

Lógica de Programación. Nivel II

abril, 2018



Objetivos del nivel

- Entender el uso de las instrucciones de control en los algoritmos.
- Aprender a crear instrucciones condicionales
- Entender la modularidad y su importancia
- Aprender a crear módulo reusables
- Comprender el paso de parámetros

Prerrequisitos del nivel

- Lógica de Programación Nivel I

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestra sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños.
Copyright 2018. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Estructuras Repetitivas. Parte 1

- 1.1.- Definiciones.
- 1.2.- Control de Ciclos.
- 1.3.- Tipos de Ciclos.
- 1.4.- Ciclos Infinitos.

Capítulo 2. Estructuras Repetitivas. Parte 2

- 2.1.- Estructura Repetitiva "para".
- 2.2.- Número de Iteraciones Variable.
- 2.3.- Incremento Variable.

Capítulo 3. Expresiones Lógicas

- 3.1.- Concepto.
- 3.2.- Operadores Relacionales.
- 3.3.- Construcción de Expresiones Lógicas.

Capítulo 4. Estructuras Selectivas. Parte 1

- 4.1.- Concepto.
- 4.2.- Estructura Selectiva Simple.

Capítulo 5. Estructuras Selectivas. Parte 2

- 5.1.- Estructura Selectiva Doble.
- 5.2.- Aplicaciones de la Estructura Selectiva.

Capítulo 6. Estructuras Selectivas. Parte 3

- 6.1.- Anidación de Condicionales.
- 6.2.- Comparación de Variables.
- 6.3.- Evaluación de Intervalos.

Capítulo 7. Estructuras Selectivas. Parte 5

- 7.1.- Operadores Lógicos.
- 7.2.- Condiciones Compuestas.

Capítulo 8. Tabla de la Verdad

- 8.1.- Concepto.
- 8.2.- Tabla de la Conjunción.
- 8.3.- Tabla de la Disyunción.

Capítulo 9. Modularidad. Parte 1

- 9.1.- Concepto.
- 9.2.- Definición de Módulos.
- 9.3.- Llamado de Módulos.

Capítulo 10. Variables: Globales y Locales

- 10.1.- Variables Globales.
- 10.2.- Variables Locales.
- 10.3.- Ámbito de Las Variables.

Capítulo 11. Comunicación Entre Módulos. Parte 1

- 11.1.- Concepto.
- 11.2.- Parámetros Formales.
- 11.3.- Parámetros Actuales.
- 11.4.- Relación Entre Los Parámetros.

Capítulo 12. Comunicación Entre Módulos. Parte 2

- 12.1.- Parámetros Por Valor.
- 12.2.- Parámetros Por Referencia.
- 12.3.- Módulos Con Ambos Parámetros.

Capítulo 13. Modularidad. Parte 2

- 13.1.- El Cuerpo Principal.
- 13.2.- Estructura Jerárquica.
- 13.3.- Reusabilidad Con Parámetros.

Capítulo 14. Modularidad. Parte 3

- 14.1.- Creación de Sub-módulos.
- 14.2.- Ámbito de Uso de Los Parámetros.

Capítulo 15. Módulos Como Funciones

- 15.1.- Concepto.
- 15.2.- Declaración.
- 15.3.- Llamado de Funciones.



