Capítulo 11

Funciones

Concepto General

Ningún concepto a nivel de la programación es mas importante que el concepto de Función. Sin temor a equivocarme puedo garantizarle que la Función es lo que podríamos llamar la gran "vedette" de la programación. Esto por ahora no será muy significativo para usted pues tendrá que depurar muy bien el concepto de Función como tal para que vea que tan sencillo se vuelve programar basados en esta poderosísima herramienta.

Antes de entrar a definir qué es una función quiero exponerle un breve texto para que lo lea detenidamente:

Receta para preparar unos Huevos Fritos

Ingredientes:

- *2 Huevos Crudos*
- *¼ de cucharadita de sal*
- *Una cucharadita de aceite*

Preparación:

Colóquese una cacerola a calentar en medio. Échese la cucharadita de aceite hasta cuando esté bien caliente. Quiébrense los huevos y vacéese su contenido en la cacerola. Esperar hasta cuando

la clara esté bien blanca. Echar por encima del huevo de manera que quede repartida uniformemente la sal que se incluyó en los ingredientes.

Resultado:

Unos ricos huevos en cacerola que pueden ser acompañados con un pan.

Estoy absolutamente seguro que en este momento del libro usted estará un poco desconcertado pues no sabrá el porqué he comenzado este capítulo explicando algo que cada uno de nosotros de alguna manera sabe como es preparar unos huevos fritos. Antes de entrar un poco mas en materia no está de más reescribir la receta anterior de la siguiente forma

Receta para preparar unos Huevos Fritos

Ingredientes:

- *2 Huevos Crudos*
- *¼ de cucharadita de sal*
- *Una cucharadita de aceite*

Preparación:

Colóquese una cacerola a calentar en medio.
Échese la cucharadita de aceite hasta cuando esté bien caliente.
Quiébrense los huevos y vacéese su contenido en la cacerola.
Esperar hasta cuando la clara esté bien blanca.
Echar por encima del huevo de manera que quede repartida uniformemente la sal que se incluyó en los ingredientes.

Resultado:

Unos ricos huevos en cacerola que pueden ser acompañados con un pan.

Solo es un pequeño cambio como para que de alguna manera se vaya asociando con lo que hasta el momento hemos visto. De acuerdo a todo lo anterior entonces tenemos la *Receta para preparar unos Huevos Fritos* o, dicho mejor en nuestros términos, *el Algoritmo para preparar unos Huevos Fritos.*

Acerca de esta Receta (o Algoritmo) vamos a plantear algunas precisiones:

a. La receta tiene un nombre específico que, en condiciones normales, no lo tiene ninguna otra receta. Esto quiere decir algo que aunque parece redundante será muy importante plantearlo desde ahora y es que la única receta que debe *llamarse Receta para preparar unos Huevos Fritos* es la que nos enseña a preparar unos huevos fritos. Obvio, cierto..?
b. Para la preparación efectiva de esta receta podemos ver que se necesitan unos ingredientes sin los cuales sería imposible realizarla, por lo menos, tal como está explicada en el texto de la

misma. Estos ingredientes serán diferentes por cada receta aunque puedan existir recetas que tengan los mismos ingredientes y su diferencia radicará en el texto de la misma o sea en su preparación real. Es importante anotar que los ingredientes tienen unas características en tamaño, en peso y en medida. Igualmente es útil anotar que los ingredientes no se explican detalladamente, por ejemplo: Se supone, en esta receta, que estamos hablando de huevos de gallina.

c. La preparación no es mas que un conjunto de pasos secuenciales y ordenados que nos permiten lograr el objetivo inicial (que en este caso era *preparar unos Huevos Fritos*). Estos pasos no se pueden alterar ya que hacerlo no nos garantizaría que el resultado final fuera el esperado.
d. El resultado final de la receta debe lograr el objetivo inicial planteado y ese resultado final es el que puede ser utilizado para todo aquello en lo cual consideremos que puede sernos útil.

Bien, de acuerdo a lo dicho anteriormente, nuestra receta también podríamos haberla escrito de la siguiente forma para estar acorde con las normas algorítmicas de seudocódigo planteadas a lo lardo de este libro.

Receta para preparar unos Huevos Fritos *(2 Huevos Crudos, ¼ de cucharadita de sal, Una cucharadita de aceite)*

```
Inicio

        Colocar cacerola a calentar en medio
        Echar la cucharadita de aceite

        Mientras el aceite no esté bien caliente
                Esperar
        Fin_Mientras

        Quebrar los huevos (uno a uno)
        Vaciar su contenido en la cacerola

        Mientras la clara no esté bien blanca
                Esperar
        Fin_Mientras

        Echar la sal por encima del huevo
        Verificar que quede repartida uniformemente
Fin
```

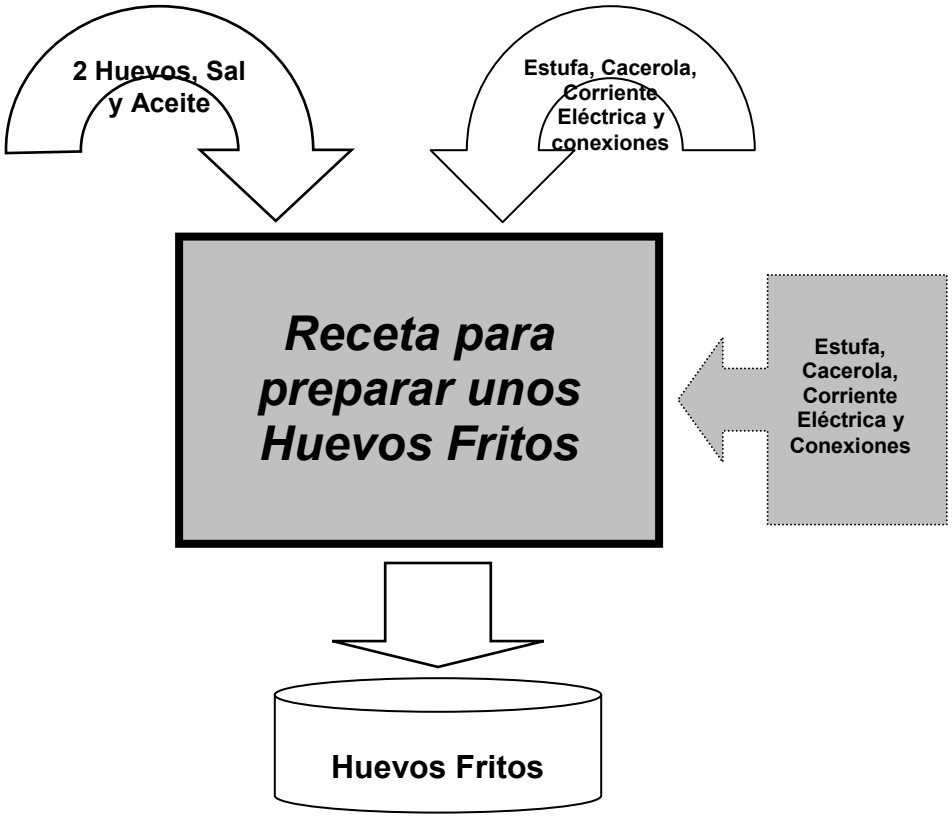
Resultado:

Unos ricos huevos en cacerola que pueden ser acompañados con un pan.

Note usted que vista así la Receta inicial puede ser vista, ahora sí oficialmente, como un algoritmo. Un algoritmo que tiene unas características:

a. Tiene un nombre específico y único
b. Tiene unos ingredientes que son los que permiten que la receta se realice
c. Tiene un conjunto de pasos que son la receta en sí
d. Tiene un resultado final que es el objetivo de la receta

Siendo así podríamos representar también este algoritmo con el siguiente diagrama de bloques



Basados en lo concluido con esta receta podemos escribir la siguiente definición:

Función

Conjunto de órdenes que:

1. Permite lograr un objetivo
2. Tiene un nombre único identificativo
3. Puede necesitar parámetros para lograr dicho objetivo
4. Nos puede retornar un resultado que deberá concordar con el objetivo propuesto
5. Puede ser manejado como una sola unidad

Conociendo usted la estética y características de este libro, sé que debe estarse preguntando el porqué esta definición, que pareciera ser una mas en la teoría de la programación, se ha colocado con tanta rimbombancia y de manera tan destacademente notoria.

No es un "decoradito" que le coloqué al libro para que usted no se me fuera a aburrir. No. Es precisamente la definición mas importante que tiene la programación pues no en vano le dije al inicio de este capítulo que el concepto de Función se podía considerar como la gran "Vedette" de la programación.....voy a explicarle el porqué.

Problemas Reales de la Programación

Cuando se desarrolla un programa el programador se enfrenta a varios problemas entre los cuales se encuentran los errores que en uno de los capítulos anteriores le expliqué. Y a hablábamos pues de los errores de sintaxis y los errores de precaución. En ese momento le decía yo que los errores mas difíciles de encontrar eran los errores lógicos y que dichos errores solo se podían enfrentar desarrollando una excelente Prueba de Escritorio. Sin embargo no es fácil concebir una prueba de escritorio cuando hemos desarrollado un algoritmo que tiene mas de 1000 líneas...(Ojo!!! Mas de mil líneas) pues de manera muy fácil podemos "perdernos" en la ejecución de dicha prueba o sencillamente hacerla mal.

También cabe anotar que cuando se desarrolla un programa extenso, muchas veces nos encontramos con que parte de el puede sernos útil en otro programa pero con frecuencia lo que nos toca hacer es volver a copiar las líneas que creemos pueden tener cabida en el otro algoritmo. De esta manera podemos resumir los problemas reales de la programación en 3 tópicos:

a. Necesidad de simplificar la idea general para lograr un objetivo

Cuando usted se enfrenta a un objetivo, cualquiera que este sea, no sabemos en primera instancia cuántas líneas (u órdenes) va a tener nuestro algoritmo. Cuando hemos escrito un algoritmo que, teóricamente suponemos logrará dicho objetivo, muchas veces nos encontramos con la gran sorpresa de que hemos desarrollado una solución muchísimo mas larga de lo que esperábamos. Basado en esto concebir una sola idea que sea solución para lograr un objetivo es mas complejo que concebir la misma idea pero fragmentada o dividida en pedacitos funcionales dado que cuando algo complejo se subdivide en fracciones simples y no se pierde ninguna característica inicial entonces eso complejo se convierte en algo igualmente simple.

b. Simplificación de la Prueba de Escritorio y por lo tanto detección muy sencilla de errores

Dado el cuestionamiento anterior podemos ver que, si tenemos un algoritmo muy extenso, entonces nuestra prueba de escritorio también será igualmente extensa y el riesgo de que nos quede mal hecho o de que omitamos pasos o de que cometamos algún error en la ejecución

simulada es muy alto y por lo tanto en la medida en que un algoritmo es extenso se reduce la confiabilidad de la prueba de escritorio no porque la teoría falle sino porque en la realidad no es tan fácil realizarle pruebas de escritorio a algoritmos de muchas órdenes y a la vez de muchas variables.

c. Reutilización del Código Fuente

Siempre que desarrollamos un algoritmo nos encontramos con que parte de lo que hicimos nos puede servir para otro algoritmo pero casi siempre nos toca volver a copiar lo que hicimos y de esa manera incorporarlo en el nuevo algoritmo. Una de las necesidades mas grandes que se comenzaron a sentir fue la necesidad de poder utilizar en posteriores programas lo que en un momento dado se estuviera haciendo, sin necesidad de hacerle ningún cambio. Esta necesidad es lo que ha llevado a desarrollar la teoría de la programación buscando nuevas y mejores formas de reutilizar el código fuente o sea lo que nosotros escribimos como solución y que hemos llamado algoritmos computacionales.

Estos tres problemas son los que llevaron a los diseñadores de Lenguajes de Programación a construir una “célula” fundamental, un núcleo de trabajo que permitiera a los programadores superar los tres grandes problemas que tiene la programación (o por lo menor facilitarle el camino para ello). Esta célula es lo que se llama Función. Es importante anotar que en algunos lenguajes de programación tiene otros nombres pero la esencia es la misma y no es mas que un conjunto de instrucciones que llevan un nombre único, que puede recibir parámetros (o ingredientes) para lograr su objetivo y que nos puede retornar un valor.

