Capítulo 6

Metodología, Técnica Y Tecnología Para Solucionar Un Problema

Hasta este momento tenemos una metodología para solucionar un problema, conocemos unas técnicas para representar la solución y hemos hablado de la tecnología a nivel de lenguajes de programación para que el computador cumpla por nosotros el objetivo propuesto. Todo esto se une en una teoría que nos permite acercarnos a la lógica de programación y por supuesto, gracias al primer contacto que ya tenemos con los lenguajes, a la programación como tal. Es importante saber que cuando se habla de lógica de programación se está hablando de ese conjunto de normas técnicas que nos permiten que de una manera sencilla nosotros desarrollemos un algoritmo entendible para la solución de un problema y cuando se habla de programación como tal es la utilización de lenguajes que permiten que nuestra solución sea entendida y ejecutada por un computador.

Precisamente y con el ánimo de ilustrar todo la teoría que hasta el momento hemos visto, vamos a dar tres enunciados y vamos a resolverlos aplicando toda la metodología para solucionar un problema, utilizando las técnicas de representación y codificándolos en unos lenguajes de programación. Para ello vamos a tener en cuenta que la teoría expuesta hasta el momento se resume en los siguientes tópicos:

1. Concepción del problema

Es muy importante que cuando tengamos un enunciado podamos tener de él una concepción acertada de manera que nos podamos alcanzar un objetivo y que ése objetivo sea el que

realmente necesita ser solucionado. La concepción del problema es el camino para tener la certeza de que lo hemos entendido correctamente y que lo que buscamos solucionar coincide con lo que se busca solucionar en el problema. Dentro de la concepción del problema tendremos las siguientes etapas:

a. Clarificación del objetivo

Por lo dicho en capítulos anteriores es muy importante que a través de un razonamiento teórico y textual nos sentemos a reflexionar en cuanto a los alcances de nuestro objetivo (enunciado como un problema) ya que con eso tendremos muy claro no solo hacia donde debemos ir sino hasta donde debemos llegar.

b. Algoritmo

Es el conjunto de pasos que nos permiten llegar (ojalá de la mejor de las formas) a alcanzar el objetivo propuesto. Debe ser organizado y, ante todo, ordenado para que sea absolutamente entendible.

c. Prueba de Escritorio

Es la prueba reina de un algoritmo. Nos permite saber si realmente está bien o no. Cuándo un algoritmo está bien..? Solamente cuando realmente alcanza el objetivo propuesto. Si un algoritmo no alcanza el objetivo que inicialmente se propuso estará mal así haga maravillas en su desarrollo.

2. Técnicas de Representación

Es importante que usted conozca y domines las técnicas de representación porque con ello usted podrá evaluar ventajas y desventajas reales (y para usted) y podrá determinar cuál es la técnica mas apropiada para la representación de sus algoritmos. No está de mas decir que cuando se quiere representar un algoritmo solamente se utiliza una de las técnicas pero para los objetivos explicativos de este libro representaremos los algoritmos de este capítulo y de otros subsiguientes con las tres técnicas, solo para que usted encuentre diferencias entre ellos y ello le permita realizar una correcta evaluación y establecer unos criterios firmes acerca de su utilización.

a. Diagramas de Flujo

Representados por signos en donde el hilo conductor de la lógica se representa por flechas que van a significar la dirección del flujo de la idea.

b. Diagramación Rectangular Estructurada

Esquema en donde se utiliza un rectángulo como base y utilizando solo tres tipos de notaciones se puede representar todo lo que para nosotros sea parte de un algoritmo.

c. Seudocódigo

Texto basado en unas normas técnicas que lo hace muy entendible y sobre todo muy fácil de codificar y que representa, obviamente, la solución que hayamos planteado a través de un algoritmo.

3. Transcripción o Codificación

Es la representación de un algoritmo a través de un Lenguaje de Programación. En este capítulo utilizaremos los lenguajes *Basic*, *Pascal*, *C* y *Cobol* como ejemplos y explicaremos brevemente y de manera muy somera, ya que no es el objetivo del libro, algunos tópicos acerca de cada uno de los lenguajes. También es importante que usted sepa que cuando vaya a desarrollar realmente programas aplicativos solo va a tener que codificar en un solo lenguaje de programación. En este libro lo haremos en cuatro lenguajes solo por el ánimo explicativo del libro y para establecer algunas diferencias entre uno y otro lenguaje.

PRIMER ENUNCIADO

Desarrollar un programa que permite leer un número entero positivo y determinar si es par.

1. Concepción del problema

a. Clarificación del objetivo

Se trata de recibir un número entero (para lo cual utilizaremos una variable de tipo entero), verificar que es un número positivo y determinar si es un número par. Recordemos pues que son números pares aquellos que son divisibles exactamente entre dos, o sea, aquellos que al dividirlos entre 2 su residuo es cero. Algunos números pares son 18, 6, 4, 56 y 88. Algunos números que no son pares son 45, 7, 19, 23 y 99 ya que no cumplen con las condiciones de los números pares. En caso de que el número leído sea para avisaremos a través de un título que el *número sí es par* y en caso de que no sea así entonces haremos lo mismo avisando que el *número no es par*. Apenas hayamos avisado a través de un título que el número es par o que no lo es, entonces allí deberá terminar nuestro algoritmo.

b. Algoritmo

Algoritmo para determinar si un número es par
Inicio
    Leer un número y guardarlo en una variable entera
    Si ese número es negativo
        Escribir que ese número no sirve para nuestro propósito
    Sino
        Preguntar si el número es par
        Si lo es entonces escribir que el número leído es par
        Si no lo es escribir que el número leído no es par
Fin

Ya de por sí debemos tener en cuenta que el algoritmo es en sí la esencia de nuestra idea tal y como está representado aquí. Ahora lo que tenemos que hacer es ir mutando nuestra idea para que se convierta en una solución mas aproximada a lo técnico que a lo informal. Comencemos pues con un análisis detallado de cada una de las órdenes que aparecen en este algoritmo:

Si vamos a convertir este algoritmo en un programa entonces el nombre debe ser un poco mas técnico. De esta manera no lo vamos a llamar *Algoritmo para determinar si un número es par* sino que lo vamos a llamar *Algoritmo Número_Par* y será nuestra obligación recordar que el Algoritmo Número_Par es el algoritmo que nos permite leer un número y determinar si es par.

Como vamos a tener la necesidad de utilizar una variable para que almacene el número que se va a leer entonces es necesario que al inicio del algoritmo declaremos una variable de tipo entero a la cual vamos a llamar (por conveniencia técnica) num, de esta forma la cabecera de nuestro algoritmo que estaba así

Algoritmo para determinar si un número es par
Inicio

Se va a transformar en

Algoritmo Número_Par
Variables
    Entero: num
Inicio

Esto significa que durante el algoritmo vamos a utilizar una variable que la vamos a llamar num, que solo podrá almacenar datos de tipo entero y que cuando se utilice en operaciones sus resultados se van a regir por las reglas de la aritmética entera (es decir sin decimales).

Como ya tenemos una variable en donde vamos a almacenar el número que se lea entonces la orden *Leer un número y guardarlo en una variable entera* se convertirá conceptualmente en *Leer*

un número y guardarlo en la variable entere num que por lo dicho en capítulos anteriores es lo mismo que decir *Lea num* (orden dada al computador).

Entonces nuestro algoritmo que hasta el momento era

Algoritmo para determinar si un número es par
Inicio
    Leer un número y guardarlo en una variable entera

Se convierte ahora en

Algoritmo Número_Par
Variables
    Entero :num
Inicio
    Lea num

Como ya tenemos el valor guardado en una variable entonces preguntar por el número leído es lo mismo que preguntar por el contenido de la variable y hemos de recordar que cuando se utiliza el nombre de una variable en un algoritmo eso representará que nos estamos refiriendo al contenido de dicha variable. Igualmente preguntar si un número es negativo se reduce a preguntar si dicho número es menor que 0 valiéndonos de un operador relacional (<). En esas condiciones la pregunta

    Si ese número es negativo
        Escribir que ese número no sirve para nuestro propósito

Se convierte en

    Si num < 0
        Escriba "El número debe ser positivo"

Y por lo tanto nuestro algoritmo que originalmente era

Algoritmo para determinar si un número es par
Inicio
    Leer un número y guardarlo en una variable entera
    Si ese número es negativo
        Escribir que ese número no sirve para nuestro propósito

Se convierte ha transformado, hasta el momento, en

Algoritmo Número_Par
Variables

```
        Entero :num
Inicio
        Lea num
        Si num < 0
                Escriba "El número debe ser positivo"
```

Si esta última pregunta es falsa querrá decir que el número es mayor que ó igual a 0 y por lo tanto pasaremos a realizar la siguiente pregunta

```
Sino
                Preguntar si el número es par
                Si lo es entonces escribir que el número leído es par
                Si no lo es escribir que el número leído no es par
Fin
```

Que consistirá en determinar si el número es par para avisar a través de un título que Sí lo es o que No lo es. Pero como convertimos técnicamente la pregunta *Si el número es par* para que el computador la pueda ejecutar y obtener la respuesta apropiada.? Una de las formas es aprovechando las características de la aritmética entera. Recuerde que en esta aritmética no se generan decimales por lo tanto si nosotros tomamos un número y lo dividimos entre dos eso nos dará un resultado. Si ese resultado lo multiplicamos por dos ¿Nos volverá a dar el mismo número inicial...? Sí, pero solamente cuando este haya sido par ya que si hubiera sido impar al hacer la división entre dos se pierden sus decimales.