Capítulo 1. ESTRUCTURAS REPETITIVAS. PARTE 1

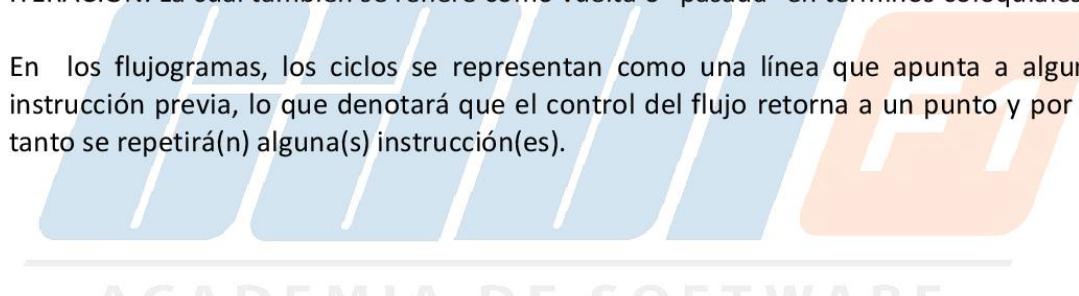
1.1.- Definiciones

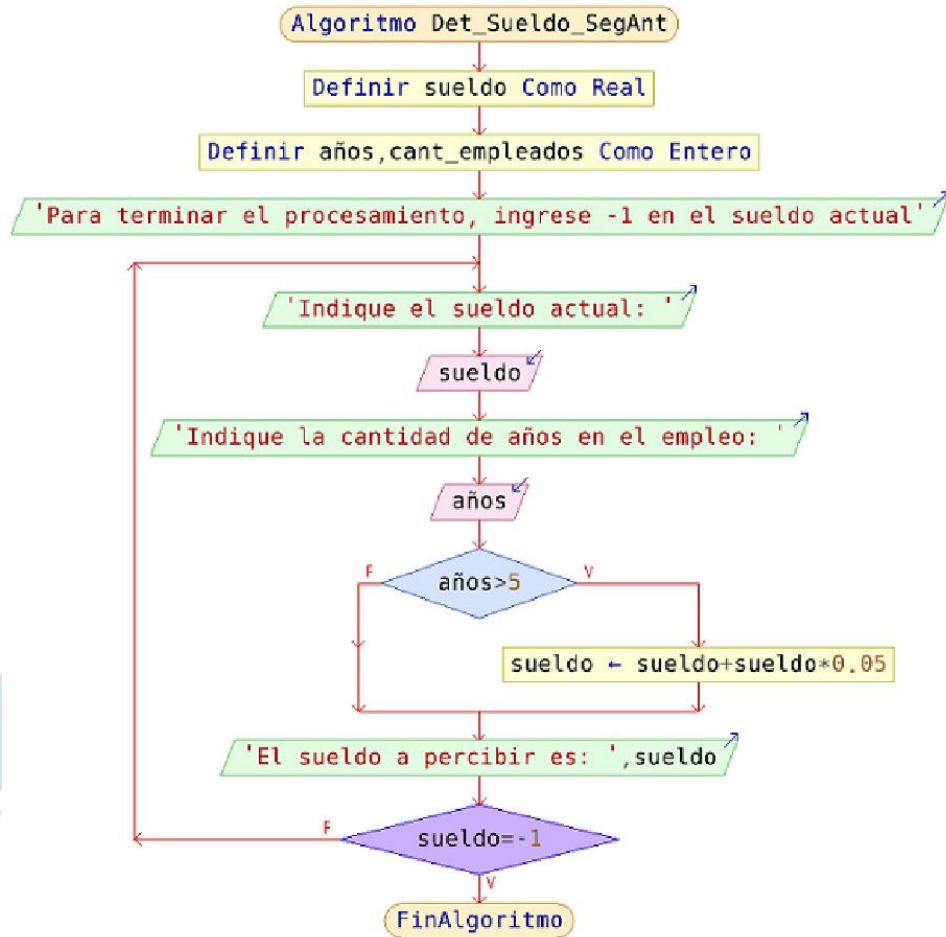
Todavía no se ha visto como incluir cambios en el control de ejecución usando estructuras selectivas, de manera que en esos algoritmos para algunas instrucciones se ejecuten dependiendo de condiciones, pero también hay problemas cuya resolución amerita repetir pasos específicos, por ejemplo: promediar varias notas leídas.

Una INSTRUCCIÓN ITERATIVA es aquella que se coloca en el código para lograr que ciertas instrucciones se ejecuten un número finito de veces.

En las estructuras repetitivas, de ciclo o bucle, cada repetición del bucle se llama ITERACIÓN. La cual también se refiere como vuelta o "pasada" en términos coloquiales.

En los flujogramas, los ciclos se representan como una línea que apunta a alguna instrucción previa, lo que denotará que el control del flujo retorna a un punto y por lo tanto se repetirá(n) alguna(s) instrucción(es).





Un ejemplo cotidiano donde se plantean ciclos, es la corrección que hace un profesor de los exámenes de una sección, ya que para ello tendría que:

Ver el nombre del alumno, ver la cédula, leer la respuesta de la primera pregunta y compararla con la respuesta correcta, decidir si está buena o no, asignar la nota a la pregunta e ir acumulando la nota. El paso de corregir la pregunta se va a repetir tantas veces, como preguntas tenga el examen.

Al final el profesor asigna la nota final al examen y lo anota en la lista de publicación de notas. Adicionalmente este algoritmo lo va a repetir con cada examen presentado por sus alumnos. ¿Dónde están los ciclos?

1.2.- Control de Ciclos

Todo bucle implica la evaluación de una condición, que es la que va a determinar si debe ejecutarse el bucle y hasta cuando. Por esto, cuando se define un ciclo se deben determinar 2 cosas: qué instrucciones van a repetirse en el ciclo y qué condición va a mantener el ciclo (dicho de otra forma, como se controlará el ciclo). Esto último es muy importante y está muy relacionado con el empleo de condiciones.

Los ciclos pueden romperse (o controlarse) de 2 formas:

- cuando se haya cumplido el número de iteraciones que se esperaba dar.
- cuando ocurra algo específico, por ejemplo, que una variable tome un valor.

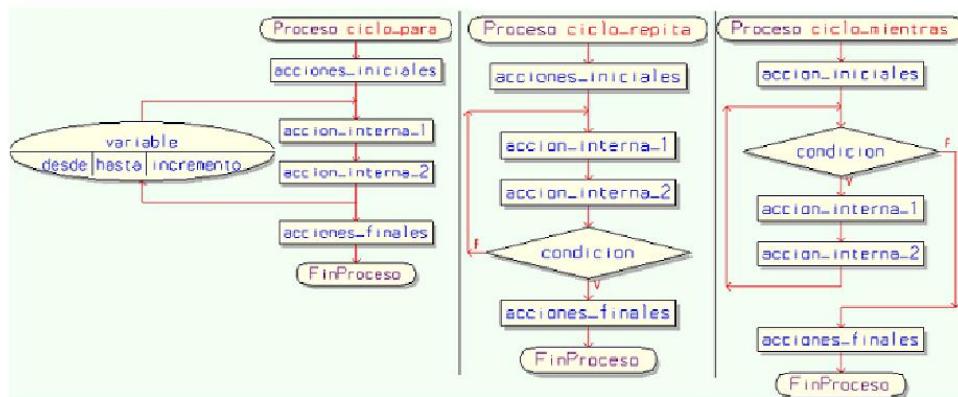
Por lo anterior, cada problema puede ser muy particular. La forma de control del ciclo o de los ciclos (en caso de que haya varios) debe determinarse al analizar el problema.

1.3.- Tipos de Ciclos

Los lenguajes de programación tradicionalmente implementan 3 tipos de ciclos, los cuales se usan dependiendo de la situación y del algoritmo a resolver. Los tipos de ciclos son:

- PARA o DESDE (for): se utiliza generalmente cuando la cantidad de iteraciones es predecible (se conoce antes de iniciar el ciclo).
- REPETIR (repeat, do..while): se utiliza cuando la cantidad de iteraciones es impredecible.
- MIENTRAS (while): también se utiliza cuando la cantidad de iteraciones es impredecible, pero a diferencia del anterior, se utiliza en situaciones más particulares.

Cada tipo de ciclo se representa de manera diferente dentro de un flujograma:



1.4.- Ciclos Infinitos

El establecimiento de una condición incorrecta en un ciclo podría hacer que éste nunca se rompiera, generando así lo que se llama en programación "un ciclo infinito". Esto nunca debería ocurrir en situaciones tradicionales, puesto que debemos recordar que los algoritmos deben ser finitos.