Macro Algoritmo

Este concepto ha permitido colocar a las funciones en el puesto que les corresponde pues con él podemos desarrollar algoritmos en donde, basados en Funciones, llegamos claramente a

a. Simplificar el algoritmo
b. Simplificar la Prueba de Escritorio y por lo tanto encontrar fácilmente errores lógicos
c. Reutilizar el Código Fuente

Qué es pues el Macro Algoritmo..? Es un algoritmo dividido en unidades funcionales en donde cada una de ellas logra un pequeño objetivo dentro de toda la solución y en conjunto dichas unidades logran el Objetivo General. Vamos a verlo con un ejemplo:

Ejemplo.- Almacenar 10 números enteros en un vector y mostrar cual es el mayor de los números leídos.

Ajustándonos a lo que hasta antes de este capítulo se había explicado un posible algoritmo solución a este objetivo podría ser el siguiente:

```
Algoritmo Buscar_Mayor
Variables
        Entero :        V( 10 ),                    // Almacenará los 10 datos enteros
                        Ind,                        // Servirá como Indice del Vector
                        Mayor                       // Almacenará el mayor de los números
                                                    // leídos
Inicio
        Escriba "Digite 10 números enteros"         // Solicita los 10 datos enteros

        Ind = 1                                     // Comienza en la primera posición
        Mientras Ind < = 10                         // Mientras no haya llegar a la última
                                                    // posición del vector
                Lea V ( Ind )                       // Lea un entero y guárdelo en la
                                                    // posición Ind del Vector
                Ind = Ind + 1                       // Incremente la posición
        Fin_Mientras                                // Fin del ciclo

        Mayor = V ( 1 )                             // Inicialice la variable Mayor con el
                                                    // contenido del Vector en la primera
                                                    // posición

        Ind = 2                                     // Comience en la segunda posición

        Mientras Id < = 10                          // Mientras no haya llegado a la última
                                                    // posición del vector
                Si V ( Ind ) > Mayor                // Si el contenido del vector en la
                                                    // posición Ind es mayor que el
                                                    // contenido de la variable Mayor
                        Mayor = V ( Ind )           // Entonces V ( Ind ) será el nuevo
                                                    // mayor
                Fin_Si                              // Fin de la decisión
        Ind = Ind + 1                               // Pase a la siguiente posición del
                                                    // vector
        Fin_Mientras                                // Fin del ciclo

        Escriba "El mayor número leído es ", Mayor  // Muestre el resultado

Fin                                                 // Fin del algoritmo
```

Usted podrá notar que este es un algoritmo común y corriente. Ahora bien vamos a suponer tres cosas que nos serán muy útiles en este ejercicio:

1. Supongamos que este es un algoritmo muy extenso
2. Supongamos que este es un algoritmo muy complejo
3. Supongamos que este es un algoritmo exageradamente útil

Note usted que este algoritmo de ejemplo, al cual es necesario que le hagamos las suposiciones aquí expuestas, se divide fundamentalmente en tres partes: En la primera parte se leen 10 datos

enteros y se almacena cada uno en un vector, en la segunda parte se busca cuál es el número mayor de entre los números leídos y en la tercera parte se muestra el resultado final. Sé que coincidimos en el análisis de este algoritmo y en su subdivisión en estas tres partes tan lógicas. Bueno pues precisamente ese es el Macro Algoritmo. Escribámoslo técnicamente

```
Macro Algoritmo Buscar_Mayor
Inicio
        Lectura_de_Datos
        Búsqueda_del_Mayor
        Muestra_de_Resultado
Fin
```

Es evidente que nuestro algoritmo se simplificó muchísimo (y mucho mas si partimos de las suposiciones hechas inicialmente). Ahora bien cabría la pregunta *Qué contiene cada uno de estos procesos..?* Pues veamos cómo quedaría el algoritmo completo.

Algoritmo Buscar_mayor

```
Variables
        Entero :        V ( 10 ),
                        Ind,
                        Mayor
Función Principal
Inicio
        Lectura_De_Datos
        Búsqueda_del_Mayor
        Muestra_de_Resultado
Fin

Función Lectura_De_Datos
Inicio
        Escriba "Digite 10 números enteros"

        Ind = 1
        Mientras Ind < = 10

                Lea V ( Ind )

                Ind = Ind + 1
        Fin_Mientras
Fin

Función Búsqueda_del_Mayor
Inicio

        Mayor = V ( 1 )
        Ind = 2

        Mientras Id < = 10
                Si V ( Ind ) > Mayor
                        Mayor = V ( Ind )
```

```
            Fin_Si
        Ind = Ind + 1
        Fin_Mientras
Fin

Función Muestra_del_Resultado
Inicio
        Escriba "El mayor número leído es ", Mayor
Fin
```

Estoy seguro que usted que usted debe estar pensando que el algoritmo no se acortó sino que se alargó y puede que tenga razón si lo miramos en la cantidad de líneas que tienen las dos versiones del mismo algoritmo. Quiero exponerle algunas reflexiones acerca de esta versión del algoritmo:

a. Note que existe una Función Principal cuyo objetivo es permitir la coordinación de los llamados a las demás funciones.
b. También notará que este algoritmo está organizado en unas unidades funcionales donde cada una cumple un pequeño objetivo frente al objetivo general. Como puede ver usted la Función Principal se encarga de llamar a las demás funciones, la ***Función Lectura_De_Datos*** cumple con el objetivo de leer los datos y almacenarlos en el Vector, la ***Función Búsqueda_del_Mayor*** cumple con el objetivo de encontrar cuál es el mayor de los números almacenados en el vector y la función ***Muestra_del_Resultado*** se encarga de entregarnos el valor solicitado.
c. Esta forma de escribir el programa nos permite (mirando la función principal) entender mucho mas fácil la idea general de solución de este algoritmo y por lo tanto eso nos simplifica el camino al momento que de alguna forma le queramos hacer ajustes, mejoras o correcciones.
d. Realizarle una prueba de escritorio al algoritmo inicial (que no se olvide que estamos suponiendo que es un algoritmo extenso y muy complejo) se reduce ahora a realizar tres pruebas de escritorio sencillas.

   La primera prueba será a la ***Función Lectura_De_Datos*** que como ya sabemos busca como objetivo leer 10 datos enteros y almacenarlos en un Vector. Si al realizar dicha prueba vemos que SOLO esta función cumple su pequeño objetivo entonces no tendremos que preocuparnos mas de ella.

   La segunda prueba de escritorio será a la ***Función Búsqueda_del_Mayor*** cuyo objetivo es encontrar el mayor número almacenado en un Vector de 10 posiciones enteras. Si al realizar esta prueba vemos que la función cumple con este objetivo entonces no tendremos que preocuparnos mas por ella.

   La tercera prueba (y en este caso la mas sencilla) será a la ***Función Muestra_del_Resultado*** cuyo objetivo será mostrar en pantalla un dato entero almacenado en una variable llamada Mayor. Si al realizar la prueba de escritorio vemos que realmente se cumple el objetivo entonces no tendremos que preocuparnos mas ni por la función ni por el objetivo al igual que en todos los casos.

e. Como puede notar si al realizarle independientemente las respectivas pruebas de escritorio a estas tres funciones cada una cumple su objetivo (independientemente) entonces ya habremos desarrollado el algoritmo completo pues la unión de las tres es lo que nos permite lograr el objetivo general del mismo.
f. Si alguien nos dijera que nuestro algoritmo no está funcionando bien o sea que no cumple con el objetivo general de encontrar el número mayor de entre los 10 números leídos entonces esto querrá decir, sin lugar a dudas, que el error tiene que estar en la función que se encarga de seleccionar el mayor dado que este proceso se realiza solo en esta función y no en ninguna de las otras dos.

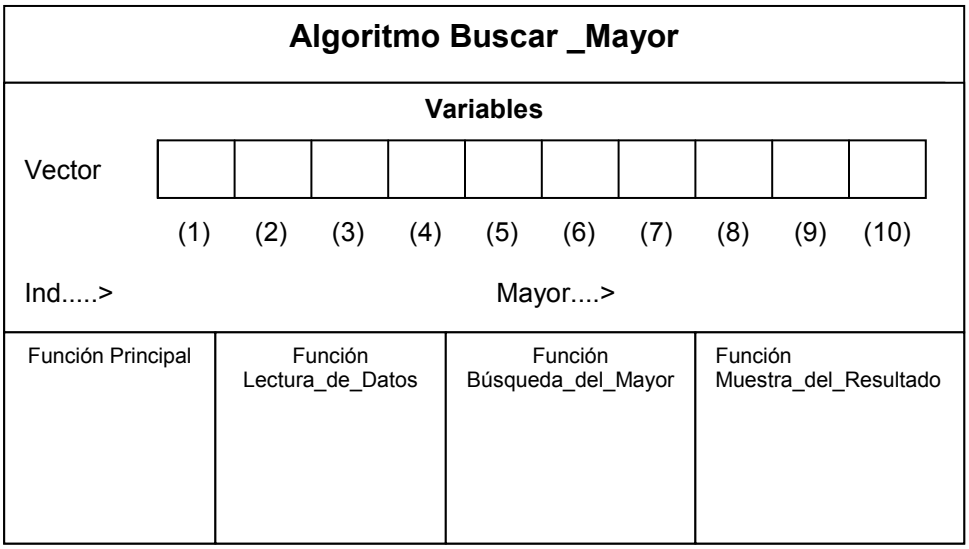
g. Si nos encontramos con que al ejecutar el algoritmo al final nos muestra en pantalla que el número mayor es un número que estamos seguros que no estaba entre los número leídos originalmente entonces esto querrá decir que el error con toda seguridad está en la lectura de datos y que, seguramente, los datos que se están almacenando en el vector no corresponden a los datos leídos.

h. Si vemos que no nos muestra ningún resultado nuestro algoritmo al momento de ejecutarlo entonces esto quiere decir que el error tiene que estar en la ***Función Muestra_de_Resultado***.

i. Si todas las pruebas de escritorio han sido exitosas entonces la Función Principal estará bien pues es la que llama a las otras tres funciones y todo lo que tendremos que revisar es el orden de los llamados.

No podemos desconocer todas las ventajas que surgen cuando utilizamos el concepto de función para simplificar la solución de un problema y hacerlo mucho mas sencillo en su concepción y su posterior ejecución. Es importante anotar que cuando se habla aquí de ejecución se hace referencia al caso en el cual el algoritmo y se encuentre debidamente codificado y transcrito en un Lenguaje de Programación.

Utilizando un diagrama de bloques podríamos representar el algoritmo planteado de la siguiente manera:



Como se puede notar en un programa hecho con funciones el esquema gráfico para el desarrollo de la prueba de escritorio cambia un poquito. Sé que usted se estará preguntando qué podría ir dentro del cajoncito de cada función. Precisamente vamos a entrar en un concepto muy interesante que nos va a permitir hacer mucho mas flexible y mucho mas sencilla la programación.

Variables Globales y Variables Locales

Hasta el momento hemos hablado de variables en general y sencillamente las hemos definido al inicio de cada algoritmo y comenzamos a trabajar con ellas y listo. Ahora vamos a tecnificar un poco mas la definición de las variables para que podamos llegar a construir funciones realmente flexibles y reutilizables.

Una variable **Global** es aquella que se declara por fuera de cualquier función y por lo tanto puede ser reconocida en cualquier función. En el ejemplo que tratamos en este momento podemos ver que las variables *Vector (10), Ind* y *Mayor* son variables globales porque están declaradas por fuera de cualquier función (incluyendo la función principal). Por eso estas variables las utilizamos indiscriminadamente en cualquiera de las funciones.

Una variable **Local** es aquella que se declara dentro de alguna función y solo puede ser reconocida en esa función. En el ejemplo aún no hemos comenzado a utilizar variables locales. Esta definición implica que una función puede tener internamente declarada una variable con un nombre determinado y otra función también puede tener internamente declarada otra variable con el mismo nombre y no habrá ningún error ya que cada variable es diferente y es reconocida en la función en donde se declare. Eso es como María la de los Díaz y María la de los Pérez.

