# Prog. Básica - Laboratorio 1 Introducción a la algorítmica

**Nombre:** Oihan Irastorza Carrasco **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.

**1. Diferencia.** Crea un algoritmo que, dados dos números enteros num1 y num2, calcule su diferencia.

**2. Producto.** Crea un algoritmo que pida al usuario dos números enteros, calcule su producto y lo muestre en pantalla.

**3. La edad aparente.** Crea un algoritmo que, dado un número natural que indica la edad de una persona, calcule su edad aparente. La edad aparente de una persona se calcula obteniendo el doble de su edad aumentado en cuatro.

**4. Área del círculo.** Crea un algoritmo que, dada la longitud de un segmento, calcule el área de un círculo con un radio de dicha longitud.

**5. Es\_par.** Crea un algoritmo que, dado un número natural, indique si es par (true) o impar (false).

**6. Ordenar dos números (I).** Crea un algoritmo que, dados dos valores enteros mayores que 0, num1 y num2, los ordene de menor a mayor de manera que num1 contenga el menor y num2 el mayor (utiliza como base el algoritmo de calcular el máximo –Máximo1– que se explicó en la clase de teoría)

**7. Ordenar dos números (reusando el ejercicio 1.6).** Crea un algoritmo que pida al usuario dos valores enteros mayores que 0 los ordene de menor a mayor usando el algoritmo del ejercicio 3 y los presente por pantalla en ese orden (utiliza como base el algoritmo de calcular el máximo –Máximo2– que se explicó en la clase de teoría).

**8. Ordenar tres números (I).** Crea un algoritmo que pida al usuario tres valores enteros mayores que 0, num1, num2 y num3, los ordene de menor a mayor y los presente por pantalla en ese orden.

1. **Diferencia**

Crea un algoritmo que dados dos números enteros num1 y num2, calcule su diferencia.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? No

Después, rellena la especificación:

|  |
| --- |
| Datos: 2 parámetros, num1 y num2, ambos in integer.  Pre: los números tienen que ser enteros.    Resultado: se retorna la diferencia entre ambos números (diferencia: integer).  Post: resta max(num1, num2) - min(num1, num2) (la diferencia entre ambos). |

* 1. Casos de Prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Num1 | Num2 | Descripción | Diferencia |
| 1 | 9 | 1 | Ambos números positivos, num1 mayor | 8 |
| 2 | 1 | 7 | Ambos números positivos, num2 mayor | 6 |
| 3 | -4 | 8 | Num1 negativo y num2 positivo | 12 |
| 4 | 8 | -15 | Num1 es positio y num2 negativo | 23 |
| 5 | -19 | -3 | Dos números negativos | 16 |

* 1. Algoritmo

FUNCTION diferencia\_dos\_numeros(num1, num2: in Integer) return Integer IS

-- Datos: 2 parametros, num1 y num2, ambos in integer

-- Pre: los dos numeros tienen que ser enteros

-- Post: max(num1, num2) - min(num1, num2) (la diferencia entre ambos)

-- Resultado: se retorna la diferencia entre ambos numeros (diferencia: integer)

diferencia: Integer;

BEGIN

if (num1 > num2) then

diferencia := num1 - num2;

elsif (num1 < num2) then

diferencia := num2 - num1;

else

diferencia := 0;

end if;

return diferencia;

END diferencia\_dos\_numeros;

1. **Producto**

Crea un algoritmo que pida al usuario dos números enteros, calcule su producto y lo muestre en pantalla.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? No

Después, rellena la especificación:

|  |
| --- |
| Entrada: el usuario introduce 2 integers.  Pre: 2 números introducidos por el usuario, tienen que ser enteros. Esto siempre será así porque hemos tipado las variables como integer.  Salida: se muestra por pantalla al usuario el producto de num1 y num2  Post: se multiplica un número con el otro (producto). |

* 1. Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| # | Entrada estándar (teclado) | Descripción | Salida estándar (pantalla) |
| 1 | 1 y 9 | Ambos números positivos | 9 |
| 2 | 4 y 18 | Ambos números positivos | 72 |
| 3 | 5 y -56 | Num1 positivo y num2 negativo | -280 |
| 4 | -13 y 1 | Num1 negativo y num2 positivo | -13 |
| 5 | -34 y -80 | Ambos números negativos | 2720 |