Aun así, existen situaciones en las cuales los algoritmos deben ejecutar un ciclo "infinito" controlado, pero ese tipo de situaciones no serán abarcadas en este curso. Los ciclos "infinitos" se romperán por causas externas, por ejemplo, los sistemas que funcionan recibiendo "eventos" o mensajes del usuario de forma asíncrona (sistemas de chat, de ventanas, etc.) necesitan un ciclo infinito.

Capítulo 2. ESTRUCTURAS REPETITIVAS. PARTE 2

2.1.- Estructura Repetitiva "para"

La estructura repetitiva DESDE o PARA (llamada "for" en la mayoría de los lenguajes de programación) se usa cuando se conoce el número exacto de veces que se debe ejecutar el ciclo. El ciclo lleva asociado una variable que se denomina variable índice, a la que se le asigna un valor inicial, su valor final y opcionalmente se le asocia la cantidad en la que variará.

La variable índice se incrementa o decrementa de manera automática, en cada iteración del bucle, en un valor constante. El programador no se tiene que ocupar de actualizar el valor de esta variable en cada iteración del bucle debido a que es una operación que está implícita.

Así pues, dado que en cada iteración del bucle, la variable índice se actualiza automáticamente, cuando se alcanza el valor que se ha establecido como final, se termina la ejecución del mismo.

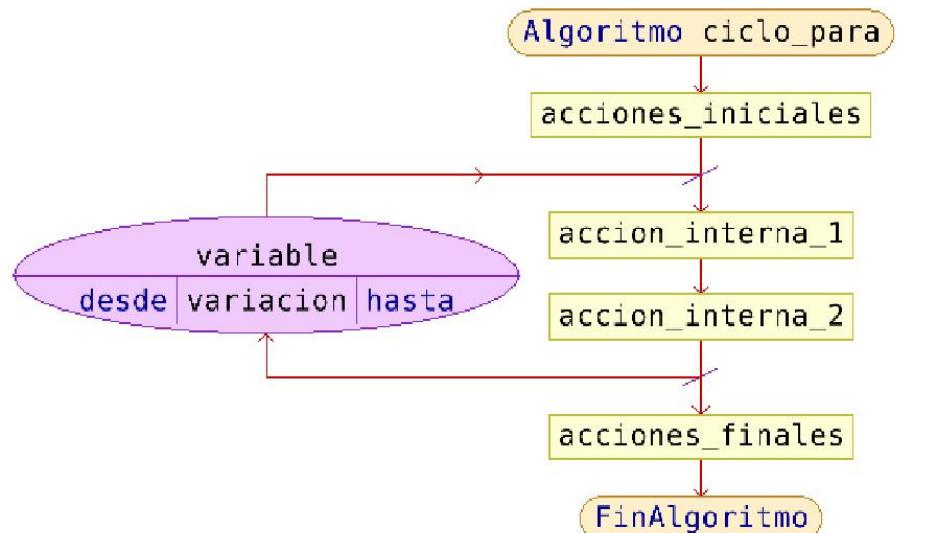
La estructura general del ciclo "para" es la siguiente:

```
acciones_iniciales
para variable=valorinicial hasta valorfinal

accion_interna_1
accion_interna_2

fin para
acciones_finales
```

El flujograma general del ciclo “para” es el siguiente:



En PseInt la estructura del ciclo "para" es la siguiente:

instrucciones_iniciales
para variable=vi hasta vf con paso n

accion_interna_1
accion_interna_2

fin para
instrucciones_finales

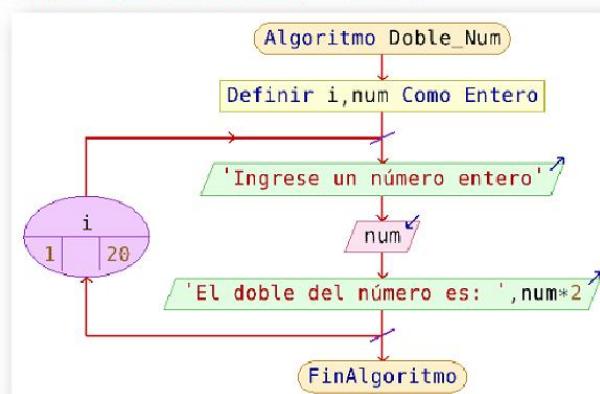
Donde vi es el valor inicial de la variable y vf es el valor final. La sección [con paso] es opcional, allí se establece en cuantas unidades se va ir modificando la variable que controla el ciclo, si se omite, se asume que la variable se va incrementando de 1 en 1 en cada iteración.

Por ejemplo, si quiere leer 20 números enteros y mostrar su doble. En este caso el valor inicial es 1 y el valor final es 20, y la variable que controla el ciclo aumentara de 1 en 1 (comportamiento por omisión).

```

1 Algoritmo Doble_Num
2     definir i,num Como Entero
3     Para i=1 hasta 20 Hacer
4         Mostrar "Ingrese un número entero " Sin Saltar
5         Leer num
6         Mostrar "El doble del número es: " num*2
7     FinPara
8 FinAlgoritmo

```



El valor inicial no necesariamente tiene que ser 1 o 0, puede ser cualquier valor, que generalmente es menor que el valor final. El siguiente es un ejemplo:

```

1 Algoritmo Doble_Num
2     definir i,num Como Entero
3
4     Para i=50 hasta 75 Hacer
5         Mostrar "Ingrese un número entero " Sin Saltar
6         Leer num
7         Mostrar "El doble del número es: " num*2
8     FinPara
9
10 FinAlgoritmo

```

2.2.- Número de Iteraciones Variable

Normalmente este valor final es conocido de antemano en el enunciado, pero también puede ser un dato de entrada, por ejemplo:

```

1 Algoritmo Doble_Num
2   Definir i,num,n Como Entero
3
4   Mostrar "Ingrese cuántos números va a procesar" Sin Saltar
5   Leer N
6
7   Para i=1 hasta N Hacer
8     Mostrar "Ingrese un número entero " Sin Saltar
9     Leer num
10    Mostrar "El doble del número es: " num*2
11  FinPara
12
13 FinAlgoritmo