Supongo que usted hasta este momento del libro ya ha encontrado que es muy importante ver lo que se explica con un ejemplo. Por eso vamos a desarrollar un ejemplo sencillo en el cual se va a hacer necesario que usted se imagine que es un algoritmo muy complejo y difícil ya que solo así le encontrará, por ahora, sentido a muchos de los conceptos nuevos que aquí le presento.

Ejemplo

Leer dos números enteros y mostrar el resultado de su suma.

Clarificación del Objetivo

Como puede ver el objetivo es muy sencillo y bien podría solucionarse de la siguiente forma:

```
Algoritmo Suma_Sencilla
Variables
        Entero :        Num1,
                        Num2,
                        Res
```

```
Inicio
        Escriba "Digite dos datos enteros"
        Lea Num1, Num2

        Resultado = Num1 + Num2

        Escriba Resultado
Fin
```

De resolverlo así pues no tendría mucho sentido extender esta explicación pero lo importante es que por ahora usted suponga que este es un algoritmo complejo pues basado en su desarrollo sobre funciones vamos a explicar qué son las variables globales y locales. Vamos pues a desarrollar este mismo algoritmo (o sea a leer dos números enteros y mostrar su suma) pero utilizando funciones.

Algoritmo

Algoritmo Suma_con_Funciones

Funcion Principal

```
Variables
        Entero :                                // Estas son variables locales para la
                                                // Función Principal  lo cual quiere decir
                                                // que pueden ser utilizadas solo dentro
                                                // de ella pues solo allí serán
                                                // reconocidas
                Num1,                           // Almacenará uno de los números
                Num2,                           // Almacenará el otro de los números
                Res                             // Almacenará el resultado de la suma
Inicio
        Escriba "Digite dos números enteros"    // Solicita los dos enteros
        Lea Num1, Num2                          // y los lee

        Res = Sumar ( Num1, Num2 )              // El contenido de la variable Res
                                                // será igual a lo que retorne la función
                                                // Sumar a la cual se le han enviado
                                                // dos parámetros (como quien dice dos
                                                // ingredientes) que deberá sumarlos


        Escriba Res                             // Escriba el resultado de la suma
Fin                                             // Fin de la función principal

Funcion Sumar ( Entero a, Entero b)             // Función Sumar que para lograr su
                                                // objetivo siempre va a necesitar dos
                                                // datos enteros

Variables
        Entero : Res                            // Variable que almacenará la suma de
                                                // los dos datos (ingredientes) recibidos
                                                // por la función Sumar
Inicio
```

```
    Res = a + b                    // Se le asigna a la variables Res
                                   // la suma del contenido de a mas el
                                   // contenido de b

    Retorne ( Res )                // Se le ordena  a la función que
                                   // devuelva el valor almacenado en la
                                   // variable local Res
Fin                                // Fin de la función Sumar
```

Sé que aún puede tener la inquietud del porqué he complicado tanto un algoritmo tan sencillo pero no se preocupe, siga leyendo y verá que el objetivo es puramente pedagógico y busca precisamente que usted tenga muy claro cada uno de los elementos que intervienen en una función y el manejo de variables. También estoy seguro de que en este instante usted se sentirá como si no supiera absolutamente nada de programación y como si todo lo que hasta el momento ha estudiado se hubiera desvanecido. No se preocupe, a continuación vamos a desarrollar (paso a paso) la prueba de escritorio detallada y notará que lo que aquí presentamos en este momento aparentemente confuso no es mas que el pináculo de aplicación de todo lo que hasta el momento hemos visto.

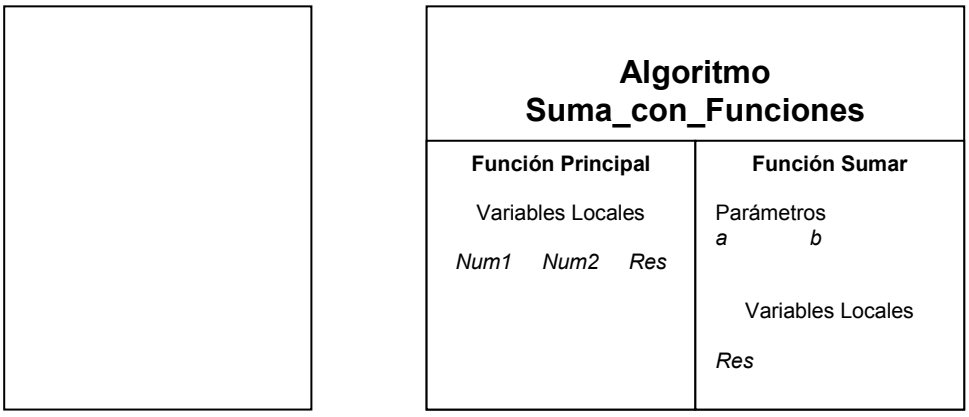
Prueba de Escritorio

Como puede ver nuestro algoritmo no tiene variables globales debido a que todas las variables que se involucran en él están declaradas dentro de alguna función (no se olvide que la función principal no es mas que otra función con la única diferencia que es la primera que se va a ejecutar). Vamos a tener pues en la memoria dos funciones: la **Función Principal** y la **Función Sumar**, cada una con sus respectivas variables.

Algoritmo Suma_con_Funciones

P A N T A L L A

M E M O R I A



Como puede ver ya nuestra memoria no es, como hasta el momento había sucedido, un espacio solo para unas variables y no mas. Ahora la memoria se convierte en el área en donde convergen Funciones, Variables Locales y Variables Globales (si las hay) y nuestro objetivo con las pruebas de escritorio es manejar la memoria tal como lo haría el computador.

Aunque haciendo honor a esta última frase que acabo de escribir, debo admitir que inicialmente en la memoria del computador solo existe es la función principal y que las demás se van colocando en ella (junto con sus variables) en la medida en que se vayan llamando. Iniciamos pues nuestro algoritmo solicitando dos números enteros y almacenándolos el primero en la variable Num1 y el segundo en la variable Num2. Vamos a suponer que dichos números son el 15 y el 18.

Funcion Principal

Variables
Entero : Num1,
Num2,
Res
Inicio
Escriba "Digite dos números enteros"
Lea Num1, Num2

P A N T A L L A

Digite dos números enteros

15
18

M E M O R I A

Algoritmo Suma_con_Funciones

Función Principal	**Función Sumar**
Variables Locales *Num1* *Num2* *Res* *15* *18*	Parámetros *a* *b* Variables Locales *Res*

Como puede notarse y tal como se había explicado el primer número (15) quedó almacenado en la variable *Num1* y el segundo número (18) quedó almacenado en la variable *Num2*. A continuación en nuestro algoritmo aparece la orden según la cual en la variable *Res* se almacenará el valor que retorne la función *Sumar* a la cual se le envían como parámetros (o ingredientes para que logre su objetivo) los contenidos de cada una de las variables *Num1* y *Num2*.

Res = Sumar (Num1, Num2)

Al hacer el llamado a la función *Sumar* entonces se activa esta función y se le envían como parámetros los contenidos de las variables *Num1* y *Num2* que se almacenan respectivamente en las variables *a* y *b*. En estos momentos se suspende momentáneamente la ejecución de la *Función Principal* debido a que como la variable *Res* es igual al valor que retorne la *Función Sumar* entonces tendremos primero que ejecutarla antes de continuar.

Realmente no es que se suspenda sino que la variable *Res* de la función principal queda a la espera de lo que le retorne la *Función Sumar*. Entonces para ejecutar la función sumar primero almacenamos en la variable *a* el valor que tenía la variable *Num1* y en la variable *b* el valor que tenía la variable *Num2*.

Funcion Sumar *(Entero a, Entero b)*

Variables
    Entero : Res
Inicio
    Res = a + b

    Retorne (Res)

Fin

P A N T A L L A

Digite dos números enteros

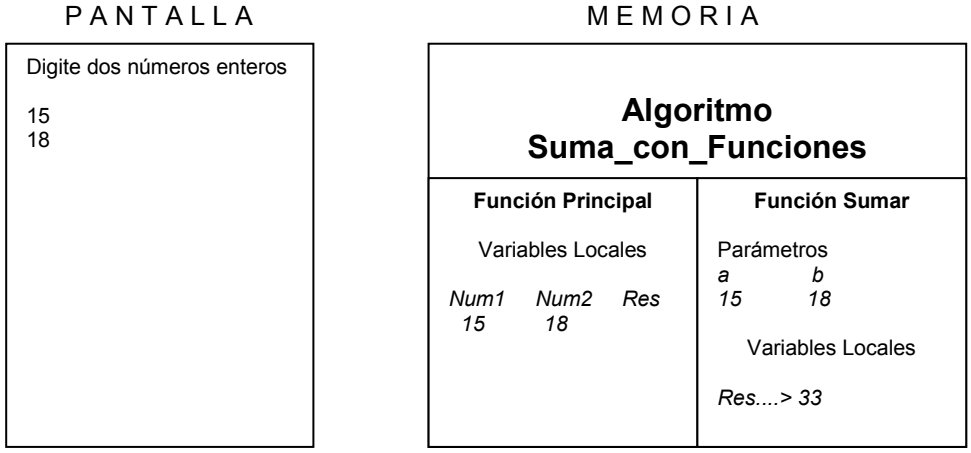
15
18

M E M O R I A

Algoritmo Suma_con_Funciones

Función Principal	**Función Sumar**
Variables Locales *Num1* *Num2* *Res* *15* *18*	Parámetros *a* *b* *15* *18* Variables Locales *Res*

Se puede notar que en la función *Sumar* la primera orden es almacenar en la variable *Res* el resultado de sumar los contenidos de las variables *a* y *b*. Tenga en cuenta que como a almacena el valor original de *Num1* y b almacena el valor original de *Num2* entonces ejecutar la suma *a + b* es igual que ejecutar la suma *Num1 + Num2*. Luego la orden *Res = a + b* se convierte en *Res = 15 + 18* lo cual indica que en la variable *Res* queda almacenado el valor *33*.



Seguidamente encontramos en la función *Sumar* la orden *Retorne (Res)* lo cual indica que la función *Sumar* va a devolver el valor almacenado en *Res* que es *33*. Allí, en el momento de retornar un valor, es donde termina la función *Sumar*. Tenga en cuenta que la variable *Res* que aparece en la *Función Principal* es diferente a la variable *Res* que aparece en la *Función Sumar* debido a que cada una es variable local para su respectiva función (no se olvide que eso es como María la de los Díaz y María la de los Pérez).

