# Prog. Básica - Laboratorio 5 Subprogramas

**Nombre**:_____**Fecha**:_____

Nota: es muy posible que en los programas de pruebas que se facilite haya que añadir más casos de prueba. Puede que haya algún subprograma sin especificar (precondición, postcondición) en ese caso deberéis escribir la especificación vosotros mismos.

## 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?

## 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?

## 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?

## 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?

## 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?

## 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.

# 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 el es igual a la suma de números superiores a el.

Ejemplos:
6 es número medio
la suma de los números inferiores a el es
1+2+3+4+5=15
la suma números superiores a él es
7+8=15
7 no es número medio
la suma de los números inferiores a el es
1+2+3+4+5+6=21
la suma números superiores a el es
8+9=17
8+9+10=27
(Llegados a este punto ya sabemos que no vamos a encontrar ninguna suma de números superiores a el 7 tal que el resultado sea igual que la suma de sus inferiores.)

35 es número medio
la suma de los números inferiores a él es
1+2+3+4+5+...34=595
la suma números superiores a él es
36+37+...+49=595

# 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.

Algoritmo principal:
num:integer;
comienzo
num←100;---primer número con el que probaré

repetir salir si num>999 ---este loop me permitirá ir probando con los números del 100 al 999
si(es_narcisista(num)) entonces
escribir(num);
escribir(“es narcisista”);
fin_si;
num←num+1;
fin_repetir;

fin;

subprograma es_narcisista:

```
función/procedimiento??? es_narcisista(n: in/out??? Integer) ……..
--Especificación:
--Entrada: un numero entero
--Pre: el valor  >=100
--Salida: booleano
--Post: devolverá true si el número de entrada es narcisista, es decir, si la suma de sus cifras
      --elevado a un número n da como resultado el número en cuestión.

potencia: integer;
no_posible:boolean;
comienzo
no_posible←false;---indicará si no es posible obtener una potencia tal que las cifras del número en cuestión
                        ---elevadas a esa potencia y sumadas entre sí den como resultado el número.
          potencia←1;
repetir salir si no_posible=true o encontrado;

                    ¿?????
          fin_repetir;
¿????
fin_es_narcisista;

función/procedimiento??? cifras_elevadas(n: in/out??? integer; expon: in/out integer;¿????) …
Especificación: Deberéis rellenarla vosotros.

rdo: integer←0;
n_aux:integer←n;

comienzo

repetir salir si ¿????;
                    --- bucle que  permitirá ir obteniendo la suma de las cifras del número elevadas a expon
          fin_repetir;
¿????
fin_ cifras_elevadas;
```

# 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. Ejemplos:
4: 2+2
6. 3+3
8: 3+5
22: 3+19,  5+17,  11+11
24:  5+19,  7+17,  11+13

¿Como obtenemos los factores por ejemplo del 24?

Primero comenzaremos probando con el 2 por ser este el primer número primo. El complementario al 2 para formar el 24 es 24-2, el 22, comprobamos si es primo, y no lo es así que probamos otra combinación. Probamos con el siguiente primo, el 3, esta vez el complementario es el 21, dado que 24-3=21. Comprobamos si 21 es primo, como no lo es, pasamos al siguiente primo, en este caso el 5. El complementario para formar el 24 es el 19 (24-5=19). Comprobamos si es primo y lo es. Así encontramos la primera combinación de factores primos. Ahora pasaremos a obtener la siguiente probando con el siguiente número primo…..

El algoritmo correspondiente a principal_calcular_factores_primos.adb será:

```
con pedir_número ,calcular_factores_primos; -- para decir que vamos a hacer uso de estos
-- subprogramas
num,: integer;
pedir_numero(num);
repetir salir si num=0;
        calcular_factores_primos(num);
        pedir_número(num);
fin_repetir;
```

El algoritmo de calcular_factores_primos podría ser dado un número num:

```
con calcular_primeros_factores_primos,calcular_siguientes_factores_primos;
-- para decir que vamos a hacer uso de estos
-- subprogramas
con Ada.Text_Io, Ada.Integer_Text_Io;
utilizar Ada.Text_Io, Ada.Integer_Text_Io;Factor_Primo1_ 2; Factor_primo2:_num-
factor_primo1;

Calcular_Primeros_Factores_Primos(Num,Factor_Primo1,Factor_Primo2);
Repetir salir si Factor_Primo1> Num/2;
        escribir("los factores son: ");
        escribir(factor_primo1);
        escribir(factor_primo2);
        calcular_siguientes_factores_primos(num,factor_primo1,factor_primo2);
fin repetir;
```

Calcular_primeros_factores_primos y calcular_siguientes_factores_primos hará uso de

1) El algoritmo de calcular_siguiente_suma_primos dados un numero (num) del cual se quiere calcular la suma de primos y un numero primo (primo)
2) es_primo que dado un numero nos dirá si el numero es primo o no.
3) siguiente_numero_primo que dado un número primo nos dirá cual es el siguiente número primo.
Como se ve en la implementación del programa puede ser interesante utilizar los siguientes subprogramas:

a) pedir_número: Pedirá un número al usuario y seguirá haciéndolo hasta que el número sea par > 2 o 0

Especificación
Entrada: -
Pre: -
Salida: 1 número entero
Post: valor_salida1 sera par >2 o 0

b) calcular_factores_primos: Dado un número calculara todas las posibles combinaciones de números primos tal que la suma de estos números sea igual al número dado inicialmente, llamando inicialmente a calcular_primera_suma_de_primos y a calcular_siguiente_suma_primos sucesivamente hasta que ya no existan mas parejas de primos que sumen el número inicial

Especificación
Entrada: Un numero entero
Pre: el valor_entrada del numero sera par >2
Salida: varios pares de numeros enteros
Post: cada par cumplira valor_salida1 y valor_salida2 seran y cumplirán que valor_salida1+ valor_salida2 = valor_entrada

c) calcular_primera_suma_primos: Dado un número y un par de números primos cuya suma lo forman, calculara el siguiente par números primos tal que la suma de estos números sea igual al numero dado inicialmente

Especificación

Entrada: 2 números enteros
Pre: el valor_entrada1 del numero será par >2 valor_entrada2 será 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

d) calcular_siguiente_suma_primos: Dado un número y un par de números primos cuya suma lo forman, calculara el siguiente par números primos tal que la suma de estos números sea igual al numero dado inicialmente
Especificación
Entrada: 2 numeros enteros
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

e) siguiente_número_primo: Dado un numero calculara el siguiente numero primo
Especificación
Entrada: Un numero entero
Pre: el valor_entrada del numero sera >0
Salida: 1 numeros enteros
Post: valor_salida sera el siguiente valor primo tal que valor_salida > valor_entrada y el rango de numeros entre [valor_entrada .. valor_salida] no contiene otro numero primo

f) es_primo:
Especificación
Entrada: Un numero entero
Pre: el valor_entrada del numero >1
Salida: un booleano
Post: valor true si el numero es primo (Cuidado el numero 1 no es primo) sino false