Vamos a hacerlo con un ejemplo: Si dividimos 7 entre 2 ¿cuánto nos da..? (Recuerda que son datos enteros y que estamos trabajando con aritmética entera por ser estos dos datos enteros) pues el resultado es 3 ya que en aritmética entera no se generan decimales. Y si ahora tomamos este resultado y lo multiplicamos por 2 nos va a dar 6 que no es igual al 7 inicial, por lo tanto podemos decir que como 6 no es igual a 7 entonces el 7 no es par.

Tal vez usted dirá Qué bobada..!! Si todos sabemos cuando un número es para o no. Pero no se olvide que el que va a ejecutar este algoritmo (convertido obviamente en programa es el computador y ese sí no fue a la escuela como nosotros).

Igualmente y para continuar con nuestro ejemplo si el número 8 es dividido entre 2 obtenemos el resultado 4 y si ese resultado es multiplicado por 2 obtenemos el 8 inicial. Por lo tanto podemos decir que 8 es un número par.

Luego para determinar si un número cualquiera es par todo lo que tenemos que hacer es dividirlo entre 2 y multiplicarlo por 2 si al final se obtiene el resultado inicial es porque el número es par. Si no se obtiene el resultado inicial es porque el número no es par.

De tal forma que nuestra pregunta

```
Sino
                Preguntar si el número es par
```

```
        Si lo es entonces escribir que el número leído es par
        Si no lo es escribir que el número leído no es par
Fin
```

Se convierte en

```
Sino
        Si num / 2 * 2 = num
                Escriba "El número leído es par"
        Sino
                Escriba "El número leído no es par"
Fin
```

Cuando se vaya a resolver la pregunta *Si num / 2 * 2 = num* no se olvide de la jerarquía de operadores para que el resultado sea el correcto y por ende la respuesta a dicha pregunta.

Entonces nuestro algoritmo que inicialmente era

```
Algoritmo para determinar si un número es par
Inicio
    Leer un número y guardarlo en una variable entera
    Si ese número es negativo
        Escribir que ese número no sirve para nuestro propósito
    Sino
        Preguntar si el número es par
        Si lo es entonces escribir que el número leído es par
        Si no lo es escribir que el número leído no es par
Fin
```

Se ha convertido ahora en un algoritmo técnico así

```
Algoritmo Número_Par
Variables
    Entero :num
Inicio
    Lea num
    Si num < 0
        Escriba "El número debe ser positivo"
    Sino
        Si num / 2 * 2 = num
            Escriba "El número leído es par"
        Sino
            Escriba "El número leído no es par"
Fin
```

Cuál es la verdadera diferencia entre uno y otro..? Pues la diferencia es que la segunda versión de este algoritmo es absoluta y fácilmente codificable en un Lenguaje de Programación y la primera versión no es tan fácilmente codificable dado la gran cantidad de razonamientos que hay que

hacer. Debo aclararle que la primera versión es el algoritmo como tal o sea puro y sin ningún tipo de retoque. La segunda versión es el mismo algoritmo pero expresado bajo la técnica del seudocódigo que no es mas que una representación técnica y textual de un algoritmo.

c. Prueba de Escritorio

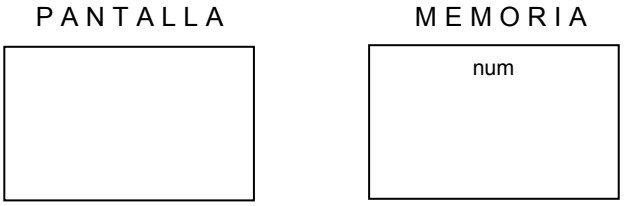
Cómo se hace realmente una prueba de escritorio..? Muy sencillo usted va a tener dos elementos que manejar en una prueba de escritorio: el primero es la memoria en donde se van a manejar las variables que intervengan en el programa y el segundo es la pantalla (o unidad de salida cualquiera que ésta sea) por donde usted va a obtener los resultados de su algoritmo. Entonces desarrolle paso a paso lo que diga el algoritmo utilizando las variables que el mismo le indique y colocando en pantalla los título que él mismo algoritmo le diga. Sencillamente suponga que usted es el computador. Cuando llegue al Fin del algoritmo todo lo que tiene que hacer es mirar en la pantalla (o unidad de salida que usted haya representado) y ver si lo que dice allí coincide con el objetivo que inicialmente se había propuesto. De ser así su algoritmo estará bien. Si no es así el algoritmo estará mal y usted tendrá que corregirlo para volver a realizarle una prueba de escritorio.

De esta manera si tomamos el algoritmo técnico final y le realizamos una prueba de escritorio obtenemos lo siguiente

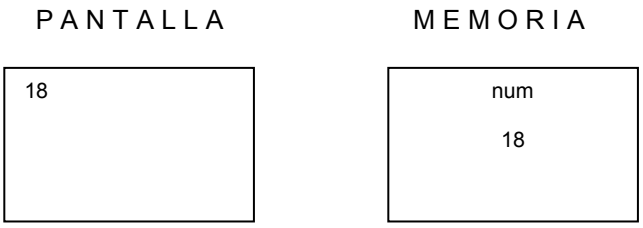
```
Algoritmo Número_Par
Variables
        Entero :num
Inicio
        Lea num
        Si num < 0
                Escriba "El número debe ser positivo"
Sino
                Si num / 2 * 2 = num
                        Escriba "El número leído es par"
                Sino
                        Escriba "El número leído no es par"
Fin
```

Tal como lo indica el algoritmo en su parte inicial en memoria tenemos una variable que se llama num y en pantalla inicialmente estamos en blanco

```
Algoritmo Número_Par
Variables
        Entero :num
```



A partir de este momento somos computadores. Solo vamos a hacer lo que el algoritmo nos diga.... El algoritmo nos dice *Lea num* entonces vamos a asumir que el número 18 es escrito a través del teclado y reflejado en la pantalla. Dicho número es almacenado en la variable num porque así se lo hemos dicho a través del mismo algoritmo (no se olvide que *Lea num* significa *Lea un número entero y guárdelo en la variable num*).



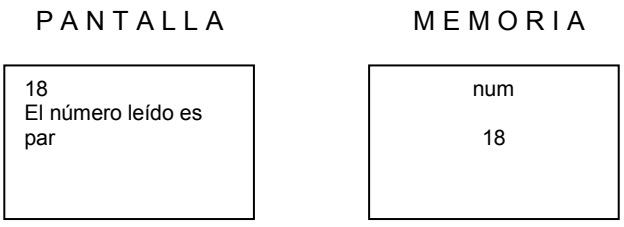
A continuación sigue la pregunta *Si num < 0*. Como sabemos que cuando se refiera a la variable num realmente se está refiriendo al contenido de la variable num entonces internamente la pregunta sería *Si 18 < 0* a lo cual podemos responder (y así lo haría el computador) que es Falso. Por lo cual optamos por desarrollar las órdenes que se encuentran después del *Sino* (que son las órdenes que se deben ejecutar cuando la condición sea Falsa como en este caso). Nuestra situación tanto de pantalla como de memoria siguen iguales, no ha habido ningún cambio en ninguna de las dos.