Ahora sí podemos volver a la *Función Principal* pues ya la *Función Sumar* ha devuelto un valor. A quién se lo devuelve..? Se acuerda usted que suspendimos la prueba de escritorio de la *Función Principal* pues se necesitaba saber cuál es era el valor que se iba a almacenar en la variable *Res* (local para la función principal)..? Pues bien ahora es el momento de retomar esa orden en donde se había hecho la suspensión y era la orden

Res = Sumar (Num1, Num2)

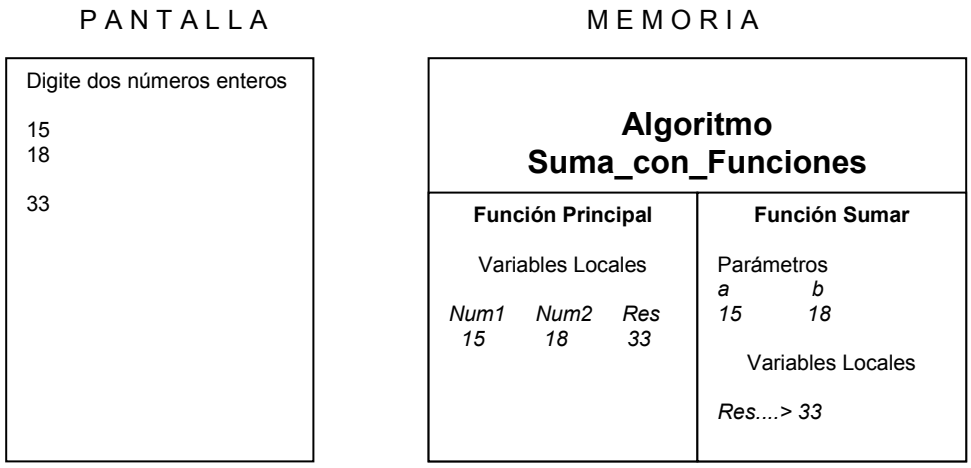
Como la *Función Sumar* al habérsele enviado los parámetros *Num1* y *Num2* (o sea 15 y 18 respectivamente) retornó el valor *33* entonces internamente esta orden se convierte en

Res = 33

Que no es mas que una orden sencilla de asignación (no se olvide que ese cambio se hace internamente). Ya con esto podemos ver entonces que en la variable *Res* de la *Función Principal* queda almacenado el valor *33*. Y como la orden que sigue a continuación dentro de la *Función Principal* es mostrar en pantalla el valor almacenado en la variable *Res*

Escriba Res

Entonces nuestra prueba de escritorio termina realizándolo así



Quedando pendiente solamente dar por terminada esta prueba de escritorio pues hemos llegado al fin de la *Función Principal*

Fin

Ahora sí podemos ver en la pantalla que el valor final es igual a la suma de los dos números iniciales y precisamente ese era el objetivo: *Leer dos números enteros y mostrar su suma*. Sé que a esta altura usted debe estar pensando que no se justificaba hacer tanto para lograr un objetivo tan sencillo (y tiene toda la razón). No se olvide que el objetivo de este ejemplo es solamente didáctico pues ahora si vamos a ver un ejemplo que va a tener mucha utilidad pero antes quiero plantearle algunas reflexiones acerca de este ejemplo.

a. No existe para el caso presentado variables globales.
b. Se utilizan variables locales en cada una de las funciones.
c. Una de las funciones (la Función Sumar) necesita unos parámetros que son como los ingredientes con los cuales esa función puede lograr su objetivo. Recuerde que dicho objetivo era retornar el resultado de sumar dos números entonces lo que necesita para poder cumplirlo son los dos números.
d. Cuando una función retorna un valor entonces allí termina la función.
e. Una función (en este caso la Función Principal) puede llamar a otra (en este caso la Función Sumar).
f. Cualquier función puede llamar a otra función solo se debe tener en cuenta que si una función A llama a una función B entonces dicha función B no debe llamar explícitamente a la función A ya que con su retorno es suficiente.
g. Una función A puede llamar a una función B y esa función B puede llamar a una función C y así sucesivamente. Cada retorno regresará el control del programa a la función llamadora.
h. Un algoritmo puede tener muchas funciones. Todo dependerá del objetivo que se quiera lograr y del conocimiento de esta técnica por parte del programador.
i. El fin de la función Principal es el fin de todo el algoritmo.

Ahora sí, basados en todo lo que hasta ahora hemos visto, vamos a ver un ejemplo que le brinde a usted un panorama mas práctico no sin desconocer la importancia de los ejemplos didácticos que van hasta el momento.

Ejemplo

Leer un número y si es un número par calcular su factorial, mostrarlo en pantalla y determinar si también es par.

Clarificación del Objetivo

Como ya podemos suponer el objetivo de este algoritmo es leer un número entero y sencillamente calcularle su factorial en el caso de que dicho número sea par. Primero que nada tendremos que determinar si el número es par para ello recordemos que un número es par si al dividirlo entre 2 y volverlo a multiplicar por 2 nos da como resultado el mismo número. Este razonamiento está basado en las características y propiedades de la aritmética entera según la cual ninguna operación genera decimales. Siendo así entonces podremos saber que el número 8 es par si cumple con la condición establecida

```
8 = 8 / 2 * 2
```

Resolviendo esta expresión por jerarquía de operadores sabemos que primero que nada se haría la división y luego se realizaría la multiplicación, luego

```
8 = 8 / 2 * 2
8 =   4   * 2
8 =       8
```

Al ver que desarrollando esta expresión obtenemos el mismo valor inicial entonces podemos implementar un algoritmo que concluya que el número 8 es par. Sin embargo hagamos la prueba con un número impar para que usted vea que pasa

```
9 = 9 / 2 * 2
9 =   4   * 2
9 =       8
```

Vemos que el resultado de la expresión, basados en la aritmética entera, no es igual al valor inicial planteado por lo tanto nuestro algoritmo podría concluir que 9 no es par.

Siguiendo con la clarificación del objetivo recordemos que el factorial de un número es la multiplicación sucesiva de todos los enteros comprendidos entre 1 y dicho número, partiendo de que dicho número sea positivo. No está definido el factorial para los números negativos y el factorial de 0 es 1. De esta forma el factorial de 4 es igual al resultado de multiplicar

```
1 x 2 x 3 x 4  =  24
```

El factorial de –6 no está definido y, como dice la definición, el factorial de 0 es 1.

Con estos conceptos ya podemos saber que lo que tenemos que organizar es un algoritmo que nos permita leer un número entero, verificar si es un número par y si lo es entonces calcularle su factorial, al cual también se le debe verificar si es un número par.

Algoritmo

Ya esta vez no vamos a mostrar un algoritmo y no mas, tal como en la mayoría de las veces. Esta vez vamos a explicar el porqué de la solución final. Primero que nada vamos a construir una función que nos permita recibir, como parámetro, un número entero y determinar si es par o impar. Ese *determinar* lo haremos de la siguiente forma: la función retornará un número 1 si el número es par y retornará un número 0 si el número es impar. De esta manera una solución a esta función podría ser

```
Funcion Es_Par ( Entero n )        // Función Es_Par que recibe como parámetro un valor
                                   // entero
Inicio
        Si  n / 2 * 2 = n          // Pregunta si el valor recibido es par
                Retorne ( 1 )      // entonces que retorne un 1
        Sino                       // Sino
                Retorne ( 0 )      // que retorne un 0
        Fin_Si                     // Fin de la decisión
Fin                                // Fin de la función
```

Usted tal vez se preguntará a quién se le retorna ese 1 o ese 0.... Pues muy sencillo, se le retorna a la función que llame a esta función porque es claro que para que esta función se ejecute, otra función la tiene que llamar.

Vamos a hacerle una pequeña prueba de escritorio a esta función. Supongamos que llamamos a la Función Es_Par y se el envía el valor 5, o sea que el llamado es de la siguiente forma

```
Es_Par ( 5 )
```

No se olvide que una función se llama escribiendo sencillamente su nombre. La palabra función se ha colocado en este libro solo para fines didácticos pues es muy importante que a lo largo de él usted vaya reconociendo las verdadera unidades fundamentales de trabajo de un algoritmo.
Siendo ese el llamado entonces la variable *n* se cargará con el número *5* e internamente nuestra función preguntará

```
Si 5 / 2 * 2 = 5
```

Lo cual es *Falso* dado que *5 / 2 * 2* es igual a *4* entonces la función retornará un *0* tal como la habíamos planeado pues se había dicho que debería retornar 0 en caso de que el número no fuera par.

Ahora vamos a probar en el caso de que el llamado a la *Función Es_Par* sea el siguiente

Es_Par (8)

Entonces el parámetro (ingrediente) llamado *n* se cargará con el valor *8* y por lo tanto la función tomará la siguiente decisión

*Si 8 / 2 * 2 = 8*

Lo cual es *Verdadero* pues *8 / 2 * 2* es igual a *8* por propiedades de la aritmética entera. Con esto podemos garantizar que la Función Es_Par nos permite determinar si un número es par o no. En caso de que el número que le enviemos como parámetro sea par, esta función retornará 1 y en caso de que dicho parámetro no sea par entonces nos retornará un 0. Con esto ya no tenemos que preocuparnos por la decisión de si un número es par o no.

Ahora vamos a construir una función a la cual le enviemos un parámetro entero y ella nos retorne el factorial de ese número. Esta función nos retornará un número entero positivo en caso de que el parámetro que le enviemos sea igualmente positivo, nos retornará un número 1 en caso de que el parámetro que le enviemos se un 0 y nos retornará un número –1 en caso de que le enviemos un valor negativo. Como ya sabemos que no existen factoriales negativos eso querrá decir que si al llamar la función, ella nos retorna un –1 es porque el parámetro que le habíamos enviado era negativo y con ello podremos detectarlo. Una posible solución a esta función podría ser

```
Función Factorial ( Entero n )      // Función Factorial que recibirá como parámetro un
                                    // entero
Variables Locales                   // Declaración de las variables locales
        Entero :    Facto,          // Almacenará el valor del factorial
                    Cont            // Permitirá generar los números desde 1 hasta el
                                    // parámetro recibido que es a quien se le debe calcular
                                    // el factorial
Inicio
        Si n < 0                    // Si el parámetro recibido es negativo
                Retorne ( -1 )      // Retorna un –1 que luego será interpretado desde la
                                    // función llamadora

        Facto = 1                   // Inicie la variable Facto en 1
        Cont = 1                    // Inicie la variable Cont en 1
        Mientras Con < = n          // Mientras Cont no haya llegado al número recibido
                                    // como parámetro
                Facto = Facto * Cont // Multiplique sucesivamente todos los enteros
                Cont = Cont + 1     // comprendidos entre 1 y el número recibido como
                                    // parámetro
        Fin_Mientras                // Fin del ciclo Mientras

        Retorne ( Facto )           // Retorne el valor almacenado en la variable Facto
Fin                                 // Fin de la función
```

Vemos a desarrollar una pequeña prueba de escritorio a esta función para ver si cumple con el objetivo planteado que es el de calcular el factorial del número que se el envíe como parámetro. Vamos a comenzar suponiendo que a la función se le llama enviándosele el valor entero –4.

Factorial (-4)

La función *Factorial* recibirá ese número *–4* en el parámetro *n* y procederá a su ejecución que luego de declarar las variables locales *Facto* y *Cont* procederá a preguntar si *n* (o sea el valor recibido como parámetro) es menor que *0*. Como esto es *Verdadero* entonces la función retorna el número –1 y allí termina. No se olvide que apenas en una función se encuentra una orden *Retorne*, la función llega en su ejecución hasta allí. Ya desde la función llamadora analizaríamos el resultado que nos retorne la función Factorial y sabemos que en caso de que retorne el número –1 esto querrá decir que el valor que se le envió era negativo y podremos decirle al usuario que *No están definidos los factoriales de números negativos*.

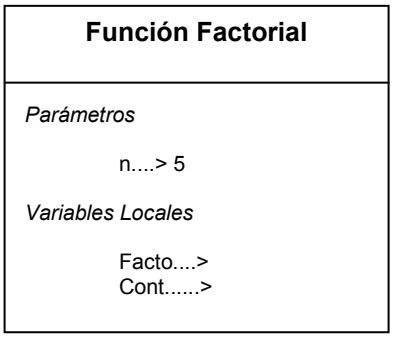
Veamos cuál sería el resultado que retornaría la función Factorial en caso de que el llamado fuera

Factorial (5)

En este caso nuestra prueba iniciaría almacenando este valor 5 en la variable n que lo recibiría y declarando las variables locales de la función

Función Factorial (Entero n)

Variables Locales
Entero : Facto,
Cont

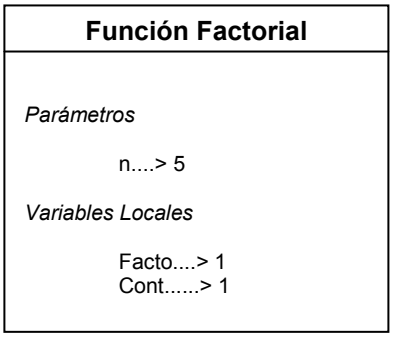


Nuestra función comienza, pues, preguntando

Inicio
Si n < 0
Retorne (-1)

Lo cual es Falso dado que en la variable n está almacenado el valor 5. Por lo tanto la prueba continúa con el resto de la función. Se inicializa la variable Facto en 1 y la variable Cont en 1.