```

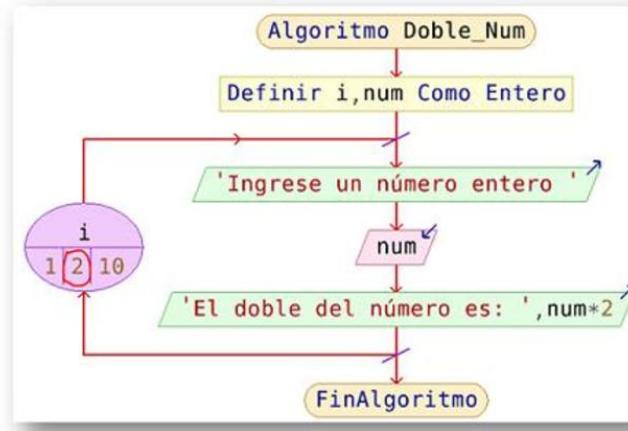
2.3.- Incremento Variable

El incremento de la variable normalmente se hace de 1 en 1, pero hay ocasiones donde se necesita que varíe en otros valores (2 en 2, 5 en 5, etc). A continuación se muestra un ejemplo de cómo cambiar el incremento de la variable que controla el ciclo:

```

1 Algoritmo Doble_Num
2   Definir i,num Como Entero
3
4   // Este ciclo se repetirá 5 veces porque
5   // la variable i, se incrementará de 2 en 2
6   Para i=1 hasta 10 con paso 2 Hacer
7     Mostrar "Ingrese un número entero " Sin Saltar
8     Leer num
9     Mostrar "El doble del número es: " num*2
10  FinPara
11
12 FinAlgoritmo

```



Con esta variación es posible realizar ciclos que se ejecuten en decremento (también llamados down-to), donde el valor de la variable que controla el ciclo va desde un valor inicial mayor que el valor final. El decremento puede realizarse 1 en 1 o con otros valores:

```
1 Algoritmo CTA_Regresiva
2   Definir i Como Entero
3
4   Para i=10 hasta 1 con paso -1 Hacer
5     Mostrar i
6
7   FinPara
8
9
10 FinAlgoritmo
11
12
```

```
PS4Int - Ejecutando proceso CTA_REGRESIVA
*** Ejecución Iniciada. ***
10
9
8
7
6
5
4
3
2
1
*** Ejecución Finalizada. ***
```



Capítulo 3. EXPRESIONES LÓGICAS

3.1.- Concepto

Una expresión lógica es aquella que solo puede devolver dos valores: verdadero o falso (nótese que dice "o", es decir, no puede ser verdadero y falso a la vez). Los algoritmos tienen instrucciones en las cuales se evalúa si algo es verdad o no, por ejemplo:

- ¿ Tengo dinero ?
- ¿ Eres mayor de edad ?
- ¿ Esta haciendo frio ?
- ¿ Es de noche ?
- ¿ Eres más alto que yo ?

En todas estas expresiones sólo hay 2 posibles resultados: verdadero o falso. La respuesta dependerá de cada caso y la respuesta del algoritmo deberá ser diferente según el resultado. En los problemas que se van a intentar resolver con un programa de computadora, habrá diversidad de circunstancias en las cuales los algoritmos deben hacer una cosa u otra. Es necesario detectar estas expresiones lógicas que estarán involucradas en cada problema.

3.2.- Operadores Relacionales

Las expresiones lógicas de algoritmos de computadoras deben expresarse en términos de variables comparándose con otras variables o valores. Para esto, se necesitan los operadores relacionales. Los operadores relacionales son:

- < Menor que
- > Mayor que
- = igualdad (en C: ==)
- <> Diferente (en C: !=)
- <= Menor o igual que
- >= Mayor o igual que

Los valores que se comparan deben ser del mismo tipo, es decir, no se puede comparar números con alfanuméricos, caracteres con booleanos, etc.

En las expresiones lógicas, el operador relacional se coloca entre los 2 operandos (los elementos que se desean comparar entre ellos). Estos operandos pueden ser constantes o variables o combinación de ambos. Por ejemplo:

$$5 > 7$$

El resultado de esta expresión siempre es falso, debido a que el número 5 no es mayor que el número 7. Rara vez se comparan 2 literales, debido a que el resultado siempre será el mismo. Otra forma de escribir esta expresión es la siguiente:

$$7 < 5$$

De igual forma, esta expresión es falsa. Es una expresión equivalente a la anterior.

Para evaluar una expresión lógica:

- Se resuelve el primer operando y se sustituye por su valor.
- Se resuelve el segundo operando y se sustituye por su valor.
- Se aplica el operador relacional y se devuelve el valor booleano correspondiente.

Por ejemplo:

$$\begin{aligned} ((5 * 4) + 1 - (5 ^ 2)) &< (2 - 1) \\ -4 &< 1 \end{aligned}$$

La expresión retorna verdadero.

Normalmente se utilizan variables en las expresiones lógicas. El resultado de la expresión dependerá del valor de las variables. Por ejemplo, si se tienen las variables "x" y "y", se le asigna el valor 2 a "x" y a la variable "y" se le asigna el valor 4. Cada respuesta debe ser VERDADERO o FALSO, según corresponda:

1. $x > 10$: _____
2. $y < 0$: _____
3. $3 \geq x$: _____

4. $x == y$: _____
5. $x <> y$: _____
6. $y > x$: _____
7. $x >= y$: _____
8. $3*x > y$: _____
9. $x+y < 5$: _____
10. $y-2 == x$: _____

3.3.- Construcción de Expresiones Lógicas

Las expresiones lógicas exemplificadas anteriormente de forma natural, deben estructurarse de forma tal que se comparan variables con valores usando operadores lógicos. Por ejemplo: la expresión ¿ Eres mayor de edad ? tiene implícito la evaluación de la variable "edad", la expresión ¿ Esta haciendo frio ?, tiene implícito la variable "temperatura ambiente", etc. En cada pregunta o expresión lógica debe determinarse la o las variables involucradas.