A continuación, y siguiendo con el *Sino* nos encontramos con la pregunta *Si num / 2 * 2 = num*. Para ello lo primero que debemos hacer es resolver la primera parte de la pregunta o sea realizar la operación que se derive de *num / 2 * 2* y obtener su resultado. Así reemplazando *num* por su contenido y aplicando la jerarquía de operadores obtenemos que la expresión y su resultado sería

*num / 2 * 2 = num*
*18 / 2 * 2 = 18*
*9 * 2 = 18*
18 = 18

Con lo cual la pregunta inicial que era *Si num / 2 * 2 = num* se convierte en *Si 18 = 18* a lo cual nuestra respuesta (como computadores que somos en este momento) es Verdadero entonces procedemos a realizar la acción de Escribir en pantalla “El número leído es par” tal como lo indica nuestro algoritmo.

Luego en nuestra pantalla aparece



Verificamos finalmente lo que hay en pantalla y vemos que está el número *18* y después la frase *El número leído es par* que es Verdad referente al número 18. Como nuestro objetivo era desarrollar

un programa que nos permitiera leer un número y determinar si era un número par entonces podemos decir que este algoritmo SÍ cumple con el objetivo o sea que está bien.

Ahora usted deberá realizarle la prueba de escritorio a este algoritmo suponiendo que el número inicial leído es el 25. No se olvide de la jerarquía de los operadores y de que mientras usted esté haciendo una prueba solo debe acatar y ejecutar las órdenes tal y como se las indique el algoritmo pues es así como el computador va a hacer realidad nuestros programas.

2. Técnicas de Representación

Una vez desarrollado el algoritmo y habiéndosele realizado una correcta "Prueba de Escritorio" se procede a representarse por ALGUNA DE LAS FORMAS TECNICAS vistas. Hago hincapié que son alguna porque para efectos de aprendizaje en este libro representaremos este algoritmo usando las tres técnicas pero en la realidad se utiliza solamente una de ellas. Es muy importante que sepa que la representación se debe hacer cuando el algoritmo está escrito técnicamente.

Para ello recordemos que hemos, hasta el momento, desarrollado el mismo algoritmo de dos formas: La forma informal y la forma técnica. Se acuerda cuál es la forma informal ..? Pues aquí se la presento de nuevo:

```
Algoritmo para determinar si un número es par
Inicio
        Leer un número y guardarlo en una variable entera
        Si ese número es negativo
                Escribir que ese número no sirve para nuestro propósito
        Sino
                Preguntar si el número es par
                Si lo es entonces escribir que el número leído es par
                Si no lo es escribir que el número leído no es par
Fin
```

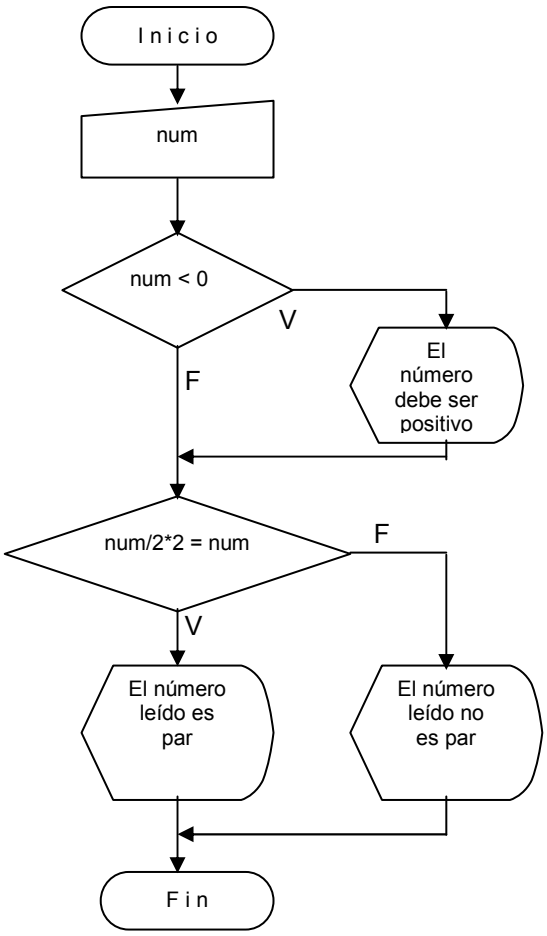
Como puede ver la característica principal es que la forma informal nos da una idea muy coloquial de nuestra solución pero se sale mucho de los esquemas técnicos que necesitamos para poder llevar este algoritmo a un computador. Por eso la forma técnica es la que nos permite realmente llevar al computador lo que hemos considerado como solución a nuestro objetivo. Cuál es la forma técnica..? Pues aquí se la presento también:

```
Algoritmo Número_Par
Variables
        Entero :num
```

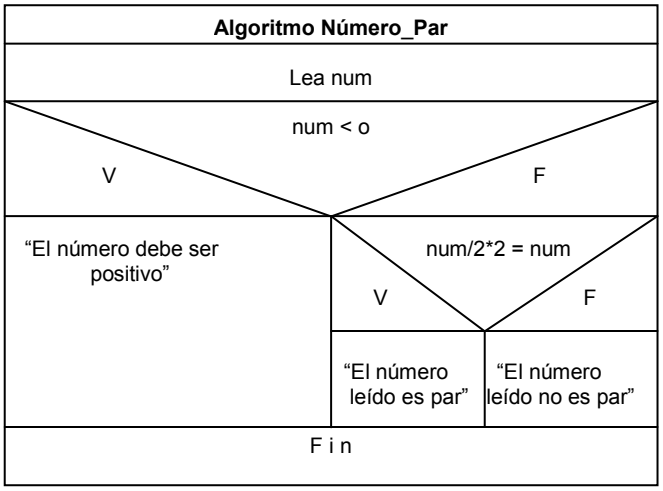
```
Inicio
        Lea num
        Si num < 0
                Escriba "El número debe ser positivo"
Sino
                Si num / 2 * 2 = num
                        Escriba "El número leído es par"
                Sino
                        Escriba "El número leído no es par"
Fin
```

En la forma técnica muchas de las órdenes son equivalentes a instrucciones de un Lenguaje de Programación y ello es lo que nos permite que fácilmente podamos convertir el algoritmo en un programa. Esto significa que es ésta última forma la que vamos a representar utilizando las técnicas estudiadas.

a. Diagrama de Flujo



b. Diagramación Rectangular Estructurada



No se olvide que lo que va entre comillas dobles en la Diagramación Rectangular Estructurada representa un título y así debe quedar bien sea en la pantalla o en cualquier unidad de salida.

c. Seudocódigo

Y cómo se representa este algoritmo en Seudocódigo..? Pues precisamente el algoritmo técnico (o escrito teniendo en cuenta algunas normas técnicas) es el equivalente del algoritmo solución en seudocódigo. De acuerdo a esto (y por una vez mas) el Seudocódigo sería:

```
Algoritmo Número_Par
Variables
        Entero :num
Inicio
        Lea num
        Si num < 0
                Escriba "El número debe ser positivo"
        Sino
                Si num / 2 * 2 = num
                        Escriba "El número leído es par"
                Sino
                        Escriba "El número leído no es par"
Fin
```

3. Transcripción o Codificación

Con el ánimo de dar ejemplos concretos voy a mostrarle a usted, amigo lector, que cuando se va a programar lo mas importante no es el conocimiento de un Lenguaje de Programación (ya que con su uso uno se va acostumbrando a sus reglas) sino la lógica de programación que usted use para

desarrollar soluciones algorítmicas encaminadas a lograr un objetivo. De acuerdo con esto la diferencia entre un Lenguaje y otro serán solo sus instrucciones debido a que la lógica seguirá siendo la misma. Vamos pues a mostrar como sería el algoritmo del ejercicio escrito en términos de cuatro lenguajes: Basic, Pascal, C y Cobol. El objetivo perseguido en esta parte del libro no es que usted de una vez comience a programar utilizando un Lenguaje determinado. El objetivo es que usted simplemente vea, con un ejemplo sencillo, que utilizar cualquier lenguaje de programación es fácil si tiene usted una lógica que le permita desarrollar y entender unos buenos algoritmos y además tenga una somera idea de qué es utilizar un lenguaje como tal. No voy a detenerme a explicar nada de la sintaxis de cada lenguaje ya que no es ése el objetivo del Libro.