Facto = 1
Cont = 1



Nuestra función continúa con las órdenes

Mientras Cont < = n

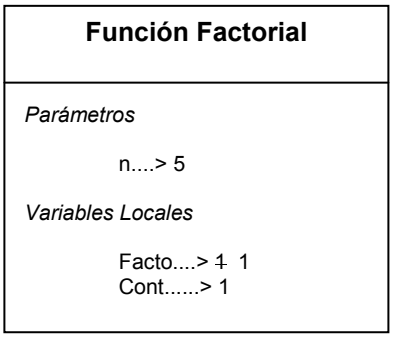
*Facto = Facto * Cont*
Cont = Cont + 1

Fin_Mientras

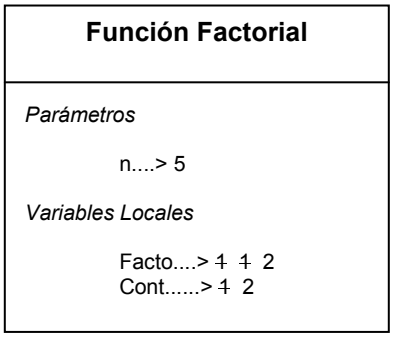
Se pregunta si el contenido de la variable *Cont* (que es 1) es menor que el contenido de la variable *n* (que es 5). Como es *Verdadero* entonces se ejecuta la operación

*Facto = Facto * Cont*

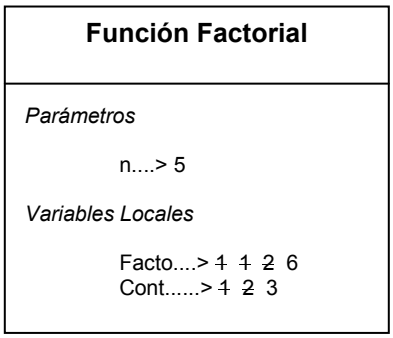
Luego en la variable *Facto* queda almacenado el número 1



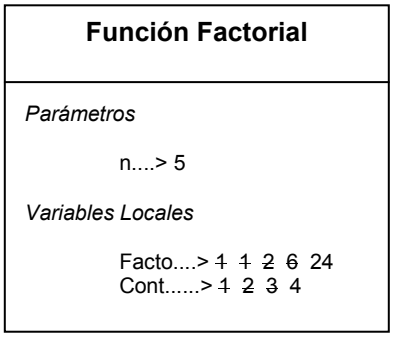
Se incrementa en 1 el contenido de la variable *Cont* y se vuelve a preguntar si dicho valor es menor o igual que el valor almacenado en *n* (que sigue siendo 5). Como es *Verdadero* entonces se vuelve a ejecutar la multiplicación *Facto = Facto * Cont*.



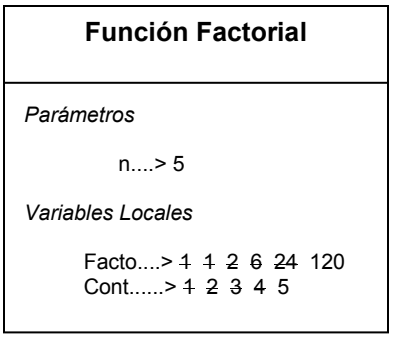
Se vuelve a incrementar en 1 el contenido de la variable *Cont* y volvemos a evaluar la condición del ciclo Mientras. Como el valor almacenado en *Cont* sigue siendo menor que *n* entonces volvemos a hacer la multiplicación *Facto = Facto * Cont*.



Volvemos a incrementar en 1 el contenido de la variable *Cont* y de nuevo se evalúa la condición del ciclo. El valor almacenado en *Cont* sigue siendo menor o igual que el valor almacenado en n por lo tanto se ejecuta de nuevo la operación *Facto = Facto * Cont*.



Se vuelve a incrementar en 1 el contenido de la variable *Cont* (que esta vez ya valdrá 5) y se pregunta de nuevo la condición del ciclo. Como *Cont* sigue siendo menor o igual que n pues ambas almacenarían el valor 5 entonces se ejecuta por última vez la operación que se encuentra en el cuerpo del ciclo o sea *Facto = Facto * Cont*.



Se vuelve a incrementar en 1 el contenido de la variable *Cont* y vemos que como almacenaría un número 6 no se cumple la condición del ciclo o sea de que *Cont* <= *n* pues n sigue valiendo 5 por lo tanto nos salimos del ciclo y ejecutamos la orden que está después del *Fin_Mientras*

Retorne (Facto)
Fin

Que no es mas que retornar el valor almacenado en la variable Facto. En este caso retornaría el valor 120 que efectivamente corresponde al factorial del valor recibido que fue el número 5. En el caso de que el llamado a la función Factorial sea

Factorial (0)

Sencillamente en la variable n se cargará este parámetro, se inicializará a variable *Facto* en 1 y la variable *Cont* en 1. Cuando se llegue al ciclo a la primera vez que se evalúe la condición del mismo (o sea *Cont* <= *n*) como *Cont* vale 1 y n vale 0 entonces la condición será *Falsa* y pasaremos a la instrucción que está después del *Fin_Mientras*. Esta instrucción ordena retornar el valor almacenado en *Facto* que para este caso sería el número 1. Con ello se cumpliría también con esta función que cuando se le envíe el número 0 ella retorna el número 1 como su factorial.

Podemos pues concluir que la función Factorial así como está concebida realmente nos permite que le enviemos un número y ella nos calcula su correspondiente factorial.

Ahora sí vamos a desarrollar el algoritmo completo basado en estas dos funciones de las cuales ya no tenemos que preocuparnos pues ya tenemos la absoluta certeza de que la *función Es_Par* permite determinar si un número es par y la *función Factorial* nos permite calcular el factorial de un número.

Algoritmo Determina_Facto_Par

```
Funcion Principal
Variables Locales
        Entero :        Num,            // Almacenará el número original leído
                        Aux1,           // Variables Auxiliares
                        Aux2
Inicio
        Escriba "Digite un número entero"       // Solicita un número entero
        Lea Num                                 // lo lee y lo almacena en Num
        Aux1 = Es_Par ( Num )                   // Aux1 es igual a lo que retorne la
                                                // función Es_Par enviándole como
                                                // parámetro el valor almacenado en
                                                // Num

        Si Aux1 = 0                             // Si ese valor retornado es cero
                Escriba "El número leído es impar"  // entonces el número no es par
        Sino                                    // Sino (el número si es par)
                Aux2 = Factorial ( Num )        // Aux2 es igual al valor que retorne la
                                                // función Factorial enviándole como
                                                // parámetro el contenido de la variable
                                                // Num

                Si Aux2 = -1                    // si ese valor retornado por la función
                                                // Factorial es –1 entonces quiere decir
                                                // que el número original era negativo

                        Escriba "No está definido el factorial para números negativos"

                Sino                            // Sino entonces se muestra en pantalla
                                                // que el factorial del valor almacenado
                                                // en Num es igual al valor almacenado
                                                // en Aux2 que fue lo que retornó la
                                                // función Factorial
                        Escriba "El factorial de ", Num, " es ", Aux2
                Fin_Si
        Fin_Si
Fin


Funcion Es_Par ( Entero n )     // Función Es_Par que recibe como parámetro un valor
                                // entero
Inicio
        Si  n / 2 * 2 = n       // Pregunta si el valor recibido es par
                Retorne ( 1 )   // entonces que retorne un 1
        Sino                    // Sino
                Retorne ( 0 )   // que retorne un 0
        Fin_Si                  // Fin de la decisión
Fin                             // Fin de la función


Función Factorial ( Entero n )  // Función Factorial que recibirá como parámetro un
                                // entero
Variables Locales               // Declaración de las variables locales
```

```
        Entero :        Facto,          // Almacenará el valor del factorial
                        Cont            // Permitirá generar los números desde 1 hasta el
                                        // parámetro recibido que es a quien se le debe calcular
                                        // el factorial
Inicio
        Si n < 0                        // Si el parámetro recibido es negativo
                Retorne ( -1 )          // Retorna un –1 que luego será interpretado desde la
                                        // función llamadora

        Facto = 1                       // Inicie la variable Facto en 1
        Cont = 1                        // Inicie la variable Cont en 1
        Mientras Con < = n              // Mientras Cont no haya llegado al número recibido
                                        // como parámetro
                Facto = Facto * Cont    // Multiplique sucesivamente todos los enteros
                Cont = Cont + 1         // comprendidos entre 1 y el número recibido como
                                        // parámetro
        Fin_Mientras                    // Fin del ciclo Mientras

        Retorne ( Facto )               // Retorne el valor almacenado en la variable Facto
Fin                                     // Fin de la función
```

Ahora usted si podrá ver lo tan útil que resulta ser para un programador utilizar el concepto de función pues como puede ver en este algoritmo ya no tenemos que realizar una prueba de escritorio detallada a todo el algoritmo ya que este se basa las funciones *Es_Par* y *Factorial* y como al hacerle su correspondiente prueba de escritorio a cada una de ellas funcionaron perfectamente podemos garantizar que el algoritmo completo también está bien. Le sugiero solo como una confirmación que le haga usted la prueba de escritorio a todo el algoritmo. Tenga muy en cuenta los valores que se retornan de cada función.

Adicionalmente en este algoritmo hemos obtenido una ganancia que yo estoy seguro que usted todavía no ha reflexionado. Tenemos dos funciones muy confiables que pueden llegar a ser utilizadas sin ningún temor en cualquier otro programa. Cabe anotar que en el programa anterior no era estrictamente necesario utilizar las variables *Aux1* y *Aux2* pero facilitan la claridad del algoritmo.

Ejemplo

Leer números hasta que digiten 0 y determinar a cuánto es igual el promedio de los factoriales de los números pares positivos.

Clarificación del Objetivo

Vamos a leer varios números, no sabemos cuántos, y en la medida en que los leamos iremos acumulando la suma de los factoriales de los números pares al tiempo que los iremos contando. Cuando nos digiten el valor 0 entonces realizaremos la división entre el valor acumulado de los

factoriales de los números pares y la cantidad de números pares que entraron y ese es el resultado que nos solicita este enunciado.

Puede usted notar que de alguna manera este ejercicio tiene una leve relación con el ejercicio anterior pues vamos a necesitar determinar si un número es par y también vamos a necesitar el cálculo del factorial del número. En estas condiciones se hará muy sencillo desarrollar este algoritmo pues nos vamos a apoyar en las mismas funciones (óigase bien, las mismas funciones) que utilizamos en el ejercicio anterior.