Expresión natural

- ¿ Tengo dinero ?
- ¿ Eres mayor de edad ?
- ¿ Esta haciendo frio ?
- ¿ Es de noche ?
- ¿ Eres más alto que yo ?

Expresión lógica

- dinero > 0
- edad > 17
- temperatura < 20
- luminosidad <= 1
- estatura_mia < estatura_tuya

En algunos casos puede que no sea tan fácil escribir la expresión natural como expresión lógica algorítmica. Por ejemplo, en la pregunta: "esta haciendo frío" puede ser relativo cuando es frío, porque para algunas personas frío puede ser debajo de 16 grados. La pregunta "es mayor de edad" tendría implícito también la pregunta "¿a qué edad se es mayor de edad?". En algunos países la mayoría de edad se cumple a los 21, en otros a los 18.

Los siguientes son ejemplos de expresiones que se pueden llevar a expresiones lógicas:

- ¿ se acabaron las entradas ?
- ¿ aprobé la materia ?
- ¿ mi hermano es más viejo que yo ?

- ¿ eres de la tercera edad ?
- ¿ estoy libre de deudas ?
- ¿ ya comenzó la película ?
- ¿ este año es bisiesto ?



Capítulo 4. ESTRUCTURAS SELECTIVAS. PARTE 1

4.1.- Concepto

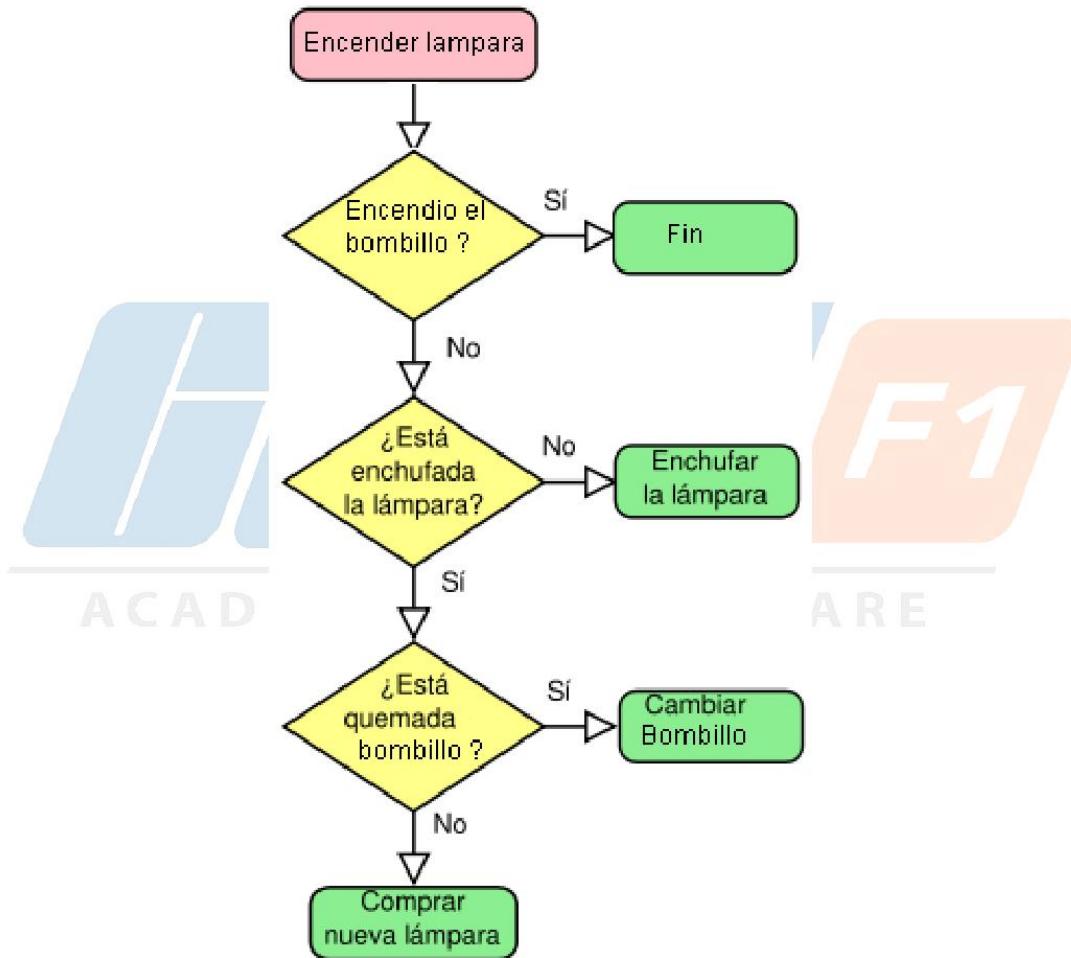
Hasta ahora se han explicado algoritmos llamados "secuenciales", donde todas las instrucciones que se colocan en el algoritmo son ejecutadas siempre, sin importar lo que suceda con los datos durante la ejecución del programa.

Una estructura selectiva es una instrucción que permite indicar al algoritmo que realice algunas instrucciones solo bajo algunas circunstancias, es decir, que estén condicionadas. Son también denominadas estructuras de control de flujo, estructuras condicionales o de toma de decisiones. Hay tres tipos de estructuras selectivas:

- simples
- dobles
- múltiples.



Una instrucción condicional debe tener una expresión lógica a evaluar. Las condiciones van a estar configuradas bajo una diversa gama de situaciones posibles, la idea es detectar cual o cuales instrucciones están condicionadas y por cual condición.