Con lo dicho anteriormente recordemos (otra vez) el algoritmo solución original:

```
Algoritmo Número_Par
Variables
        Entero :num
Inicio
        Lea num
        Si num < 0
                Escriba "El número debe ser positivo"
        Sino
                Si num / 2 * 2 = num
                        Escriba "El número leído es par"
                Sino
                        Escriba "El número leído no es par"
Fin
```

a. Versión en Lenguaje Basic

```
input num
if num < 0 then
    print "El número debe ser positivo"
else
    if int(num/2*2) = num then
            print "El numero leído es par"
    else
            print "El número leído no es par"
```

b. Versión en Lenguaje Pascal

```
program numero_par;
var
        num     :       integer;
begin
        readln(num);
        if (num < 0) then
                writeln (' El número debe ser positivo ');
        else
            if  (num/2*2 = num) then
```

```
            writeln(' El número leído es par ');
        else
            writeln(' El número leído no es par ');
end.
```

c. Versión en Lenguaje C

```
#include <iostream.h>

void main( )
{
        int num;

        cin >> num;
        if ( num < 0)
                cout << "El número debe ser positivo ";
        else
                if (num/2*2 == num)
                        cout << "El número leído es par";
                else
                        cout << "El número leído no es par";
}
```

d. Versión en Lenguaje Cobol

```
        IDENTIFICATION DIVISION.
        PROGRAM_ID. NUMERO_PAR.

        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. CLON.
        OBJECT-COMPUTER. CLON.

        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 NUM                  PIC 99.

        PROCEDURE DIVISION.
        INICIO.
                ACCEPT NUM LINE 10 POSITION 10 NO BEEP
                IF NUM IS LESS THAN 0 THEN
                        DISPLAY " EL NÚMERO DEBE SER POSITIVO" LINE 12 COL 10
                ELSE
                        IF NUM / 2 * 2 IS EQUAL TO NUM THEN
                                DISPLAY "EL NÚMERO LEÍDO ES PAR" LINE 12 COL 10
                        ELSE
                                DISPLAY "EL NÚMERO LEÍDO NO ES PAR" LINE 12 COL 10
                STOP RUN.
```

Puede usted notar que la utilización de cualquier lenguaje de programación se reduce casi a reemplazar algunas palabras clave por las que corresponden en el Lenguaje pero la lógica como tal permanece allí intacta. Por ahora solo me interesa que, con un vistazo ligero, usted solamente compare que, en el fondo, la utilización de cualquier Lenguaje es lo mismo sin importar cuál sea. Siempre tenga en cuenta que lo importante, lo realmente importante, lo verdaderamente importante es la lógica con la cual usted desarrolle sus algoritmos.

SEGUNDO ENUNCIADO

Leer dos números enteros positivos y determinar si el último dígito de un número es igual al último dígito del otro.

1. Concepción del problema

---

a. Clarificación del objetivo

Primero que nada vamos a determinar cuál es el objetivo concreto de este enunciado. Según él tendremos que leer dos números enteros (y por lo tanto tendremos que almacenar cada uno en una variable diferente) y tendremos que comparar mutuamente el último dígito de un número con el último dígito del otro número. En aras de que el enunciado esté absolutamente claro diremos que, para este algoritmo, será el último dígito de un número aquel que tenga el menor peso decimal. De esta forma el último dígito del número 256 es el 6 y el último dígito del número 59 será el 9.

Ahora bien si los dos números leídos son, por ejemplo, el 189 y el 65 entonces tendremos que comparar el 9 (último dígito del 189) con el 5 (último dígito del 65). Si fueran esos los dos números leídos entonces tendríamos que decir que no son iguales. Pero si los números leídos fueran el 37 y el 347 entonces al comparar el 7 del 37 con el 7 del 347 tendríamos que decir que son iguales.

Sé que pensará que esta es una explicación que sobra puesto que el enunciado posiblemente para usted sea muy claro pero es muy bueno que se acostumbre, aún con enunciados muy sencillos, a clarificar el objetivo de manera que pueda usted explicarlo con absoluta certeza. Repito no importa que el enunciado sea sencillo ó parezca sencillo. Lo importante es que usted sepa claramente para donde va para saber por donde se va a ir.

b. Algoritmo

Versión Informal

Algoritmo para comparar el último dígito
Inicio
    Leer un número entero y guardarlo en una variable entera
    Leer otro número entero y guardarlo en otra variable entera

```
        Guardar en una variable el último dígito del primer número leído
        Guardar en otra variable el último dígito del último dígito leído

        Comparar el contenido de estas dos últimas variables
        Si son iguales
                Escribir que los dos últimos dígitos son iguales
        Si no son iguales
                Escribir que los dos últimos dígitos no son iguales
Fin
```

Antes de desarrollar la versión técnica (o el seudocódigo) debemos pensar un momento en la forma como le vamos a decir el computador que guarde en una variable *el último dígito de cualquier número*. Para ello nos vamos a valer de los operadores aritméticos y de esa manera, basado en operaciones, obtener el último dígito de cualquier número (siempre que éste sea positivo). Usted tal vez se preguntará porqué tenemos que valernos de operaciones para obtener el último dígito de un número y no podemos decirlo así explícitamente, sabiendo que el enunciado es tan claro. La razón es muy sencilla. El computador no fue a la escuela y por eso el solo obedece órdenes claras y "ejecutables sin razonamientos", es decir, órdenes que no involucren ningún razonamiento adicional para ser realizadas.

Basado en ello para nosotros es muy claro hablar de *el último dígito de un número cualquiera* pero para el computador no. Por esta razón vamos a utilizar las operaciones pertinentes para obtener el último dígito del número leído. Vamos a asumir que el número leído será almacenado en una variable que se llamará *num*. Si *num* contuviera el valor 156 entonces tendríamos que obtener el dígito 6. Además vamos a asumir que ese último dígito será almacenado en una variable llamada *ud* (como de último dígito). Entonces podríamos decir, para este ejemplo específico que

ud = num – 150

De nuevo si asumimos que *num* vale 156 entonces el resultado será el 6 que estamos buscando. Vemos que 150 es un número constante o sea que si el número almacenado en num fuera 897 no nos serviría ese 150 para obtener el último dígito. Lo que por ahora si podemos hacer es expresar ese 150 en términos de num aprovechando las características de la aritmética entera. El número 150 sería igual a dividir *num* entre 10 y posteriormente multiplicarlo por 10. Recuerde que en aritmética entera se generan decimales. Por lo tanto la expresión *num* / 10 * 10 es realizada por el computador de la siguiente forma (teniendo en cuenta la jerarquía de operadores):

Primero se realiza la división *num* / 10. En el ejemplo hemos asumido que el valor almacenado en *num* es 156 entonces la división *num* / 10 nos da como resultado 15.

Segundo se realiza la multiplicación de ese resultado por 10 lo cual para el ejemplo nos daría 150. Y hemos obtenido el valor que buscábamos.

Ahora vamos a reemplazar la expresión inicial por su equivalente usando la variable *num*. De esta manera tenemos:

ud = num – 150

Como 150 es lo mismo que num / 10 * 10, entonces

*ud = num – num / 10 * 10*

Ahora realicémosle una prueba (paso a paso) asumiendo que la variable *num* tiene el valor 854.

*ud = num – num / 10 * 10*
*ud = 854 – 854 / 10 * 10*
*ud = 854 – 85 * 10*
ud = 854 – 850
ud = 4

Vemos que en la variable *ud* quedó almacenado el último dígito del número. Veamos otra prueba (paso a paso) asumiendo que el valor almacenado en la variable *num* es 5468.

*ud = num – num / 10 * 10*
*ud = 5468 – 5468 / 10 * 10*
*ud = 5468 – 546 * 10*
ud = 5468 – 5460
ud = 8

Pues podemos decir que hemos encontrado la forma genérica de almacenar en una variable el último dígito de un número cualquiera (que era parte de lo que estábamos buscando). No se olvide que el objetivo de este ejercicio es comparar el último dígito de cada uno de dos números y determinar si son iguales. Esto que hasta el momento se ha hecho es solo la forma de facilitar el ejercicio. Esta solución solo tiene una restricción. Vamos a asumir que el valor almacenado en *num* es –563. Realicémosle la prueba (paso a paso) con la fórmula

*ud = num – num / 10 * 10*
*ud = - 563 – (- 563) / 10 * 10*
*ud = - 563 – (- 56) * 10*
ud = - 563 – (- 560)
ud = - 563 + 560
ud = - 3

Esto significa que para que esta fórmula sea utilizada correctamente tenemos que asegurarnos que el valor almacenado en la variable num es positivo.

