedure prueba_es_medio is

   package Boolean_E_S is new Enumeration_Io(Boolean);
   use Boolean_E_S;
   -- esto sirve para leer y escribir valores de tipo Boolean

begin
        put_line("Primera prueba: es_medio(6) debería devolver true y devuelve: ");
        put(es_medio(6));

        put_line("Segunda prueba: es_medio(8) debería devolver false y devuelve: ");
        put(es_medio(8));

end prueba_es_medio;
```

es_medio.adb

```
-- Author: Aitziber
with suma_superiores, suma_inferiores;

function es_medio(n: in integer) return boolean is
n_inf,n_sup: integer;
result: boolean;
begin
   n_inf:=suma_inferiores(n);
   n_sup:=suma_superiores(n);
```

```
    if n_inf = n_sup then
        result:= TRUE;
    else
        result:= FALSE;
    end if;
    return result;
end es_medio;
```

suma_superiores.adb

```
-- Author: Aitziber
with suma_inferiores;

function suma_superiores(n_sup: in integer) return integer is
   n,n_inf:integer;
begin
   n:=n_sup+1;
   n_inf:=suma_inferiores(n_sup);
   loop exit when ( n_inf <= n);
        n:=(n+(n+1));
   end loop;
    return n;
end suma_superiores;
```

suma_inferiores.adb

```
-- Author: Aitziber
function suma_inferiores(n_inf: in integer) return integer is
    kont,n:integer;

begin
    kont:=n_inf-1;
    n:=n_inf-1;
    loop exit when kont=0;
        kont:=kont-1;
        n:=n+kont;
    end loop;
    return n;
end suma_inferiores;
```

8º ejercicio

Número Narcisista: Número de n dígitos que resulta ser igual a la suma de las potencias de orden n de sus dígitos.

Ejemplo: 153 = $1^3 + 5^3 + 3^3$ (información obtenida de la wikipedia). Implementar un subprograma que imprima los 3 primeros números narcisistas de 3 cifras, el primero será el 153. Para ello se probará con cada número de entre 153 y 999 si el número en cuestión es o no narcisista (implementaremos un subprograma es_narcisista.adb). Para determinar si un número es narcisista, se irá probando a elevar sus cifras a las distintas potencias comenzando por la unidad, hasta que la suma sea superior al número en cuestión. Utilizar las plantillas números_narcisistas_de_tres_cifras.adb, es_narcisista.adb y cifras_elevadas.adb, y añadid cuantos subprogramas sean necesarios. Hay dos programas de prueba el primero prueba_números_narcisistas_de_tres_cifras.adb consistirá en llamar a tu subprograma números_narcisistas_de_tres_cifras.adb y te dirá cuales son los números narcisistas de tres cifras que tu subprograma debería obtener. El segundo prueba_narcisista.adb comprobará si el código de es_narcisista.adb funciona correctamente.

prueba_es_narcisista.adb

```
with Ada.Text_Io;
use Ada.Text_Io;

with es_narcisista;

procedure prueba_es_narcisista is
   -- este programa hace llamadas a la funcion capicua y es util
   -- para comprobar si su funcionamiento es correcto

   package Boolean_E_S is new Enumeration_Io(Boolean);
   use Boolean_E_S;
   -- esto sirve para leer y escribir valores de tipo Boolean

begin
   Put("Primera prueba: es_narcisista(101) debe ser false y el resultado es ");
   Put(es_narcisista(101));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;
   -- esta instrucción lee un caracter "return", es decir, sirve para
   -- esperar a que el usuario pulse return
```

```
   Put("Segunda prueba: es_narcisista(153) debe ser true y el resultado es ");
   Put(es_narcisista(153));
   New_Line(3);
   Put_Line("Pulsa return para continuar");
   Skip_Line;

end prueba_es_narcisista;
```

es_narcisita;