* 1. Algoritmo

WITH Ada.Text\_IO, Ada.Integer\_Text\_IO;

USE Ada.Text\_IO, Ada.Integer\_Text\_IO;

PROCEDURE producto\_dos\_numeros IS

-- Entrada: el usuario introduce 2 integers

-- Pre: 2 numeros introducidos por el usuario, tienen que ser enteros

-- Salida: se muestra por pantalla al usuario el producto de num1 y num2

-- Post: se multiplica un numero con el otro (producto)

num1: Integer;

num2: Integer;

producto: Integer;

BEGIN

put("Introduzca el primer numero: ");

get(num1);

new\_line;

put("Introduzca el segundo numero: ");

get(num2);

new\_line;

new\_line;

producto := num1 \* num2;

put("El producto es: " & Integer'Image(producto));

END producto\_dos\_numeros;

1. **La edad aparente**

Crea un algoritmo que, dado un número natural que indica la edad de una persona, calcule su edad aparente. La edad aparente de una persona se calcula obteniendo el doble de su edad aumentado en cuatro.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? No

Después, rellena la especificación:

|  |
| --- |
| Datos: 1 parámetro, edad: in Positive.  Pre: edad > 0. No es necesario si ponemos edad de tipo Positive.  Resultado: el programa retorna la edad aparente: (edad \* 2) + 4.  Post: (edad \* 2) + 4, es decir, la edad aparente. |

* 1. Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| # | Edad | Descripción | Edad\_aparente |
| 1 | 2 | Numero positivo | 8 |

* 1. Algoritmo

FUNCTION edad\_aparente(edad: in Positive) return Integer IS

-- Datos: 1 parametro, edad: in positive

-- Pre: edad > 0. Siempre sera > 0 si ponemos edad de tipo Positive

-- Resultado: el programa retorna la edad aparente: (edad \* 2) + 4

-- Post: (edad \* 2) + 4, es decir, la edad aparente

BEGIN

return (edad \* 2) + 4;

END edad\_aparente;

1. **Área del círculo**

Crea un algoritmo que, dada la longitud de un segmento, calcule el área de un círculo con un radio de dicha longitud.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? Sí, dado que no especifica qué tipo de dato tiene que ser el radio (Integer, Float…).

Después, rellena la especificación:

|  |
| --- |
| Datos: radio, tipo Float.  Pre: radio > 0.0.  Resultado: Pi \* (radio \*\* 2), de tipo Float, si radio > 0.0. Sino, muestra al usuario el error.  Post: el resultado a devolver será un Float > 0.0 si el radio pasado > 0.0. |

* 1. Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| # | Radio | Descripción | Área |
| 1 | 3.0 | Float positivo | 7.45088E+01 |
| 2 | -2.0 | Float negativo | Error definido en el programa |
| 3 | 4 | Integer positivo | Error en el tipo del parámetro de entrada |
| 4 | -12 | Integer negativo | Error en el tipo del parámetro de entrada |

* 1. Algoritmo

WITH Ada.Numerics;

USE Ada.Numerics;

FUNCTION area\_circulo (radio: in Float) return Float IS

-- Datos: radio, tipo Float

-- Pre: radio > 0.0

-- Resultado: Pi \* (radio \*\* 2), de tipo Float, si radio > 0.0. Sino, muestra al usuario el error

-- Post: el resultado a devolver sera un Float > 0.0 si el radio introducido es mayor a 0.0.

BEGIN

if radio > 0.0 then

return Pi \* (radio \*\* 2);

else

raise CONSTRAINT\_ERROR with "El radio tiene que ser mayor que 0.0";

end if;

END area\_circulo;

1. **Es\_par**

Crea un algoritmo que, dado un número natural, indique si es par (true) o impar (false).

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? No

Después, rellena la especificación:

|  |
| --- |
| Datos: 1 parametro, num: in Natural.  Pre: num >= 0. Esto se da por hecho si utilizamos el tipo Natural en el parámetro num.    Resultado: Resultado: retorna es\_impar: Boolean.  Post: se tiene que devolver un booleano. Si el numero introducido dividido entre 2 devuelve un resto de 0, entonces es\_par será True. Sino, es\_par será False. |

* 1. Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| # | Num | Descripción | Es\_par |
| 1 | 8 | Numero positivo par | True |
| 2 | 13 | Numero positivo impar | False |

* 1. Algoritmo

FUNCTION par\_o\_impar(num: Natural) RETURN Boolean IS

-- Datos: 1 parametro, num: in Natural.