4.2.- Estructura Selectiva Simple

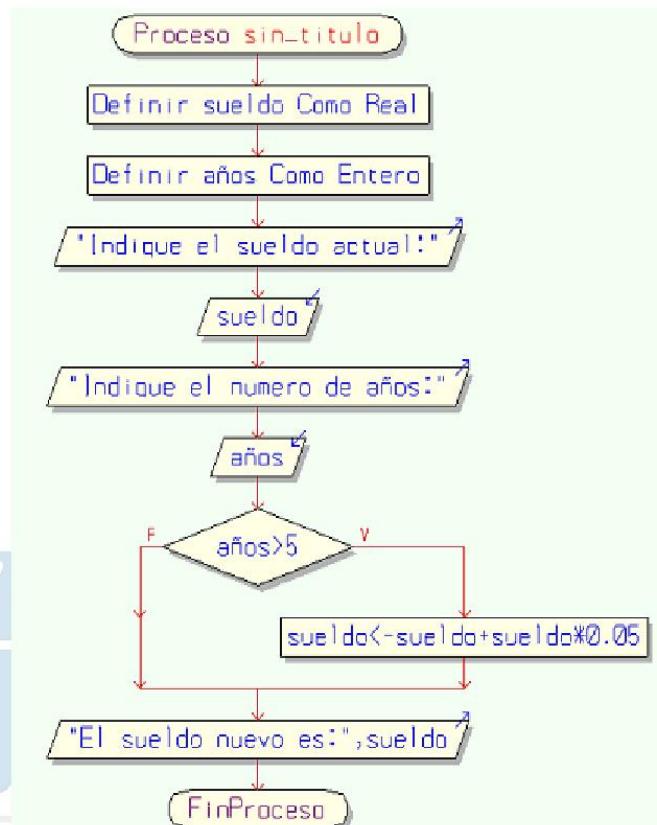
La estructura selectiva simple es la estructura condicional elemental y básica. Presenta la forma:

```
Si <expresion_logica> entonces  
    <Acciones>  
Fin si
```

En esta estructura condicional, si el resultado es verdadero se ejecuta el conjunto de acciones asociadas al bloque "entonces", si el resultado es falso no se ejecuta acción alguna. Por ejemplo: "me voy a poner una chaqueta si tengo frío", pero, que es tener frío ? Si para algunos tener frío es que la temperatura esté por debajo de 20 grados se escribe de la siguiente forma:

```
Si temperatura < 20 entonces  
    PonerChaqueta()  
fin si
```

En este caso si se tiene frío, se pone la chaqueta, en caso contrario no hace nada. El siguiente es un ejemplo de cómo aplicar el "si" en Psent, en el cual se le pagará un aumento de sueldo a un trabajador si tiene más de 5 años en la empresa:



ACADEMIA DE SOFTWARE

El algoritmo textual de este flujo gráfico se muestra a continuación. Es importante resaltar que el mensaje de salida se hace independientemente de si trabaja más de 5 años o no:

```
1 Proceso sin_titulo
2     definir sueldo Como Real
3     definir años Como Entero
4
5     mostrar "Indique el sueldo actual:"
6     leer sueldo
7     mostrar "Indique el numero de años:"
8     leer años
9
10    si años > 5 Entonces
11        sueldo = sueldo + sueldo*0.05
12    FinSi
13    mostrar "El sueldo nuevo es:", sueldo
14 FinProceso
```

Es importante poder determinar cuáles instrucciones están condicionadas y cuáles no. Colocar una instrucción dentro del "si" cuando debe estar afuera o viceversa traería un error lógico. En ejemplo anterior, las entradas se hacen siempre, debido a que no están condicionadas por el número de años que trabaja. La salida tampoco está condicionada, debido a que siempre se va a mostrar.