```
-- Author: Aitziber
with ADA.Text_IO, ADA.Integer_Text_IO;
use ADA.Text_IO, ADA.Integer_Text_IO;

with ciras_elevadas;

function es_narcisista(n: in integer) return boolean is
--pre: el numero consta de 3 cifras
--post: devolvera true si el numero es narcisista, es decir, si la suma
   ---de sus cifras elevadas a un numero 0< n <10 es igual al valor de entrada. Por ejemplo
el 153 =(1**3)+(5**3)+(3**3)
   n_sum:integer;
   result:boolean:=FALSE;
begin
   n_sum:=0;
   ciras_elevadas(n,n_sum);
   if n = n_sum then
      result:=TRUE;
   else
      result:=FALSE;
   end if;
   return result;
end es_narcisista;
```

ciras_elevadas.adb

```
-- Author: Aitziber
with obtener_tres_cifras;
```

```
procedure ciras_elevadas(n:in integer; n_sum: out integer) is
---Entrada:el numero en cuestion y la potencia a la que hay que elevar las cifras
---Pre: el numero contara con 3 cifras y la potencia será un numero natural >0 y <10
---Salida: un numero entero
---Post: devolverá la suma de las cifras elevadas a la potencia.

   n1,n2,n3:integer;
begin
    n1:=0;
    n2:=0;
    n3:=0;
    obtener_tres_cifras(n,n1,n2,n3);
     --loop exit when n <= n_sum;
    n_sum:=n1**3+n2**3+n3**3;
       -- kont:=kont+1;
     --end loop;
end ciras_elevadas;
```

obtener_tres_cifras.adb

```
procedure obtener_tres_cifras(n: in integer; n1,n2,n3: out integer) is
---pre:-
---post: imprimirá por pantalla aquellos numeros de 3 cifras que sean narcisistas
begin
  n1:=n/100;
  n2:=n/10 - (n/100 * 10);
  n3:=n - (n/10 * 10);
end obtener_tres_cifras;
```

9º ejercicio

Hallar los factores primos. Completar el programa principal_calcular_suma_de_factores_primos.adb tal que vaya pidiendo repetidamente números pares mayores que 2 al usuario, hasta que este introduzca un 0. En el programa se deberá comprobar que los números que introduce el usuario cumplen la condición de ser pares > 2, y de no ser así se le volverá a pedir al usuario que introduzca otro numero hasta que este cumpla la condición o hasta que el usuario decida salir del programa introduciendo un 0. Por cada numero par mayor que 2, se obtendrá todos los pares de números primos tal que la suma sea igual al numero par inicial.

principal_calcular_factores_primos.adb

```
with Pedir_Y_Comprobar_Dato,Calcular_Factores_Primos;
with Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_Io, Ada.Integer_Text_Io;

procedure principal_calcular_factores_primos is
   -- Entrada: un valor entero
   -- Precondicion: el valor es un numero binario (formado por ceros y unos)
   -- Salida: un valor entero
   -- Postcondicion:
   --   el resultado es el valor decimal correspondiente al numero binario inicial

   zen: Integer;

begin
   Pedir_Y_Comprobar_Dato(zen);
   loop exit when zen=0;
      Calcular_Factores_Primos(zen);
      Pedir_Y_Comprobar_Dato(zen);
   end loop;
end principal_calcular_factores_primos;
```

pedir_y_comprobar_dato.adb

```
with Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_Io, Ada.Integer_Text_Io;

procedure pedir_y_comprobar_dato (N: out integer) is
```

```
begin

   put("Introduce un numero mayor que 2 o cero para terminar: ");
   Get(N);
   loop exit when N > 2 or N=0;
      put("Valor incorrecto, introduce un numero mayor que 2: ");
      Get(N);
   end loop;

   New_Line;
end pedir_y_comprobar_dato;
```

calcular_factores_primos.adb

```
with pedir_y_comprobar_dato,calcular_primeros_factores_primos,
calcular_siguientes_factores_primos,Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_Io, Ada.Integer_Text_Io;

procedure calcular_factores_primos (zen: in integer) is
--Especificación
--Entrada: Un número entero
--Pre: el valor_entrada del número será par >2
--Salida: varios pares de números enteros
--Post: cada par cumplirá valor_salida1 y valor_salida2 serán y cumplirán que
valor_salida1+ valor_salida2 = valor_entrada