Ahora sí volvamos a nuestro problema inicial para plantear el algoritmo técnico (o dicho de una vez) el seudocódigo. Recordemos que el enunciado del problema es *Leer dos números enteros positivos y determinar si el último dígito de un número es igual al último dígito del otro*. De acuerdo a esto podríamos considerar el siguiente algoritmo como su versión técnica (no se olvide que los algoritmos expuestos aquí son la versión personal del autor. Si usted desarrolla otro algoritmo que realice lo mismo (o sea que logre el mismo objetivo) y al hacerle la prueba de escritorio ve que

realmente logra el objetivo entonces tanto su algoritmo como el mío estarán bien así sean diferentes. Para el desarrollo del algoritmo técnico (o seudocódigo) utilizaremos las variables

num1	en donde almacenaremos el primer número leído
num 2	en donde almacenaremos el segundo número leído
ud1	en donde se almacenará el ultimo dígito del primer número leído
ud2	en donde se almacenará el último dígito del segundo número leído

Esta vez para facilidad del usuario de este algoritmo, le avisaremos cuando debe entrar los números utilizando una orden Escriba (no se olvide que éstas son órdenes que se le darían al computador).

```
Algoritmo Compara_Ult_digs
Variables
        Entero :  num1, num2, ud1, ud2
Inicio
        Escriba "Digite dos números enteros "
        Lea num1, num2

        Si num1 < 0
                num1 = num1 * (-1)
        Si num2 < 0
                num2 = num2 * (-1)

        ud1 = num1 – num1 / 10 * 10
        ud2 = num2 – num2 / 10 * 10

        Si ud1 = ud2
                Escriba "El último dígito de un número es igual al último dígito del otro"
        Sino
                Escriba "El último dígito de un número no es igual al último dígito del otro"

Fin
```

Es de anotar que en el conjunto de instrucciones

```
        Si num1 < 0
                num1 = num1 * (-1)
        Si num2 < 0
                num2 = num2 * (-1)
```

Lo que se busca es asegurarnos que el valor almacenado en las variables num1 y num2 sea positivo precisamente para que nuestra fórmula para obtener el último dígito funcione bien.

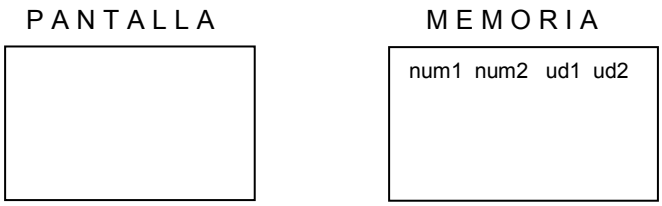
c. Prueba de Escritorio

Ya sabemos que la forma correcta de hacer una prueba de escritorio es realizar todas y cada una de las instrucciones "sin razonarlas" llevando cuidadosamente los cambios en la memoria y realizando lo pertinente en la pantalla. Cuando termine la prueba de escritorio o sea cuando el algoritmo llegue a su fin entonces mire lo que haya quedado en pantalla y compare su coherencia

con el objetivo planteado por el enunciado. Si ve que lo que está en pantalla satisface el objetivo entonces su algoritmo estará bien. Si no es asi entonces tendrá que revisar el algoritmo para que pueda alcanzar el objetivo propuesto.

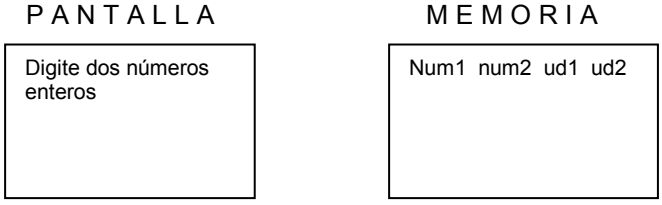
Vamos entonces a desarrollar la prueba de escritorio de este algoritmo realizando paso a paso cada una de las instrucciones y representando lo que vaya pasando tanto en memoria como en pantalla.

Algoritmo Compara_Ult_digs
Variables
Entero : num1, num2, ud1, ud2



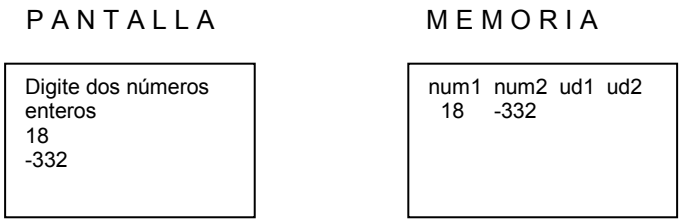
El algoritmo se inicia, para efectos de la Prueba de Escritorio, con una pantalla en blanco y unas variables ubicadas en la memoria sin ningún contenido.

Inicio
Escriba "Digite dos números enteros "



Aparece en pantalla un título solicitando dos números enteros

Lea num1, num2



Para efectos de la prueba de escritorio vamos a asumir que los números leídos son 18 y –332. El computador los almacenará así: el primer valor en la variable *num1* y el segundo valor en la variable *num2* porque en ese orden está la instrucción *Lea num1, num2* que significa *Lea un número entero y guárdelo en la variable num1 y luego lea otro número entero y guárdelo en la variable num2*.

Si num1 < 0
*num1 = num1 * (-1)*

Como el contenido de la variable *num1* es igual a 18 entonces esta pregunta es Falsa por lo tanto pasamos a la siguiente decisión.

Si num2 < 0
*num2 = num2 * (-1)*

Como el contenido de la variable *num2* es igual a –332 entonces la pregunta *Si num2 < 0* es Verdadera. Por lo tanto se ejecuta la orden que está allí o sea *num2 = num2 * (-1)* quedando almacenado en la variable *num2* el valor 332 y anulando el valor anterior de -332. No se olvide que cada que entra un nuevo valor a una variable el valor anterior se borra.

P A N T A L L A

```
Digite dos números
enteros
18
-332
```

M E M O R I A

```
num1  num2  ud1  ud2
 18   -332
       332
```

La siguiente instrucción es:

*ud1 = num1 – num1 / 10 * 10*

Tomando el valor almacenado en la variable *num1* vamos a desarrollar esta expresión paso a paso para, de esta forma, saber con certeza cuánto queda almacenado en la variable *ud1*.

*ud1 = num1 – num1 / 10 * 10*
*ud1 = 18 - 18 / 10 * 10*
*ud1 = 18 – 1 * 10*
ud1 = 18 - 10
ud1 = 8

De esta manera vemos que en la variable ud1 queda almacenado el valor 8 (que corresponde al último dígito del valor contenido en la variable num1).

P A N T A L L A

```
Digite dos números
enteros
18
-332
```

M E M O R I A

```
num1  num2  ud1
ud2
 18   –332   8
       332
```

Siguiente instrucción:

*ud2 = num2 – num2 / 10 * 10*

Conociendo que el valor almacenado en la variable *num2* es 332 entonces desarrollaremos paso a paso la expresión

*ud2 = num2 – num2 / 10 * 10*
*ud2 = 332 – 332 / 10 * 10*
*ud2 = 332 – 33 * 10*
ud2 = 332 – 330
ud2 = 2

Así comprobamos que el valor que queda almacenado en *ud2* es 2 que corresponde al último dígito del valor almacenado en *num2*.

P A N T A L L A

Digite dos números enteros
18
-332

M E M O R I A

num1 num2 ud1 ud2
18 –~~332~~ 8 2
332

Note usted que mientras se han realizado operaciones en la memoria, en la pantalla no sucede se registra ningún cambio.