Algoritmo

Algoritmo Prom_Facto_Pares

```
Funcion Principal
Variables Locales                       // variables locales de
                                        // la función principal
        Entero :    Num,                // Almacenará cada uno de los números leídos
                    Acum_Facto,         // Almacenará la suma de todos los factoriales
                                        // de los números pares
                    Cont_Pares,         // Almacenará la cantidad de números pares a
                                        // los cuales se les calculó su factorial
                    Promedio            // Almacenará el promedio solicitado
Inicio
                                        // Solicita números e indica que finalicen con 0
        Escriba "Digite números y finalice con 0"

        Lea Num                         // lee el primer número

        Mientras Num < > 0              // Mientras lo números leídos sean diferentes
                                        // de 0
                                        // si el número leído es par y si su factorial es
                                        // mayor que 1 entonces acumule su factorial y
                                        // cuéntelo
            Si Es_Par ( Num ) = 1 Y Factorial ( Num ) >= 1
                Acum_Facto = Acum_Facto + Factorial ( Num )
                Cont_Pares = Cont_Pares + 1
            Fin_Si

            Lea Num                     // lea el siguiente número
        Fin_Mientras                    // fin del ciclo

        Si Cont_Pares < > 0                         // Si realmente hubo pares
            Promedio = Acum_Facto / Cont_Pares      // Calcule el promedio solicitado

            Si Es_Par ( Promedio ) = 1              // Si el promedio es par avise
                Escriba "El promedio es ", Promedio, " y es par "
            Sino                                    // si no lo es también
                Escriba "El promedio es ", Promedio, " y no es par"
            Fin_Si
        Sino                                        // Si no hubo pares avise
            Escriba "No hubo números pares"
        Fin_Si                                      // Fin de la decisión
```

```
Fin                                          // Fin de la función principal

// A continuación van a ir las mismas funciones (exactamente las mismas que utilizamos
// en el ejemplo pasado

Funcion Es_Par ( Entero n )                  // Función Es_Par que recibe como parámetro un valor
                                             // entero
Inicio
        Si  n / 2 * 2 = n                    // Pregunta si el valor recibido es par
                Retorne ( 1 )                // entonces que retorne un 1
        Sino                                 // Sino
                Retorne ( 0 )                // que retorne un 0
        Fin_Si                               // Fin de la decisión
Fin                                          // Fin de la función


Función Factorial ( Entero n )               // Función Factorial que recibirá como parámetro un
                                             // entero
Variables Locales                            // Declaración de las variables locales
        Entero :        Facto,               // Almacenará el valor del factorial
                        Cont                 // Permitirá generar los números desde 1 hasta el
                                             // parámetro recibido que es a quien se le debe calcular
                                             // el factorial
Inicio
        Si n < 0                             // Si el parámetro recibido es negativo
                Retorne ( -1 )               // Retorna un –1 que luego será interpretado desde la
                                             // función llamadora

        Facto = 1                            // Inicie la variable Facto en 1
        Cont = 1                             // Inicie la variable Cont en 1
        Mientras Con < = n                   // Mientras Cont no haya llegado al número recibido
                                             // como parámetro
                Facto = Facto * Cont         // Multiplique sucesivamente todos los enteros
                Cont = Cont + 1              // comprendidos entre 1 y el número recibido como
                                             // parámetro
        Fin_Mientras                         // Fin del ciclo Mientras

        Retorne ( Facto )                    // Retorne el valor almacenado en la variable Facto
Fin                                          // Fin de la función
```

Aquí si puede usted notar que nuestro algoritmo resultó mas sencillo de lo que se creía debido a que ya teníamos unas funciones sobre las cuales nos podemos apoyar sin tener que volver a construirlas ni probarlas. Me parece importante destacar que el solo llamado a una función (si esta retorna algún valor) es suficiente para entrar a evaluarla sin tener que almacenar en variables auxiliares o adicionales. De manera que líneas como

```
Si Es_Par ( Num ) = 1 Y Factorial ( Num ) >= 1
```

Son absolutamente correctas y todo lo que tenemos que hacer al momento de desarrollar la prueba de escritorio es tener en cuenta que el llamado a la función se reemplaza internamente por el valor

que ella retorne. Quedará pendiente para usted desarrollar la prueba de escritorio de este algoritmo.

Menús

Concepto General

Un menú sencillamente es un conjunto de opciones que se le presentan a un usuario para que el, de manera voluntaria y libre, escoja cuál ejecutar. Al igual que cuando vamos a un restaurante, el mesero nos presenta una lista de opciones (platos) para que nosotros escojamos. En nuestro ejemplo presentaremos un menú de opción leída dado que es la estructura que el seudocódigo nos permite. Los lenguajes de programación nos facilitan herramientas para construir menús de botones, de barras deslizantes y menúes gráficos pero por ahora nos concentraremos en la parte lógica del diseño de un menú. Para ello vamos a ejemplificar el concepto a través de un ejemplo muy sencillo.

Ejemplo

Brindar las siguientes opciones a través de un menú:

1. Leer un número entero
2. Determinar si dicho número es primo
3. Determinar si dicho número es par
4. Determinar si la suma de todos los enteros comprendidos entre 1 y dicho número, es un número par
5. Determinar si la suma de todos los enteros comprendidos entre 1 y dicho número, es un número primo
6. Determinar el factorial de dicho número entero

Es importante que se tenga en cuenta que siempre que nosotros brindemos un menú de opciones deberá existir una que permita terminar con el programa o sea salir del menú que se interpretaría sencillamente como salir del programa. Como usted puede ver nuestro enunciado ya no es tan sencillo como lo fue en otras oportunidades. Para su solución vamos a analizarlo opción por opción.

1ª. Opción. Leer un número entero

Para cumplir con este objetivo vamos a construir una función que nos permita leer un número entero y retornarlo a la función llamadora. Es evidente que esta función no necesitará ningún tipo de parámetros, de tal manera que una posible solución sería la siguiente:

```
Función Lectura_Num ( )                 // Nombre de la función
Variables Locales                       // Declaración de las variables locales
        Entero :        N               // Variable que almacenará el número a leer
Inicio
        Escriba "Digite un número entero"   // Solicita un número entero
        Lea N                           // y lo lee almacenándolo en la variable N

        Retorne ( N )                   // Retorna el valor leído
Fin                                     // Fin de la función
```

Por lo simplificada de la función podemos ver que efectivamente logrará el objetivo de leer un número entero y retornarlo.

2ª. Opción. Determinar si dicho número es primo

Para cumplir efectivamente con este objetivo se hace necesario que construyamos una función que nos permita recibir como parámetro un número entero y que retorne el valor 1 si dicho parámetro es un número primo y 0 si dicho parámetro no es un número primo. Recordemos entonces, primero que nada *Qué es un número primo..?* Un número primo es un número que solo es divisible exactamente entre 1 y sí mismo. Por ejemplo: el número 19 es primo porque solo es divisible exactamente entre 1 y 19. El número 18 no es primo porque lo dividen exactamente los números 1, 2, 3, 6, 9 y 18. Esta es la definición que por mucho tiempo hemos manejado y que ahora pretendemos programar a través de una función.

Vamos a realizarle un pequeño cambio a la definición sin apartarnos de la esencia de ella pero lo que necesitamos es facilitar el algoritmo y con él, facilitar su respectiva función. Siendo N un número cualquiera los posibles divisores exactos de N están en el rango de 1 a N (eso es evidente). Sin embargo sabiendo que todos, absolutamente todos los números, son divisibles exactamente entre 1 y N (siendo N cualquier cualquier número) entonces los divisores exactos de un número N cualquiera que nos interesan estarán en el rango de 2 a N-1.

Podríamos decir que si un número N no tiene divisores exactos en el rango de 2 a N-1 entonces con toda seguridad es un número primo y lo contrario también es cierto, es decir, si un número tiene al menos un divisor exacto en el rango de 2 a N-1 entonces el número no es primo.

Verifiquemos lo anterior con un ejemplo: El número 19 es primo porque no tiene ningún divisor exacto en el rango de 2 a 18, o sea, no hay ningún número que divida exactamente a 19 que sea mayor o igual que 2 y menor o igual que 18. Por eso podemos decir con toda certeza que el 19 es primo y es verdad. El número 24, en cambio, no es primo porque en el rango de 2 a 23 el número 24 tiene los siguientes divisores: 2, 3, 4, 6, 8 y 12. Por esta razón podemos asegurar que el número 24 no es primo y también es cierto.

Por lo tanto nuestra función se reduce a determinar si el número que se reciba como parámetro tiene divisores exactos en ese rango (o sea 2 a N-1 siendo N el número a verificar). De ser así entonces se retornará 0 pues si tiene divisores exactos en ese rango significa que el número recibido como parámetro no es un número primo y de no ser así se retornará 1 pues si no tiene divisores exactos en ese rango es porque los únicos números que lo dividen exactamente son el 1 y el mismo número N sea cual fuere.

Para optimizar un poco esta función vamos a involucrarle dos pequeños cambios con el propósito de lograr el objetivo de una manera mucho mas eficiente:

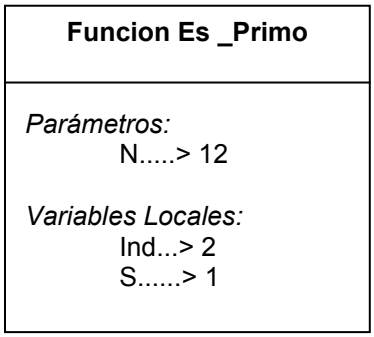
a. El rango de 2 a N-1 lo vamos a reducir de 2 a N/2 dado que ningún número tiene un divisor exacto de su mitad entera en adelante. El número 1000 por ejemplo no tiene ningún divisor exacto entre 501 y 999.
b. No vamos a esperar llegar hasta N/2 en el caso de que se encuentre al menos 1 divisor pues con ese simple hecho el número N a evaluar ya no sería primo y sería suficiente para retornar la respuesta solicitada.

Con estos dos pequeños cambios tendremos de una manera altamente eficiente la respuesta a nuestro pequeño objetivo. Para que el ciclo que inicialmente va a ir desde 1 hasta N/2 se interrumpa en cualquier momento, vamos a utilizar una variable que actuará a manera de interruptor y será la que nos va a permitir que nuestro objetivo se logre eficientemente. Sin mas preámbulos, una solución a esta función podrá ser la siguiente:

```
Funcion Es_Primo ( Entero N )
Variables Locales
        Entero :        Ind,                    // Servirá para que se generen los números
                                                // enteros de 2 a N/2
                        S                       // Variables que actuará como interruptor y que
                                                // servirá de retorno
Inicio
        Ind = 2                                 // Inicia la variable Ind en 2
        S = 1                                   // Inicia el interruptor en 1

        Mientras Ind < = N / 2 Y S = 1          // Mientras no se haya llegado a la mitad del
                                                // número y mientras no se haya encontrado
                                                // ningún divisor exacto
                Si N / Ind * Ind = N            // Si el valor almacenado en Ind es un divisor
                                                // exacto de N
                        S = 0                   // Cambie el valor del interruptor
                Fin_Si                          // Fin de la decisión

                Ind = Ind + 1                   // Pase al siguiente número dentro del rango
        Fin_Mientras                            // Fin del ciclo

        Retorne ( S )                           // Retorne el valor almacenado en S
Fin                                             // Fin de la función
```

Vamos a efectuarle una pequeña prueba de escritorio a esta función para verificar que efectivamente funcione bien. Nuestra función comienza almacenando el parámetro que se le envía en la variable N. Supongamos que se le envía un número 12. No se olvide que esta función

retornará 1 si el número (parámetro) es primo o 0 si no lo es. Con la última aclaración quiero recordar que esta función no va a mostrar nada en pantalla, solo va a retornar un 1 o un 0 dependiendo del valor que se reciba como parámetro.




```
Funcion Es_Primo ( Entero N )
Variables Locales
        Entero :        Ind,
                        S
Inicio
        Ind = 2
        S = 1
```

Seguidamente asignamos a la variable Ind el valor 2 y a la variable S el valor 1 y procedemos a entrar en el ciclo mientras que se plantea.

```
        Mientras Ind < = N / 2 Y S = 1
                Si N / Ind * Ind = N
                        S = 0
                Fin_Si

                Ind = Ind + 1
        Fin_Mientras
```

Conociendo los valores almacenados en las variables respectivas se evalúa la condición *Mientras Ind < = N / 2 **Y** S = 1* y vemos que el valor almacenado en Ind efectivamente es menor o igual que N/2 dado que Ind es igual a 2 y N/2 es igual a 6 y además se ve que el valor almacenado en S es igual a 1, por lo tanto toda la condición es Verdadera entonces efectuamos la pregunta

```
                Si N / Ind * Ind = N
```

Reemplazando por los valores respectivos la decisión se convierte en

```
                Si 12 / 2 * 2 = 12
```

Lo cual es efectivamente Verdadero porque se está preguntando en el fondo si el número 12 es divisible exactamente entre 2 y así es por lo tanto ejecutamos la orden

S = 0

Seguidamente luego del *Fin_Si* correspondiente ejecutamos la orden de incrementar en 1 el valor almacenado en la variable *Ind*.