ACADEMIA DE SOFTWARE

Capítulo 5. ESTRUCTURAS SELECTIVAS. PARTE 2

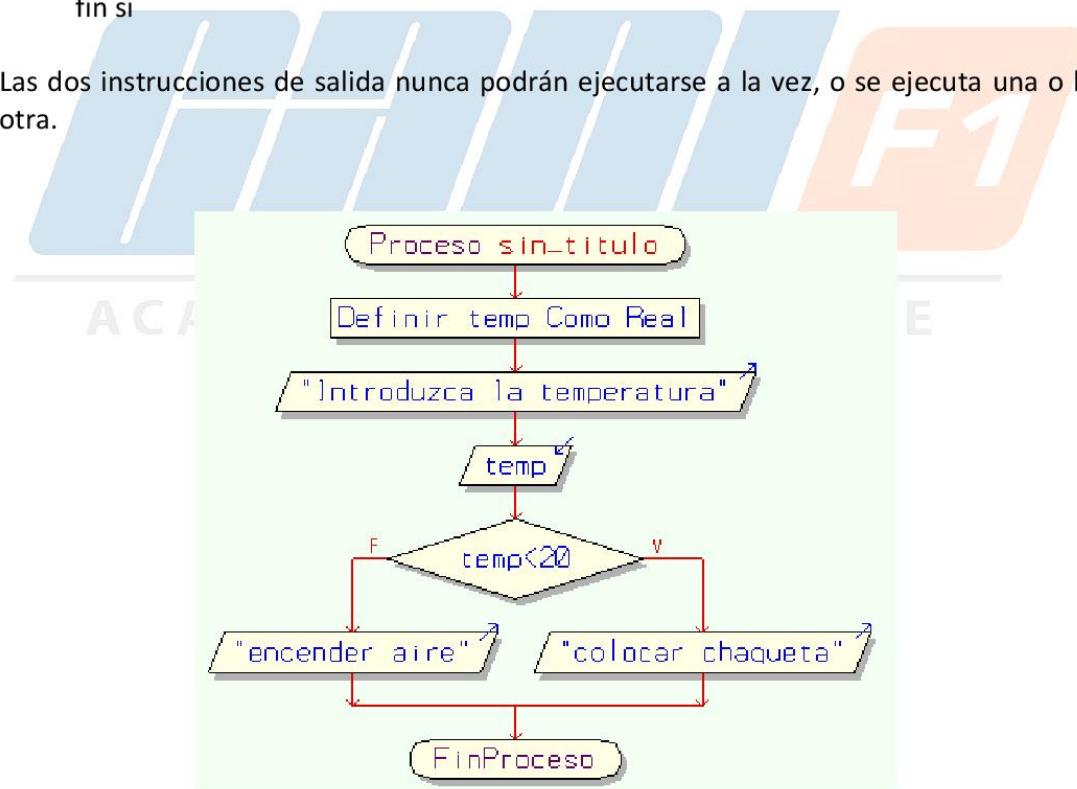
5.1.- Estructura Selectiva Doble

La estructura selectiva doble es una estructura condicional que evalúa una condición dada. Si es verdadera, se ejecutan el conjunto de acciones asociadas a la parte del "entonces", si es falsa se ejecuta el conjunto de acciones asociadas a la parte del "sino". En el ejemplo anterior, podemos hacer algo en caso de que no haya frío:

```

Si temperatura < 20 entonces
    mostrar "colocar chaqueta"
sino
    mostrar "encender aire"
fin si
  
```

Las dos instrucciones de salida nunca podrán ejecutarse a la vez, o se ejecuta una o la otra.



El siguiente es un ejemplo completo de cómo usar un condicional doble: Dado un monto calcular el descuento considerando que por encima de 100 el descuento es el 10% del monto y si es igual o menor que 100 el descuento es 2%:

```

1 Proceso sin_titulo
2     definir monto,descuento como real
3
4     mostrar "ingrese el monto"
5     leer monto
6
7     si monto > 100 entonces
8         descuento = monto * 10 /100
9     sino
10        descuento = monto * 2 /100
11    fin si
12
13    mostrar "El descuento es de " descuento
14
15 FinProceso

```

Es importante destacar que la instrucción selectiva doble no tiene una única forma de estructurarse, debido a que se puede usar una expresión lógica contraria, que también evalúe lo que se desea determinar. Por ejemplo, si se necesita saber si alguien es mayor de edad para dar un premio particular o si es menor de edad se le da otro premio, la instrucción se puede configurar de 2 formas equivalentes:

<pre> 1 Proceso sin_titulo 2 definir edad Como entero 3 4 mostrar "Introduzca la edad" 5 leer edad 6 si edad < 18 Entonces 7 mostrar "es menor de edad" 8 Sino 9 mostrar "es mayor de edad" 10 FinSi 11 FinProceso </pre>	<pre> Proceso sin_titulo definir edad Como entero mostrar "Introduzca la edad" leer edad si edad >= 18 Entonces mostrar "es mayor de edad" Sino mostrar "es menor de edad" FinSi FinProceso </pre>
--	---

Nótese que lo único que cambió fue la condición (se colocó la condición contraria) y por supuesto, las instrucciones que están en el "entonces" se colocan en el "sino" y viceversa.

5.2.- Aplicaciones de la Estructura Selectiva

Una de las aplicaciones más comunes de la instrucción selectiva doble es la de determinar si un número es positivo o negativo. Para esto hay que determinar si un número es mayor que cero o menor que cero. El siguiente es un ejemplo:

```
1 Proceso sin_titulo
2     definir nro Como Entero
3
4     mostrar "Introduzca un numero:"
5     leer nro
6     si nro > 0 Entonces
7         mostrar "El numero es positivo"
8     Sino
9         mostrar "El numero es negativo"
10    FinSi
11 FinProceso
```

ACADEMIA DE SOFTWARE

Otra aplicación muy frecuente de la selectiva doble es la de determinar si un número es par o impar. Se debe determinar si un número es divisible entre 2 o no. Para esto se utiliza el operador "resto de la división", que es Pselnt es el "%". Si el resto de la división de un número entre 2 es cero, significa que el número es par, de lo contrario, el número es impar. El siguiente es un ejemplo de cómo determinar si un número es par o impar:

```

1 Proceso sin_titulo
2     definir nro Como Entero
3
4     mostrar "Introduzca un numero:"
5     leer nro
6     si (nro % 2) = 0 Entonces
7         mostrar "El numero es par"
8     Sino
9         mostrar "El numero es impar"
10    FinSi
11 FinProceso

```

Otra aplicación de este tipo de estructura es combinarlas con funciones cuando se necesita hacer comparaciones de cadenas leídas por el teclado. Por ejemplo, se necesita determinar si un usuario introduce el valor "si" por el teclado. En este caso, el usuario puede escribir "SI", "Si", "sI" o "si", lo cual haría muy complicado hacer la comparación, debido a que se tendría que comparar con cada una de las posibilidades que el usuario puede introducir. Usando la función mayúscula o minúscula se simplifica este problema, debido a que se convierte la cadena introducida por el usuario y luego se compara el valor convertido. Por ejemplo:

```

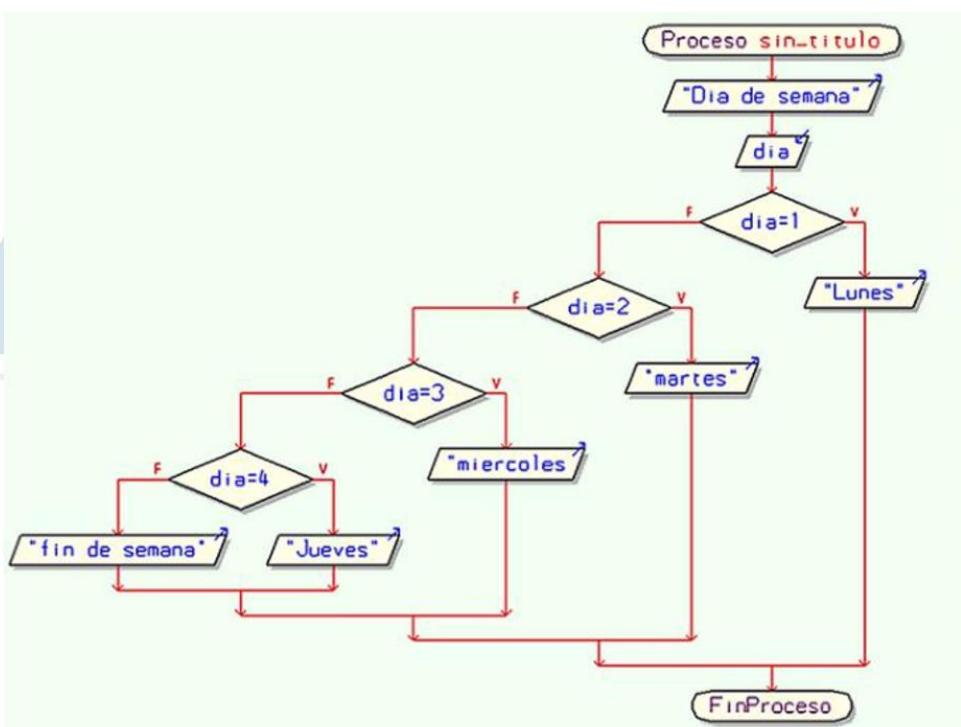
1 Proceso sin_titulo
2     definir respuesta como caracter
3
4     mostrar "Desea continuar Si o No ?"
5     leer respuesta
6
7     si Minusculas(respuesta) == "si" Entonces
8         mostrar "respondio que si"
9     Sino
10    mostrar "No respondio que si"
11    FinSi
12 FinProceso