Siguiente instrucción:

Si ud1 = ud2
    Escriba "El último dígito de un número es igual al último dígito del otro"
Sino
    Escriba "El último dígito de un número no es igual al último dígito del otro"

Sabiendo que la variable ud1 contiene el valor 8 y que la variable ud2 contiene el valor 2 internamente la decisión es *Si 8 = 2*. Como es Falso entonces deberá hacerse lo que está por el *Sino* de la decisión o sea ejecutar la orden *Escriba "El último dígito de un número no es igual al último dígito del otro"*. De esta manera obtenemos:

P A N T A L L A

```
Digite dos números
enteros
18
-332
El último dígito de un
número no es igual al
último dígito del otro
```

M E M O R I A

num1	num2	ud1	ud2
18	~~–332~~ 332	8	2

Y nuestra última instrucción (si se le puede llamar así) es:

Fin

Con lo cual damos por terminada la Prueba de Escritorio. Ahora sí verificamos lo que quedó en pantalla (solamente). En pantalla quedó

P A N T A L L A

```
Digite dos números
enteros
18
-332
El último dígito de un
número no es igual al
último dígito del otro
```

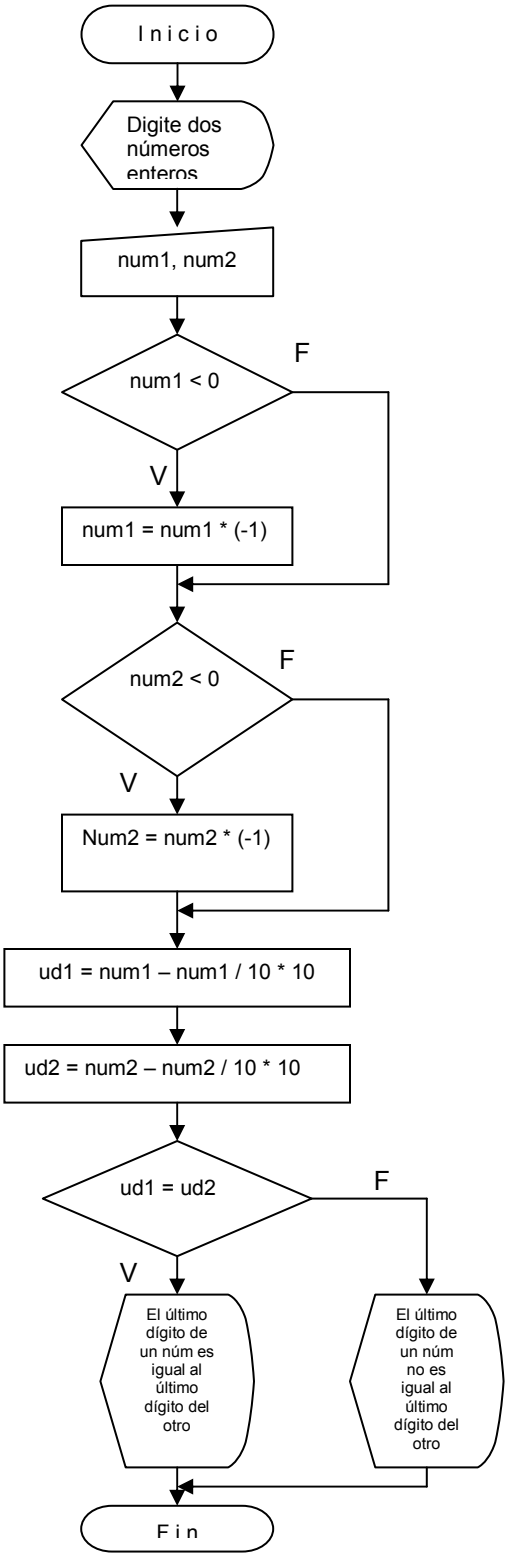
Vemos pues que el algoritmo (basado solo en lo que queda en pantalla...no se le olvide) leyó dos números enteros y pudo determinar que el último dígito de un número no es igual al último dígito del otro y como el enunciado era *Leer dos números enteros positivos y determinar si el último dígito de un número es igual al último dígito del otro* entonces podremos decir que está bien.

Ahora para estar totalmente seguro usted realizará la prueba de escritorio asumiendo que el valor almacenado en la variable *num1* es 459 y el valor almacenado en la variable *num2* es 6239. No olvide que la prueba de escritorio debe ser desarrollada paso a paso y ejecutada tal y como esté en el algoritmo para poder tener absoluta certeza de que el computador va a obtener los mismos resultados que nosotros obtendríamos en esa prueba.

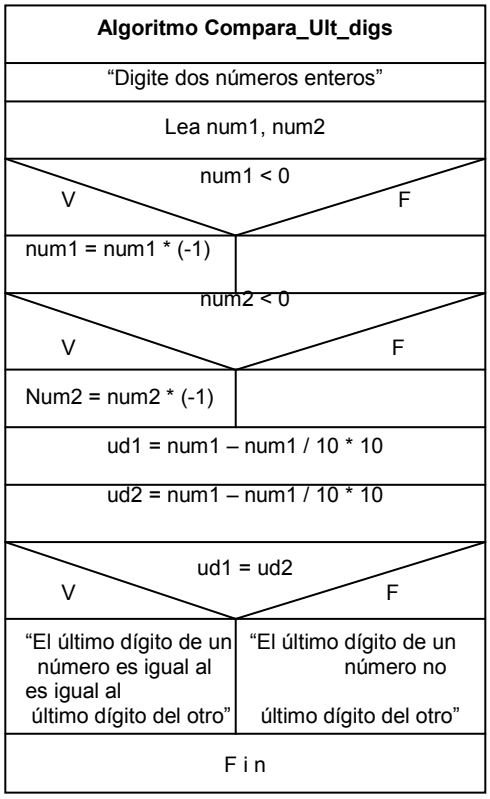
La representación gráfica nos facilita tener una comprensión mejor del algoritmo y nos permite además tenerlo escrito en términos mas técnicos y entendibles a la mayoría de programadores.

2. Técnicas de Representación

a. Diagrama de Flujo



a. Diagramación Rectangular Estructurada



b. Seudocódigo

```
Algoritmo Compara_Ult_digs
Variables
        Entero :  num1, num2, ud1, ud2
Inicio
        Escriba "Digite dos números enteros "
        Lea num1, num2

        Si num1 < 0
                num1 = num1 * (-1)
        Si num2 < 0
                num2 = num2 * (-1)

        ud1 = num1 – num1 / 10 * 10
        ud2 = num2 – num2 / 10 * 10

        Si ud1 = ud2
                Escriba "El último dígito de un número es igual al último dígito del otro"
        Sino
                Escriba "El último dígito de un número no es igual al último dígito del otro"

Fin
```

No se olvide que el seudocódigo no es mas que la versión técnicamente escrita del algoritmo original de manera que ya para nosotros es conocido. Es importante que sepa que en la práctica real se usa el seudocódigo ó el diagrama de flujo ó el diagrama rectangular para representar el algoritmo.

3. Transcripción o Codificación

a. Basic

```
Print “Digita dos números enteros”
Input num1, num2

If num1 < 0 then
    num1 = num1 * (-1)

If num2 < 0 then
    num2 = num2 * (-1)

ud1 = num1 – num1 / 10 * 10
ud2 = num2 – num2 / 10 * 10

if ud1 = ud2
            Print “El último dígito de un número es igual al último dígito del otro”
else
            Print “El último dígito de un número no es igual al último dígito del otro”
```

b. Pascal

```
Program Compara_Ult_digs;
Var
        num1, num2, ud1, ud2 : integer;
Begin
        Writeln (‘Digite dos números enteros ‘);
        Readln (num1, num2);

        if num1 < 0 then
                num1 := num1 * (-1);
        if num2 < 0 then
                num2 := num2 * (-1);

        ud1 := num1 – num1 / 10 * 10;
        ud2 := num2 – num2 / 10 * 10;

        If ud1 = ud2 then
                Writeln (‘El último dígito de un número es igual al último dígito del otro’);
        else
```

```
            Writeln('El último dígito de un número no es igual al último dígito del otro');

End.
```

c. C

```
#include <iostream.h>

void main( )
{
        int  num1, num2, ud1, ud2

        cout << "Digite dos números enteros ";
        cin >>  num1, num2;

        if (num1 < 0)
                num1 = num1 * (-1);
        If (num2 < 0)
                num2 = num2 * (-1);

        ud1 = num1 – num1 / 10 * 10;
        ud2 = num2 – num2 / 10 * 10;

        If ( ud1 == ud2 )
                Cout <<  "El último dígito de un número es igual al último dígito del otro";
        else
                Cout << "El último dígito de un número no es igual al último dígito del otro";

}
```

d. Cobol

```
        IDENTIFICATION DIVISION.
        PROGRAM_ID. COMPARA_ULT_DIGS.

        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. CLON.
        OBJECT-COMPUTER. CLON.

        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 NUM1              PIC 99.
        01 NUM2              PIC 99.
        01 UD1               PIC 99.
        01 UD2               PIC 99.

        PROCEDURE DIVISION.
        INICIO.
                DISPLAY "DIGITE DOS NÚMEROS ENTEROS"        LINE 10 COL 10
                ACCEPT NUM1                                 LINE 12 COL 10
                ACCEPT NUM2                                 LINE 14 COL 10
```

```
IF NUM1 IS LESS THAN 0 THEN
        COMPUTE NUM1 = NUM1 * (-1)
IF NUM2 IS LESS THAN 0 THEN
        COMPUTE NUM2 = NUM2 * (-1)

COMPUTE UD1 = NUM1 – NUM1 / 10 * 10
COMPUTE UD2 = NUM2 – NUM2 / 10 * 10

IF  UD1 IS EQUAL TO UD2 THEN
        DISPLAY “EL ÚLTIMO DÍGITO DE UN NÚMERO ES IGUAL AL ÚLTIMO
        DÍGITO DEL OTRO”     LINE 20 COL 10
ELSE
        ESCRIBA “EL ÚLTIMO DÍGITO DE UN NÚMERO NO ES IGUAL AL
        ÚLTIMO DÍGITO DEL OTRO” LINE 20 COL 10

STOP RUN.
```

TERCER ENUNCIADO

Leer un número entero y determinar cuántos dígitos tiene.