Funcion Es _Primo
Parámetros: N.....> 12 *Variables Locales:* Ind...> ~~2~~ 3 S.....> ~~1~~ 0

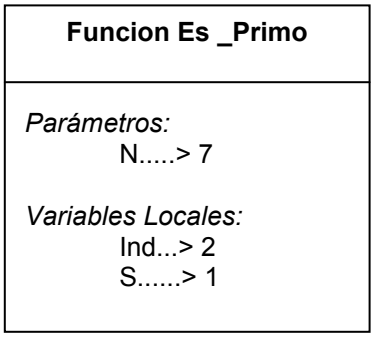
Como encontramos seguidamente el *Fin_Mientras* correspondiente entonces volvemos a evaluar la condición del ciclo

Mientras Ind < = N / 2 **Y** *S = 1*

Y vemos que el contenido de Ind sigue siendo menor o igual que el valor N/2 (o sea que 3 es menor o igual que 6) pero vemos que el valor de S ya no es 1 y por lo tanto como las dos condiciones están unidas por un operador **Y** entonces toda la condición es *Falsa*. Pasamos entonces a la instrucción que se encuentra después del *Fin_Mientras* correspondiente o sea

Retorne (S)
Fin

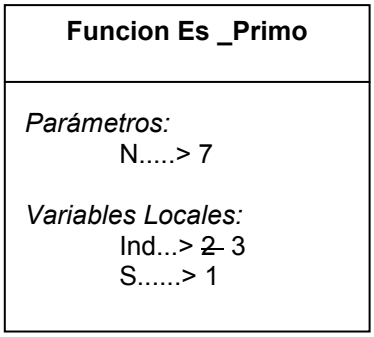
Con lo cual retornaríamos el valor almacenado en S (que esta vez sería el número 0) y que coincide con la definición pues habíamos dicho que en caso de que el parámetro no fuera un número primo entonces debía retornarse 0. De manera que para el caso de que el número no sea primo la función “funciona” (y valga esa redundancia). Veamos ahora una prueba de escritorio para el caso en que el parámetro recibido sea primo. La función deberá retornar un número 1 indicando que efectivamente el número es primo. Supongamos que el valor recibido como parámetro es el número 7 (que es primo) entonces nuestra pequeña prueba de escritorio comenzaría asignando dicho número a la variable *N* y declarando las variables *Ind* y *S* e iniciándolas con los valores 2 y 1 respectivamente.



Nuestra función continúa con el planteamiento de un ciclo que depende de que el contenido de *Ind* sea menor o igual que *N/2* y que el contenido de *S* sea 1. Como por esta vez la condición del ciclo es Verdadera entonces ejecutamos la decisión que sigue

```
Mientras Ind < = N / 2 Y S = 1
        Si N / Ind * Ind = N
                S = 0
        Fin_Si

        Ind = Ind + 1
Fin_Mientras
```

La pregunta *Si N / Ind * Ind = N* se convierte en *Si 7 / 2 * 2 = 7* o sea si 7 es divisible exactamente entre 2 lo cual es *Falso* por lo tanto pasamos a la instrucción que se encuentra después del *Fin_Si* correspondiente e incrementamos en 1 el contenido de la variable *Ind*.



Como lo que sigue es el Fin_Mientras correspondiente entonces volvemos a evaluar la condición del ciclo.

```
Mientras Ind < = N / 2 Y S = 1
```

Vemos pues que sigue siendo *Verdadera* pues el contenido de *Ind* es menor o igual que *N/2* dado que ambos valores valen 3 y el contenido de la variable *S* sigue siendo 1. Por lo tanto volvemos a entrar al ciclo y hacemos la decisión

*Si N / Ind * Ind = N*

O sea que preguntamos si el número 7 (que corresponde al contenido de *N*) es divisible exactamente entre 3 (que corresponde al contenido de *Ind*). Como la respuesta a esta decisión es *Falso* entonces pasamos a la instrucción que se encuentra después del *Fin_Si* correspondiente o sea que volvemos a incrementar en 1 el contenido de *Ind* quedando esta variable con el valor 4.

Funcion Es _Primo
Parámetros: N.....> 7 *Variables Locales:* Ind...> ~~2~~ ~~3~~ 4 S.....> 1

Como seguidamente encontramos el fin del ciclo volvemos a evaluar la condición el ciclo o sea

Mientras Ind < = N / 2 ***Y*** *S = 1*

Vemos entonces que el contenido de Ind (que es igual a 4) ya no menor ni igual que el resultado N/2 (pues este es igual a 3) por lo tanto esta parte de la decisión es Falsa aunque el contenido de S sigue siendo 1 como están unidas por un operador **Y** toda la decisión es Falsa y por lo tanto nos salimos del ciclo pasando a ejecutar la instrucción que se encuentra después del respectivo *Fin_Mientras*.

Retorne (S)
Fin

Con lo cual retornamos el valor almacenado en *S* que es igual a 1 con lo cual se confirma el propósito inicial pues habíamos dicho que se retornaría 1 si el valor recibido como parámetro es primo. Con esta prueba de escritorio podemos garantizar que la función *Es_Primo* sí nos permite determinar si el valor recibido como parámetro es un número primo o no. Solo existe una condición para su utilización exitosa y es que el valor recibido sea un número positivo.

3ª opción. Determinar si dicho número es par

Para determinar si un dato es par o no ya no tenemos que preocuparnos pues ya desarrollamos una función que s encarga de esto y la vamos a utilizar en este programa. Me refiero a la función

```
Funcion Es_Par ( Entero n )      // Función Es_Par que recibe como parámetro un valor
                                 // entero
Inicio
        Si  n / 2 * 2 = n        // Pregunta si el valor recibido es par
                Retorne ( 1 )    // entonces que retorne un 1
        Sino                     // Sino
                Retorne ( 0 )    // que retorne un 0
        Fin_Si                   // Fin de la decisión
Fin                              // Fin de la función
```

Como a esta función ya se le hizo la prueba de escritorio y tenemos la absoluta certeza de que funciona entonces no tenemos que hacer mayor cosa. Solo utilizarla bien dentro e nuestro programa.

4ª. Opción. Determinar si la suma de todos los enteros comprendidos entre 1 y dicho número, es un número par

Para cumplir este objetivo todo lo que tenemos que hacer es sumar todos los números enteros comprendidos entre 1 y el número leído y ese resultado, que deberá quedar en una variable almacenado, enviarlo como parámetro a la función *Es_Par* y ella se encargará de decirnos y si este número es par o no.

5ª. Opción. Determinar si la suma de todos los enteros comprendidos entre 1 y dicho número, es un número primo

Al igual que en el objetivo anterior debemos sumar todos los números enteros comprendidos entre 1 y el número leído y enviar ese resultado como parámetro a la función Es_Primo que acabamos de construir. Esa función se encargará de retornarnos la respuesta acerca de si el número es primo o no.

6ª. Opción. Determinar el factorial de dicho número entero

Para calcular el factorial ya tenemos una función probada que nos permite obtener ese resultado. Es la función *Factorial* y todo lo que tenemos que hacer es utilizarla bien.

```
Función Factorial ( Entero n )      // Función Factorial que recibirá como parámetro un
                                    // entero
Variables Locales                   // Declaración de las variables locales
        Entero :    Facto,          // Almacenará el valor del factorial
                    Cont            // Permitirá generar los números desde 1 hasta el
                                    // parámetro recibido que es a quien se le debe calcular
                                    // el factorial
Inicio
        Si n < 0                    // Si el parámetro recibido es negativo
                Retorne ( -1 )      // Retorna un –1 que luego será interpretado desde la
                                    // función llamadora

        Facto = 1                   // Inicie la variable Facto en 1
        Cont = 1                    // Inicie la variable Cont en 1
        Mientras Con < = n          // Mientras Cont no haya llegado al número recibido
                                    // como parámetro
                Facto = Facto * Cont // Multiplique sucesivamente todos los enteros
                Cont = Cont + 1     // comprendidos entre 1 y el número recibido como
                                    // parámetro
        Fin_Mientras                // Fin del ciclo Mientras

        Retorne ( Facto )           // Retorne el valor almacenado en la variable Facto
Fin                                 // Fin de la función
```

Como a esta función ya se le había hecho su correspondiente prueba de escritorio y ya tenemos absoluta certeza de sus resultados no tenemos que preocuparnos de ella.

Como puede ver “armar” un algoritmo a partir del concepto de funciones definitivamente se simplifica pues cada vez que usted desarrolla una función va a ser muy posible que en algoritmos futuros la pueda utilizar y estará ahorrando trabajo en el logro de los objetivos de ese nuevo algoritmo. Encontrar los errores como ya vimos es muy sencillo cuando se utiliza esa filosofía de trabajo y sobre todo comprender la lógica del algoritmo se hace todavía mas sencillo.

El enunciado inicial es:

Brindar las siguientes opciones a través de un menú:

1. *Leer un número entero*
2. *Determinar si dicho número es primo*
3. *Determinar si dicho número es par*
4. *Determinar si la suma de todos los enteros comprendidos entre 1 y dicho número, es un número par*
5. *Determinar si la suma de todos los enteros comprendidos entre 1 y dicho número, es un número primo*
6. *Determinar el factorial de dicho número entero*

El algoritmo completo solución para este enunciado podría ser el siguiente:

Algoritmo Menu_de_Opciones

Funcion Principal
Variables Locales
Entero : Num, // Almacenará el número leído
Opcion, // Almacenará la opción escogida por el usuario
Acum // Acumulará la suma de 1 hasta el número
// leído

Inicio
Opcion = 0 // Inicializamos la variable Opcion con cualquier
// valor para que la condición del ciclo
// inicialmente sea Verdadera

Mientras Opcion < > 7 // Mientras no presionen la opción de Salir

Escriba " 1. Leer un número entero " //Muestre las opciones
Escriba " 2. Determinar si dicho número es primo "
Escriba " 3. Determinar si dicho número es par "
Escriba " 4. Determinar si la suma de 1 a N es par "
Escriba " 5. Determinar si la suma de 1 a N es primo "
Escriba " 6. Calcular el factorial del número leído "
Escriba " 7. Salir "

Lea Opcion // Recibe la opción que quiere
// el usuario

Evalúe (Opcion) // Evalúe lo que el usuario digitó
Si vale 1 : // Si escogió la opción 1
Num = Lectura_Num ( ) // Entonces llame a la función
// que lee un entero y lo que
// retorne almacénelo en la
// variable Num