    Factor_primo1,factor_primo2:integer;
begin
   Calcular_Primeros_Factores_Primos(zen,Factor_Primo1,Factor_Primo2);
   loop exit when  Factor_Primo1>zen/2;
      Put("los factores son: ");
      Put(factor_primo1);
      put(factor_primo2);
      calcular_siguientes_factores_primos(zen,Factor_Primo1,Factor_Primo2);
```

```
      New_Line;
   end loop;
end calcular_factores_primos;
```

calcular_primeros_factores_primos.adb

```
with Es_Primo,Siguiente_Numero_Primo;


procedure calcular_primeros_factores_primos (zen: in integer; Primo1,Primo2:out integer)
is
--Especificacion
--Entrada: 1 numero entero
--Pre: el valor_entrada1 del numero sera par >2
--valor_entrada2 sera primo
--Salida: 2 numeros enteros
--Post: valor_salida1 y valor_salida2 seran y cumpliran que valor_salida1+ valor_salida2 =
valor_entrada y valor_salida1 > valor_entrada2



begin
   Primo1:=2;
   Primo2:=zen-Primo1;
   loop exit when es_primo(primo2)=true;
      Primo1:=Siguiente_Numero_Primo(Primo1);
      Primo2:=zen-Primo1;


   end loop;
end calcular_primeros_factores_primos;
```

siguiente_numero_primo.adb

```
with es_primo;

with Ada.integer_text_IO;
use Ada.Integer_Text_Io;

function siguiente_numero_primo (N: in integer) return integer is
--Especificación
```

--Entrada: Un número entero

--Pre: el valor_entrada del número será >0

--Salida: 1 números enteros

--Post: valor_salida será el siguiente valor primo tal que valor_salida > valor_entrada y el rango de números entre [valor_entrada .. valor_salida] no contiene otro número primo

```
    siguiente_primo:Integer;
    primo:boolean:=false;
begin
   siguiente_primo:=n+1;

   loop exit when   primo=true;
       if es_primo(siguiente_primo)=true then
          primo:=true;
       else
          siguiente_primo:=siguiente_primo+1;
       end if;

   end loop;
   return siguiente_primo;

end siguiente_numero_primo;
```

calcular_siguientes_factores_primos.adb

```
with Es_Primo,Siguiente_Numero_Primo;
```

procedure calcular_siguientes_factores_primos (N:in integer; Primo1: in out integer; Primo2:out integer) is

--Especificación

--Entrada: 2 números enteros

--Pre: el valor_entrada1 del número será par >2

--valor_entrada2 será primo

--Salida: 2 números enteros

--Post: valor_salida1 y valor_salida2 serán y cumplirán que valor_salida1+ valor_salida2 = valor_entrada y valor_salida1 > valor_entrada2

```
begin
   primo1:=Siguiente_Numero_Primo(Primo1);
   Primo2:=N-Primo1;
   loop exit when  es_primo(primo2)=true;
      Primo1:=Siguiente_Numero_Primo(Primo1);
      Primo2:=N-Primo1;
   end loop;
   end calcular_siguientes_factores_primos;
```

es_primo.adb

```
with Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_IO, Ada.Integer_text_IO;



function es_primo (N: in integer) return boolean is
--Especificación
--Entrada: Un número entero
--Pre: el valor_entrada del número >=1
--Salida: un booleano
--Post: valor_salida1 será true si valor_entrada es primo y false si valor_entrada no es
primo

  posible_Divisor:Integer;
  primo:boolean;


begin
   Primo:=True;
   posible_divisor:=2;
   if (n = 1) then
      Primo:=False;
   end if;

   loop exit when  primo=false or posible_divisor=n;
     if n rem posible_divisor=0 then primo:=false;
     else
```

```
        posible_divisor:=posible_divisor+1;
       end if;

   end loop;

   return(primo);
end es_primo;
```