1. Concepción del problema

---

a. Clarificación del objetivo

El objetivo de este algoritmo en esencia busca solicitar un número entero y contar cuántos dígitos tiene. Cómo lo vamos a lograr...? Primero que nada tengamos en cuenta que si el número leído fuera 15 tendríamos que detectar, a través de nuestro algoritmo, que tiene 2 dígitos. Si el número leído fuera 5946 tendríamos que detectar que tiene 4 dígitos. Si el número fuera –958 tendríamos que decir que tiene 3 dígitos. Entonces lo que tenemos que hacer es crear un procedimiento que nos permita saber cuántos dígitos tiene el número.

b. Algoritmo

Algoritmo para determinar la cantidad de dígitos de un número entero
Inicio
Leer un número entero
Guardar en una variable la cantidad de dígitos
Mostrar el contenido de esta variable en pantalla
Fin

Ya sabemos que éste es el algoritmo informal y también sabemos que no es este algoritmo el que podemos llevar, finalmente, a un computador pues involucra una serie de razonamientos que el

mismo computador no tendría. Por ello se hace importante que nosotros transformemos este algoritmo en uno que pueda ser transcrito fácilmente a un Lenguaje de Programación. Para ello lo primero que tendremos que tener en cuenta es que la parte de contar los dígitos del número requerirá de un proceso que posteriormente pueda ser entendible por un computador cuando lo hayamos codificado.

Vamos a aprovechar la aritmética entera de la cual disponemos para obtener la cantidad de dígitos. Supongamos que el número leído fuera 1546 y además supongamos que va a existir una variable llamada Contador De Dígitos (que abreviado sería *cd*) que va a contener la cantidad de dígitos que tenga el número. Iniciamos la variable cd en ceros y tomamos el número leído al cual vamos a llamar *num* por comodidad.

num	*cd*
1546	0

Si dividimos el valor contenido en la variable *num* entre 10 y lo almacenamos de nuevo en la misma variable entonces quedará almacenado en ella el valor 154 (por características de aritmética entera) y además vamos a incrementar en 1 el valor contenido en la variable cd.

num	*cd*
~~1546~~	0
154	1

Si volvemos a dividir el nuevo número almacenado en la variable *num* entre 10 obtendremos el número 15 y además volvemos a incrementar el valor de la variable *cd* en 1, entonces

num	*cd*
~~1546~~	0
~~154~~	1
15	2

Si volvemos a dividir el nuevo contenido de la variable *num* (que ahora es 15) entre 10 obtendremos el número 1 y si volvemos a incrementar el contenido de la variable *cd* en 1 obtendremos el valor 3.

num	*cd*
~~1546~~	0
~~154~~	1
~~15~~	2
1	3

De la misma manera si volvemos a dividir el contenido de la variable *num* entre 10 y volvemos a incrementar el valor de la variable *cd* en 1 entonces obtendremos

num	*cd*
~~1546~~	0

~~154~~	1
~~15~~	2
1	3
0	4

Ya como vemos que el valor contenido en la variable *num* ha llegado a cero y que si siguiéramos dividiendo entre 10 sucesivamente seguiría dando como resultado cero entonces podemos detener allí esta secuencia de divisiones. Puede usted notar que por cada vez que dividíamos entre 10 el valor contenido en la variable *num* incrementábamos el valor de la variable cd. Al finalizar cuando el contenido de la variable *num* ha llegado por primera vez a cero el contenido de la variable *cd* ha llegado a un número que es igual a la cantidad de dígitos que originalmente tenía la variable *num*.

Entonces si quisiéramos resumir esto que hemos hecho tendríamos que

```
cd = 0
mientras num sea diferente de 0
        num = num / 10
        cd = cd + 1
fin_mientras
```

Que no es mas que lo que hemos acabado de hacer paso a paso. Este fragmento nos permite almacenar en la variable *cd* la cantidad de dígitos del número almacenado en *num*. Y qué pasa si el número almacenado en la variable num es negativo..? Pues no pasa nada. El algoritmo sigue sirviendo y entregando el resultado correcto. Por lo tanto el algoritmo completo para determinar cuántos dígitos tiene un número que previamente fue leído expresado en término técnicos es el siguiente

```
Algoritmo Cuenta_Dígitos
Var
        Entero :num, cd
Inicio
        Escriba "Digite un número entero"
        Lea num

        cd = o
        Mientras num < > 0
                num = num / 10
                cd = cd + 1
        Fin_mientras
        Escriba "El número digitado tiene " cd "dígitos"
Fin
```

Hemos utilizado el operador < > para expresar *diferente de* pero en los seudocódigos se puede adoptar el operador que corresponda a cualquier lenguaje. Lo importante es que se utilice un operador único y que éste represente *diferente de* sin ambigüedades.

c. Prueba de Escritorio

Tomemos el algoritmo resultante para realizarle la prueba de escritorio tal como lo hemos hecho hasta el momento (y es como se debe hacer) o sea paso a paso. El algoritmo inicia su proceso teniendo en memoria dos variables (*num*, *cd*) y con una pantalla, teóricamente, en blanco.

```
Algoritmo Cuenta_Dígitos
Var
        Entero :num, cd
```

P A N T A L L A	M E M O R I A
	num cd

A continuación el algoritmo da la orden de escribir un título de petición y leer un número entero y guardarlo en la variable num (que también obviamente es entera). Vamos a suponer que el número leído es –523.

```
Inicio
        Escriba "Digite un número entero"
        Lea num
```

P A N T A L L A	M E M O R I A
Digite un número entero -523	num cd -523

A continuación se inicializa la variable cd con el valor 0

```
        cd = 0
```

P A N T A L L A	M E M O R I A
Digite un número entero -523	num cd -523 0

Seguidamente viene el ciclo

```
        Mientras num < > 0
                num = num / 10
                cd = cd + 1
        Fin_mientras
```

En el cual inicialmente se pregunta *si num< >0* (en realidad la pregunta es *Mientras num< >0* que para efectos de la prueba de escritorio se reemplaza brevemente por una decisión). En esta pregunta se sabe que si la respuesta es Verdadera entonces se deberán ejecutar las órdenes que siguen. Como en el caso inicial la variable *num* contiene el valor –523 y este valor es diferente de 0 entonces es la respuesta es Verdadera por lo tanto entramos a realizar las dos instrucciones

```
num = num / 10
cd = cd + 1
```

De manera que en la variable *num* queda almacenado el valor –52 y en la variable *cd* queda almacenado el valor 0.

P A N T A L L A

```
Digite un número
entero
-523
```

M E M O R I A

num	cd
~~-523~~	~~0~~
-52	1

Como ya el valor de la variable *num* no es el mismo que el inicial volvemos al planteamiento del ciclo a realizar la misma pregunta o sea *mientras num< >0* encontrándonos con que el valor almacenado en la variable *num* es –52 y que es realmente diferente de 0. Entonces esto quiere decir que volvemos a ejecutar las órdenes

```
num = num / 10
cd = cd + 1
```

que conforman el cuerpo del ciclo o sea el conjunto de órdenes que se deben ejecuta en caso de que la condición del ciclo sea Verdadera. Al ejecutarlas obtenemos que en la variable *num* queda almacenado el valor –5 y en la variable *cd* queda almacenado el valor 2.