-- Pre: num >= 0. Esto se da por hecho si utilizamos el tipo Natural en el parametro num.

-- Resultado: retorna es\_impar: Boolean

-- Post: se tiene que devolver un booleano. Si el numero introducido dividido entre 2 devuelve un resto de 0, entonces es\_par sera True. Sino, es\_par sera False

es\_par: Boolean := (if num mod 2 = 0 then True else False);

BEGIN

return es\_par;

END par\_o\_impar;

1. **Ordenar dos números (I)**

Crea un algoritmo que, dados dos valores enteros mayores que 0, *num1* y *num2*, los ordene de menor a mayor de manera que *num1* contenga el menor y *num2* el mayor (utiliza como base el algoritmo de calcular el máximo –*Máximo1*– que se explicó en la clase de teoría)

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? Sí, dado que no se especifica qué hacer en caso de que ambos números sean iguales y, por consiguiente, no exista ni mayor ni un menor.

Después, rellena la especificación:

|  |
| --- |
| Datos: 2 parámetros, num1 y num2, ambos in Positive.  Pre: num2, num1 > 0. Esto siempre será así puesto que los hemos tipado como Positive.    Resultado: 2 parámetros out, num1(el número menor) y num2(el número mayor), ambos Positive.  Post: num1 será el menor y num2 el mayor. Si son iguales, se dejan los valores por defecto. |

* 1. Casos de Prueba

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Num1 | Num2 | Descripción | Num1 | Num2 |
| 1 | 1 | 8 | Num1 mayor que num2 | 1 | 8 |
| 2 | 9 | 2 | Num2 mayor que num1 | 2 | 9 |
| 3 | 6 | 6 | Ambos números iguales | 6 | 6 |

* 1. Algoritmo

PROCEDURE ordenar\_dos\_numeros\_1 (num1, num2: in out Positive) IS

-- Datos: 2 parametros, num1 y num2, ambos in Positive

-- Pre: num1 y num2 > 0. Esto siempre sera asi puesto que los hemos tipado como Positive

-- Resultado: 2 parametros out, num1(el numero menor) y num2(el numero mayor), ambos Positive

-- Post: si num1 es menos que num2, num1 tomara el valor de num2 y viceversa. Ambos numeros tendran que ser mayores que 0.

BEGIN

if (num1 > num2) then

num2 := num2 + num1;

num1 := num2 - num1;

num2 := num2 - num1;

end if;

END ordenar\_dos\_numeros\_1;

1. **Ordenar dos números (reusando el ejercicio 6)**

Crea un algoritmo que pida al usuario dos valores enteros mayores que 0 los ordene de menor a mayor usando el algoritmo del ejercicio 3 y los presente por pantalla en ese orden (utiliza como base el algoritmo de calcular el máximo –*Máximo2*– que se explicó en la clase de teoría).

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? Sí, dado que no se especifica qué hacer en caso de que ambos números sean iguales y, por consiguiente, no exista un mayor y un menor.

Después, rellena la especificación:

|  |
| --- |
| Entrada: el usuario introduce 2 números positivos (num1, num2).  Pre: num2, num1 > 0 (pedidos por teclado al usuario). Esto siempre será así puesto que los hemos tipado como Positive.    Salida: mostramos al usuario el num1 (el número menor) y num2 (el número mayor).  Post: num1 será el menor y num2 el mayor. Si son iguales, se dejan los valores por defecto. |

* 1. Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| # | Entrada estándar (teclado) | Descripción | Salida estándar (pantalla) |
| 1 | 1 | 8 | Num1 mayor que num2 |
| 2 | 9 | 2 | Num2 mayor que num1 |
| 3 | 6 | 6 | Ambos números iguales |

* 1. Algoritmo

WITH Ada.Text\_IO, Ada.Integer\_Text\_IO;

USE Ada.Text\_IO, Ada.Integer\_Text\_IO;

PROCEDURE ordenar\_dos\_numeros\_2 IS

-- Entrada: el usuario introduce 2 numeros positivos (num1, num2)

-- Pre: num1 y num2 > 0. Esto siempre sera asi puesto que los hemos tipado como Positive