Si vale 2 : // Si el usuario escogió la
// opción 2
Si Es_Primo (Num) = 1 // Si lo que retorne la función
// Es_Primo enviándole como
// parámetro el valor
// almacenado en la variable
// Num es //igual a 1
// quiere decir que el número
// leído es primo
Escriba " El número es primo "
Sino
Escriba " El número no es primo "
Fin_Si
Si vale 3: // Si el usuario escogió la
// opción 3
Si Es_Par (Num) = 1 // Si lo que retorne la función
// Es_Par enviándole como
// parámetro el valor
// almacenado en la variable
// Num es igual a 1 quiere decir
// que el número leído es Par

```
                Escriba " El número es par "
        Sino
                Escriba " El número no es par "
        Fin_Si
Si vale 4 :                             // Si el usuario escogió la
                                        // opción 4
        Acum = 0                        // Inicialice esta variable en 0

        Para Ind = 1 hasta Num          // Acumule en ella el valor
                                        // resultante de sumar todos los
                                        // enteros comprendidos entre 1
                                        // y el valor leído
                Acum = Acum + Ind
        Fin_Para

        Si Es_Par ( Acum ) = 1          // Si el valor retornado por la
                                        // función Es_Par al enviársele
                                        // como parámetro el contenido
                                        // de Acum es 1 entonces eso
                                        // quiere decir que el valor de
                                        // esa suma es par
                Escriba "La suma de 1 a ", Num, " es par"
        Sino
                Escriba "La suma de 1 a ", Num, " no es par"
        Fin_Si
Si vale 5:                              // Si el usuario escoge la opción
                                        // 5
        Acum = 0                        // Inicialice esta variable con 0

        Para Ind = 1 hasta Num          // Acumule en ella el valor
                                        // resultante de sumar todos los
                                        // enteros comprendidos entre 1
                                        // y el valor leído
                Acum = Acum + Ind
        Fin_Para

        Si Es_Primo ( Acum ) = 1         // Si el valor retornado por la
                                        // función Es_primo al
                                        // enviársele como parámetro el
                                        // contenido de Acum es 1
                                        // entonces eso quiere decir
                                        // que el valor de esa suma es
                                        // un número primo
                Escriba "La suma de 1 a ", Num, " es un número primo"
        Sino
                Escriba "La suma de 1 a ", Num, " no es un primo"
        Fin_Si
Si vale 6 :                             // Si el usuario escogió la
                                        // opción 6
        Si Factorial ( Num ) = -1       // Evalué si lo que retorna la
                                        // función Factorial es igual a –1
                                        // entonces quiere decir que el
                                        // número original era un
                                        // número negativo
                Escriba " No están definidos factoriales de números
                        negativos "
```

```
                    Sino                        // De no ser así entonces
                                                // muestre en pantalla el valor
                                                // del factorial del número leído
                        Escriba "El factorial de ", Num, " es ", Factorial ( Num )
                    Fin_Si

                Si vale 7 :                     // Si el usuario escogió la
                                                // opción 7
                    Escriba "Adiós"             // Despedirse

                Sino :                          // Si el usuario no escogió ni la
                                                // opción 1, ni la 2, ni la 3, ni la
                                                // 4, ni la 5, ni la 6, ni la 7 quiere
                                                // decir que se equivocó y
                                                // entonces nos toca avisarle
                    Escriba "Opción Errada "
            Fin_Evalúe                          // Fin de la estructura Evalúe
        Fin_Mientras                            // Fin del ciclo
Fin                                             // Fin de la Función Principal
                                                // (que significa en otras
                                                // palabras Fin del algoritmo)
```

```
Función Lectura_Num ( )                 // Nombre de la función
Variables Locales                       // Declaración de las variables locales
        Entero :        N               // Variable que almacenará el número a leer
Inicio
        Escriba "Digite un número entero"   // Solicita un número entero
        Lea N                           // y lo lee almacenándolo en la variable N

        Retorne ( N )                   // Retorna el valor leído
Fin                                     // Fin de la función
```

```
Funcion Es_Par ( Entero n )             // Función Es_Par que recibe como parámetro un valor
                                        // entero
Inicio
        Si  n / 2 * 2 = n               // Pregunta si el valor recibido es par
            Retorne ( 1 )               // entonces que retorne un 1
        Sino                            // Sino
            Retorne ( 0 )               // que retorne un 0
        Fin_Si                          // Fin de la decisión
Fin                                     // Fin de la función
```

```
Funcion Es_Primo ( Entero N )
Variables Locales
        Entero :        Ind,            // Servirá para que se generen los números
                                        // enteros de 2 a N/2
                        S               // Variables que actuará como interruptor y que
                                        // servirá de retorno
Inicio
        Ind = 2                         // Inicia la variable Ind en 2
```

```
        S = 1                               // Inicia el interruptor en 1

        Mientras Ind < = N / 2 Y S = 1      // Mientras no se haya llegado a la mitad del
                                            // número y mientras no se haya encontrado
                                            // ningún divisor exacto
                Si N / Ind * Ind = N        // Si el valor almacenado en Ind es un divisor
                                            // exacto de N
                        S = 0               // Cambie el valor del interruptor
                Fin_Si                      // Fin de la decisión

                Ind = Ind + 1               // Pase al siguiente número dentro del rango
        Fin_Mientras                        // Fin del ciclo

        Retorne ( S )                       // Retorne el valor almacenado en S
Fin                                         // Fin de la función
```

```
Función Factorial ( Entero n )      // Función Factorial que recibirá como parámetro un
                                    // entero
Variables Locales                   // Declaración de las variables locales
        Entero :    Facto,          // Almacenará el valor del factorial
                    Cont            // Permitirá generar los números desde 1 hasta el
                                    // parámetro recibido que es a quien se le debe calcular
                                    // el factorial
Inicio
        Si n < 0                    // Si el parámetro recibido es negativo
                Retorne ( -1 )      // Retorna un –1 que luego será interpretado desde la
                                    // función llamadora

        Facto = 1                   // Inicie la variable Facto en 1
        Cont = 1                    // Inicie la variable Cont en 1
        Mientras Con < = n          // Mientras Cont no haya llegado al número recibido
                                    // como parámetro
                Facto = Facto * Cont // Multiplique sucesivamente todos los enteros
                Cont = Cont + 1     // comprendidos entre 1 y el número recibido como
                                    // parámetro
        Fin_Mientras                // Fin del ciclo Mientras

        Retorne ( Facto )           // Retorne el valor almacenado en la variable Facto
Fin                                 // Fin de la función
```

Usted deberá desarrollarle la prueba de escritorio a la Función Principal dado que a las demás funciones ya se le hizo su correspondiente prueba. Cuando necesite el resultado de alguna de las demás funciones todo lo que tiene que hacer es colocar el resultado que usted sabe que debe retornar la función pues ésta ya ha sido probada.

Es importante que tenga en cuenta que cuando se brinda un menú no existe un orden en las opciones que escoja el usuario. El podrá escoger cualquier opción y el algoritmo deberá funcionar. Igualmente el usuario podrá escoger una y otra opción tantas veces como quiera y solo se saldrá

del algoritmo cuando escoja la opción de *Salir* (para este caso la opción 7.). De la misma manera tenga en cuenta que todo menú deberá tener siempre una opción para Salir naturalmente.

Usted deberá realizarle una prueba de escritorio a la Función Principal y ajustar todo el algoritmo para cuando el número original sea negativo pues este caso no está considerado en él intencionalmente. Procure hacer todos los cambios en la función principal y deje las demás funciones intactas.

Ejercicios

Algunas posibles soluciones a los enunciados aquí planteados puede encontrarlas en el Libro *Algoritmos* del mismo autor.

Nota Aclaratoria: En los siguientes enunciados se deberá construir la función solicitada y una Función Principal que haga uso de la función solicitada.

1. Construir una función que reciba como parámetro un entero y retorne su último dígito.

2. Construir una función que reciba como parámetro un entero y retorne sus dos últimos dígitos.

3. Construir una función que reciba como parámetro un entero y retorne la cantidad de dígitos.

4. Construir una función que reciba como parámetro un entero y retorne la cantidad de dígitos pares.

5. Construir una función que reciba como parámetro un entero y retorne la cantidad de dígitos primos.

6. Construir una función que reciba como parámetro un entero y retorne el carácter al cual pertenece ese entero como código ASCII.

7. Construir una función que reciba como parámetro un carácter y retorne el código ASCII asociado a él.

8. Construir una función que reciba como parámetro un entero y retorne 1 si dicho entero está entre los 30 primeros elementos de la serie de Fibonacci. Deberá retornar 0 si no es así.

9. Construir una función que reciba un entero y le calcule su factorial sabiendo que el factorial de un número es el resultado de multiplicar sucesivamente todos los enteros comprendidos entre 1 y el número dado. El factorial de 0 es 1. No están definidos los factoriales de números negativos.

10. Construir una función que reciba como parámetro un entero y retorne el primer dígito de este entero.

11. Construir una función que reciba como parámetro un entero y un dígito y retorne 1 si dicho entero es múltiplo de dicho dígito y 0 si no es así.

12. Construir una función que reciba como parámetro un entero y un dígito y retorne 1 si dicho dígito está en dicho entero y 0 si no es así.

13. Construir una función que reciba como parámetro un entero y un dígito y retorne la cantidad de veces que se encuentra dicho dígito en dicho entero.

14. Construir una función que reciba como parámetros dos números enteros y retorne el valor del mayor.

15. Construir una función que reciba como parámetros dos números enteros y retorne 1 si el primer número es múltiplo del segundo y 0 si no.

16. Construir una función que reciba como parámetro un entero y retorne 1 si corresponde al código ASCII de una letra minúscula (Los códigos ASCII de las letras minúsculas van desde 97 que el código de la letra a hasta 122 que es el código de la letra z). Deberá retornar 0 si no es así.

17. Construir una función que reciba como parámetro un entero y retorne 1 si corresponde al código ASCII de un dígito (Los códigos ASCII de las letras minúsculas van desde 48 que es el código del dígito 0 hasta 57 que es el código del dígito 9). Deberá retornar 0 si no es así.

18. Construir una función que reciba como parámetro un valor entero y retornar 1 si dicho valor es el factorial de alguno de los dígitos del número. Deberá retornar 0 si no es así.

19. Construir una función que reciba como parámetro un entero y retorne 1 si dicho valor es un número de mínimo 3 dígitos. Deberá retornar 0 si no es así.

20. Construir una función que reciba como parámetro un entero y retorne 1 si en dicho valor todos los dígitos son iguales. Deberá retornar 0 si no es así.

21. Construir una función que reciba como parámetro un entero y retorne 1 si en dicho valor el primer dígito es igual al último. Deberá retornar 0 si no es así.

22. Construir una función que reciba como parámetro un entero y retorne 1 si dicho valor es múltiplo de 5. Deberá retornar 0 si no es así.

23. Construir una función que reciba como parámetro dos enteros y retorne 1 si la diferencia entre los dos valores es un número primo. Deberá retornar 0 si no es así.

24. Construir una función que reciba como parámetro dos enteros de dos dígitos cada uno y retorne 1 si son inversos. Ejemplo: 83 es inverso de 38. Deberá retornar 0 si no es así.

25. Construir una función que reciba como parámetro un entero y un dígito menor o igual a 5 y retorne el dígito del número que se encuentre en la posición especificada por el dígito que llegó como parámetro.

26. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne el mayor de los datos del vector.

27. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la posición en la cual se encuentra el mayor de los datos del vector.

28. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la cantidad de números primos almacenados en el vector.

29. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la cantidad de números que pertenecen a los 30 primeros elementos de la serie de Fibonacci.

30. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la posición del mayor número primo almacenado en el vector.

31. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne el promedio entero del vector.

32. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne el promedio real del vector.

33. Construir una función que reciba como parámetros un vector de 10 posiciones enteras y un valor entero y retorne 1 si dicho valor entero se encuentra en el vector. Deberá retornar 0 si no es así.

34. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la posición del número entero que tenga mayor cantidad de dígitos.

35. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la posición en la que se encuentre el mayor número primo que termine en 3 almacenado en el vector.

36. Construir una función que reciba como parámetro un entero y retorne ese elemento de la serie de Fibonacci.

37. Construir una función que reciba como parámetros dos enteros, el primero actuará como base y el segundo como exponente y retorne el resultado de elevar dicha base a dicho exponente.

38. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la cantidad de números terminados en 3 que contiene el vector.

39. Construir una función que reciba como parámetros un vector de 10 posiciones enteras y un dígito y que retorne la cantidad de veces que dicho dígito se encuentra en el vector. No se olvide que un mismo dígito puede estar varias veces en un solo número.

40. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y un dígito y que retorne la cantidad de números del vector que terminan en dicho dígito.

41. Construir una función que reciba como parámetro un vector de 10 posiciones enteras y un dígito y que retorne la cantidad de números del vector en donde dicho dígito está de penúltimo.

42. Construir una función que reciba como parámetro una matriz de 3x4 entera y retorne la cantidad de veces que se repite el mayor dato de la matriz.

43. Construir una función que reciba como parámetro una matriz 3x4 entera y retorne la cantidad de números primos almacenados en la matriz.

44. Construir una función que reciba como parámetro una matriz 3x4 entera y retorne la cantidad de veces que se repite el mayor número primo de la matriz.

45. Construir una función que reciba como parámetros una matriz 4x4 entera y un valor entero y retorne la cantidad de veces que se repite dicho valor en la matriz.

46. Construir una función que reciba como parámetro una matriz 4x4 entera y retorne el número de la fila en donde se encuentre por primera vez el número mayor de la matriz.

47. Construir una función que reciba como parámetro una matriz 4x4 entera y retorne el número de la columna en donde se encuentre por primera vez el número mayor de la matriz.

48. Construir una función que reciba como parámetro una matriz 4x4 entera y retorne la posición exacta en donde se encuentre almacenado el mayor número primo.

49. Construir una función que reciba una matriz 5x5 y retorne el valor de su moda. La moda de un conjunto de datos es el dato que mas se repite.

50. Construir una función que reciba una matriz 5x5 y retorne la cantidad de veces que se repite su moda.