P A N T A L L A

```
Digite un número
entero
-523
```

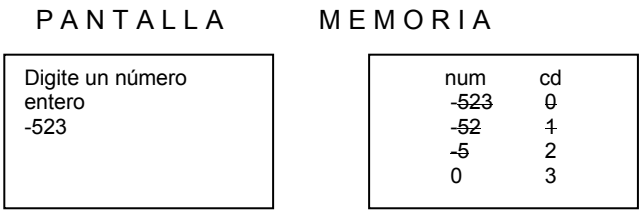
M E M O R I A

num	cd
~~-523~~	~~0~~
~~-52~~	~~1~~
-5	2

Recuerde que cada vez que es cambiado el valor de una variable, el valor anterior se pierde. Por lo tanto en estos momentos los valores que realmente hay son en la variable *num* es –5 y en la variable *cd* es 2. En estas condiciones y habiendo ejecutado el cuerpo del ciclo volvemos a la condición que determina el fin del ciclo. Ante la pregunta *Mientras num< >0* vemos que la variable *num* contiene el valor –5 y que éste es evidentemente diferente de 0 entonces la respuesta es Verdadera por lo tanto volvemos a ejecutar (una vez mas) las órdenes

```
num = num / 10
cd = cd + 1
```

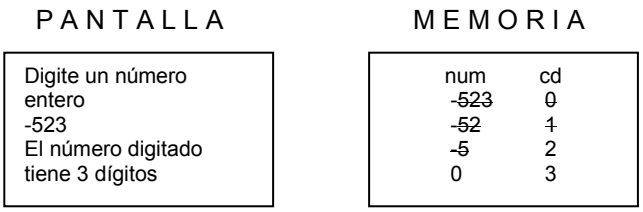
Con lo cual quedará almacenado en la variable num el valor 0 y en la variable cd el valor 3.



Con lo cual después de haber ejecutado las órdenes que conformaban el cuerpo del ciclo volvemos al inicio del mismo a verificar de nuevo la condición y vemos que esta vez ante la pregunta de que el contenido de *num* sea diferente de 0 la respuesta es Falso. Por lo cual ya no volvemos a ejecutar las órdenes que conforman el cuerpo del ciclo sino que pasamos directamente a ejecutar la orden que le sigue al ciclo o sea la que está después del finalizador *Fin_Mientras*, es decir, la orden

Escriba "El número digitado tiene " cd " dígitos"

Con lo cual saldría en pantalla el título *El número digitado tiene 3 dígitos* lo cual es cierto. Tenga en cuenta que cuando una variable aparece inmersa en un título pero por fuera de las comillas dobles esto quiere decir que saldrá en pantalla el contenido de la variable.



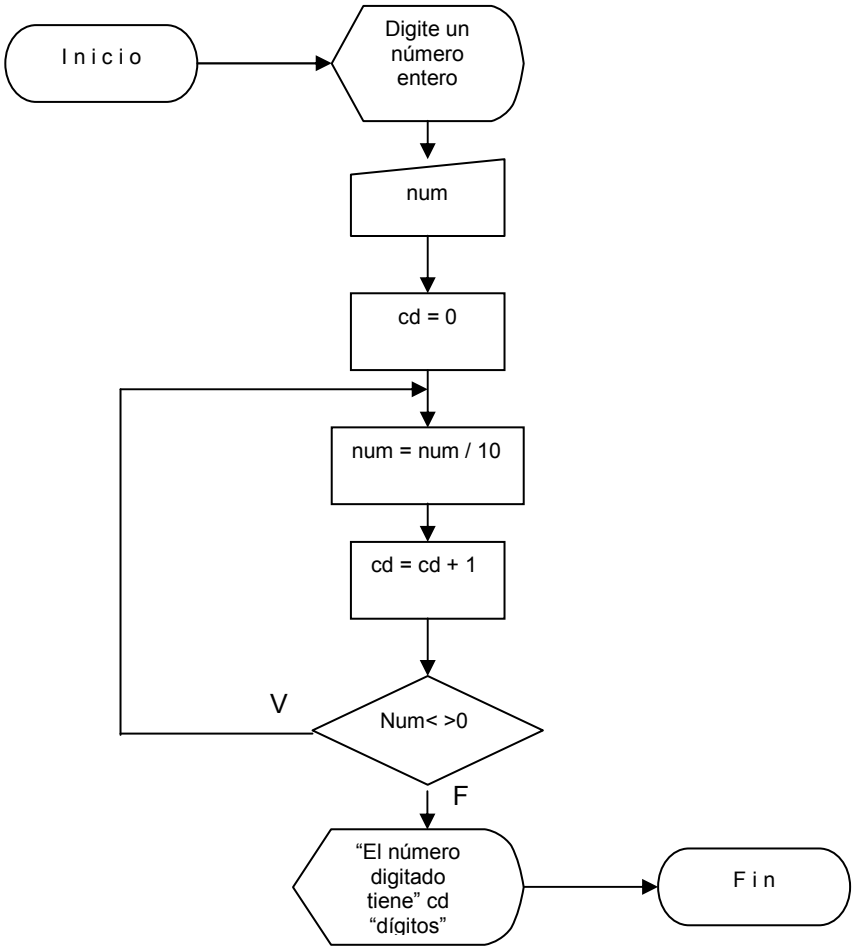
Después de lo cual lo único que nos queda es ejecutar la última orden y es finalizar el algoritmo

Fin

Ahora bien, recuperando nuestra condición de seres humanos, podemos ver que lo que está escrito en pantalla es Verdad, es decir, si hemos digitado el número –523 el algoritmo nos ha dicho algo cierto y es que este número tiene 3 dígitos. Como el objetivo inicial era leer un número y determinar cuántos dígitos tenía y este algoritmo lo hace correctamente entonces podemos decir que está bien.

2. Técnicas de Representación

a. Diagrama de Flujo



No olvide que lo importante es que se claramente cuál es el flujo de la solución.

b. Diagramación Rectangular Estructurada

Algoritmo Cuenta_Dígitos	
“Digite un número entero”	
Lea num	
cd = 0	
Mientras num<>0	
	num = num / 10
	cd = cd + 1
“El número digitado tiene” cd “dígitos”	
F i n	

c. Seudocódigo

```
Algoritmo Cuenta_Dígitos
Var
        Entero :num, cd
Inicio
        Escriba "Digite un número entero"
        Lea num

        cd = o
        Mientras num < > 0
                num = num / 10
                cd = cd + 1
        Fin_mientras
        Escriba "El número digitado tiene " cd "dígitos"
Fin
```

3. Transcripción o Codificación

a. Basic

```
Print "Digite un número entero"
Input num

cd =0
while num < > 0
        num = num / 10
        cd = cd + 1
wend
Print "El número digitado tiene", cd, " digitos"
```

b. Pascal

```
Program Cuenta_Digitos;
Var
        num, cd : integer;
begin
        writeln(' Digite un número entero ');
        readln(num);

        cd = 0
        while num < > 0 do
        begin
                num := num div 10;
                cd := cd + 1;
        end;
        writeln(' El número digitado tiene ', cd, ' dígitos');
end.
```

c. C

```
#include <stdio.h>
void main( )
```

```
{
        int num, cd;
        cout << “Digite un número entero”;
        cin >> num;

        cd = 0;
        while ( num ! = 0 )
        {
                num = num / 10;
                cd ++;
        }

        cout << “El número digitado tiene “<< cd << “ dígitos”;
}
```

d. Cobol

```
        IDENTIFICATION DIVISION.
        PROGRAM_ID. CUENTA-DIGITOS.

        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. CLON.
        OBJECT-COMPUTER. CLON.

        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 NUM          PIC 9999.
        01 CD           PIC 9999.

        PROCEDURE DIVISION.
        INICIO.
                DISPLAY “DIGITE UN NUMERO ENTERO “ LINE 10 COL 10
                ACCEPT NUM LINE 10 COL 50

                COMPUTE CD = 0
                PERFORM CONTEO UNTIL NUM = 0.
                DISPLAY “EL NUMERO DIGITADO TIENE “     LINE 14 COL 10
                DISPLAY CD                              LINE 14 COL 40
                DISPLAY “DIGITOS”                       LINE 14 COL 50
                STOP RUN.

        CONTEO.
                COMPUTE NUM = NUM / 10
                ADD 1 TO CD.
```

Es importante aclarar que el objetivo buscado en este capítulo es que usted tenga unos ejemplos claros de todos el proceso (al menos el que se hace en el papel) que se ha de tener en cuenta cuando se va a desarrollar un algoritmo y de él se va a derivar un programa. También me parece muy importante aclarar que en este libro en ningún momento se busca que usted sepa de lenguajes de programación pero lo que si es importante, por ahora, es que usted vea que la codificación en cualquier lenguaje de programación es fácil dado que sencillamente se trata de

reemplazar algunas palabras clave y utilizar una estructura establecida por los creadores del lenguaje. Lo realmente importante detrás de un programa es la lógica que se haya utilizado para el desarrollo del algoritmo.