-- Salida: mostramos al usuario el num1 (el numero menor) y num2 (el numero mayor)

-- Post: num1 sera el menor y num2 el mayor. Si son iguales, se dejan los valores por defecto.

num1, num2: Positive;

BEGIN

put("Introduce el primer numero: ");

get(num1);

new\_line;

put("Introduce el segundo numero: ");

get(num2);

new\_line;

if (num1 > num2) then

num1 := num1 + num2;

num2 := num1 - num2;

num1 := num1 - num2;

end if;

put("El numero menor es " & Integer'Image(num1) & " y el numero mayor es " & Integer'Image(num2));

END ordenar\_dos\_numeros\_2; num1 := num1 + num2;

num2 := num1 - num2;

num1 := num1 - num2;

end if;

put("El numero menor es " & Integer'Image(num1) & " y el numero mayor es " & Integer'Image(num2));

END ordenar\_dos\_numeros\_2;

1. **Ordenar tres números (I)**

Crea un algoritmo que pida al usuario tres valores enteros mayores que 0, num1, num2 y num3, los ordene de menor a mayor y los presente por pantalla en ese orden.

* 1. Especificación

Antes que nada, ¿es ambiguo el enunciado? Sí, dado que no se especifica qué hacer si 2 o 3 números son iguales (es decir, si no hay número menor y número mayor).

Después, rellena la especificación:

|  |
| --- |
| Entrada: el usuario introduce por teclado num1, num2 y num3: Positive.  Pre: num1, num2, y num3 > 0. Estos siempre serán > 0 puesto que están tipados como Positive.    Salida: se le muestra al usuario (put) num1(el menor), num2 y num3(el mayor).  Post: si num1 es mayor que num2, se intercambiaran sus valores. Mismo procedimiento con num1 y num2, y num2 y num3. Los tres numeros terminaran el programa con un valor > 0. |

* 1. Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| # | Entrada estándar (teclado) | Descripción | Salida estándar (pantalla) |
| 1 | 1, 2 y 7 | Primer número el menor y último número el mayor | 1, 2 y 7 |
| 2 | 7, 2 y 1 | Primer número el mayor y último número el menor | 1, 2 y 7 |
| 3 | 8, 1, 22 | Segundo número el menor y último el mayor | 1, 8, 22 |
| 4 | 3, 5, 3 | Dos números iguales | 3, 3, 5 |
| 5 | 5, 5, 5 | Todos los números iguales | 5, 5, 5 |

* 1. Algoritmo

WITH Ada.Text\_IO, Ada.Integer\_Text\_IO;

USE Ada.Text\_IO, Ada.Integer\_Text\_IO;

PROCEDURE ordenar\_tres\_numeros IS

-- Entrada: el usuario introduce por teclado num1, num2 y num3: Positive

-- Pre: num1, num2, y num3 > 0. Estos siempre seran > 0 puesto que estan tipados como Positive

-- Salida: se le muestra al usuario (put) num1(el menor), num2 y num3(el mayor)

-- Post: si num1 es mayor que num2, se intercambiaran sus valores. Mismo procedimiento con num1 y num2, y num2 y num3. Los tres numeros terminaran el programa con un valor > 0.

num1, num2, num3: Positive;

BEGIN

put("Introduzca el primer numero: ");

get(num1);

new\_line;

put("Introduzca el segundo numero: ");

get(num2);

new\_line;

put("Introduzca el tercer numero: ");

get(num3);

new\_line;

-- Pasar el valor menor a num1 y pasar el valor mayor a num3

if (num1 > num2) then

num2 := num2 + num1;

num1 := num2 - num1;

num2 := num2 - num1;

end if;

if (num1 > num3) then

num3 := num3 + num1;

num1 := num3 - num1;

num3 := num3 - num1;

end if;

-- Ahora el valor menor esta en num1

if (num2 > num3) then

num3 := num3 + num2;

num2 := num3 - num2;

num3 := num3 - num2;

end if;

-- Ahora el valor mayor esta en num3

put(Integer'Image(num1) & " <= " & Integer'Image(num2) & " <= " & Integer'Image(num3) & ".");

END ordenar\_tres\_numeros;

-- Ahora el valor mayor esta en num3

put(Integer'Image(num1) & " <= " & Integer'Image(num2) & " <= " & Integer'Image(num3) & ".");

END ordenar\_tres\_numeros;