```

Capítulo 6. ESTRUCTURAS SELECTIVAS. PARTE 3

6.1.- Anidación de Condicionales

En muchas situaciones, se necesita hacer una condición en caso que la evaluación de otra condición sea falsa o verdadera. En estos casos, se recomienda utilizar una anidación de sentencias condicionales. La ventaja de anidar sentencias condicionales, es que cuando una se cumple no hay por qué evaluar las condiciones que están debajo o que le siguen. Por ejemplo:



El algoritmo textual del flujograma anterior se muestra a continuación:

```

1  Proceso sin_titulo
2      Escribir "Dia de semana"
3      Leer dia
4      Si dia=1 Entonces
5          Escribir "Lunes"
6      Sino
7          Si dia=2 Entonces
8              Escribir "martes"
9          Sino
10         Si dia=3 Entonces
11             Escribir "miercoles"
12         Sino
13             Si dia=4 Entonces
14                 Escribir "Jueves"
15             Sino
16                 Escribir "fin de semana"
17             FinSi
18         FinSi
19     FinSi
20 FinSi
21 FinProceso

```

ACADEMIA DE SOFTWARE
La sintaxis general de las condiciones anidadas es la siguiente:

```

si <condicion> entonces

sino
    si <condicion> entonces

    sino
        si <condicion> entonces

        sino
            si <condicion> entonces

            sino

```

```

        fin si
    fin si
    fin si
fin si

```

6.2.- Comparación de Variables

Un ejemplo típico de uso de condiciones anidadas es el de comparar 2 variables "var1" y "var2" para saber si son iguales o cual de las 2 es la mayor. En este caso, pueden ocurrir 3 cosas:

- que "var1" sea mayor que "var2"
- que "var2" sea mayor que "var1"
- que "var1" y "var2" sean iguales.

El siguiente es un ejemplo de cómo comparar 2 variables para determinar cuál es la relación entre ellas:

```

1 Proceso sin_titulo
2     definir edad1,edad2 como entero
3
4     mostrar "Edad 1:"
5     leer edad1
6     mostrar "Edad 2:"
7     leer edad2
8
9     si edad1>edad2 entonces
10        mostrar "La mayor edad es ",edad1
11    sino
12        si edad1<edad2 Entonces
13            mostrar "La mayor edad es ",edad2
14        sino
15            mostrar "Tienen la misma edad"
16        FinSi
17    FinSi
18 FinProceso

```

6.3.- Evaluación de Intervalos

Otra aplicación típica de los "si" anidados es la evaluación de intervalos. A este se refiere, la revisión de si un valor está en algún intervalo de varios intervalos definidos. Por ejemplo, dada la edad de una persona se necesita saber cuál es su estado:

- Un bebé tiene una edad comprendida entre 0 y 2 años
- Un niño tiene edad comprendida entre 3 y 10 años
- Un adolescente tiene edad entre 11 y 17 años
- Un joven tiene edad entre 18 y 25 años
- Un adulto tiene edad entre 26 y 50 años
- Un adulto mayor tiene edad entre 51 y 64 años
- Un anciano tiene edad mayor a 64 años

En un caso como este, un valor de edad sólo puede estar en alguno de los intervalos definidos anteriormente.

El algoritmo textual que resuelve este problema es el siguiente:

```

1  Proceso intervalos_edades
2      Definir edad Como Entero
3      Mostrar "Introduzca la edad de la persona:"
4      leer edad
5      Si edad <= 2
6          Entonces Mostrar "Es un bebé"
7          Sino
8              Si edad <= 10
9                  Entonces Mostrar "Es un niño"
10             Sino
11                 Si edad <= 17
12                     Entonces Mostrar "Es un adolescente"
13                     Sino
14                         Si edad <= 25
15                             Entonces Mostrar "Es un joven"
16                             Sino
17                                 Si edad <= 50
18                                     Entonces Mostrar "Es un adulto"
19                                     Sino
20                                         Si edad <= 64
21                                             Entonces Mostrar "Es adulto mayor"
22                                             Sino Mostrar "Es un anciano"
23                                         FinSi
24
25
26
27     FinSi
28
29 FinProceso

```

Capítulo 7. ESTRUCTURAS SELECTIVAS. PARTE 5

7.1.- Operadores Lógicos

Cuando se estructuran expresiones lógicas, existen algunos casos en donde se requiere preguntar o evaluar más de una condición al mismo tiempo, donde se evalúen varias variables o donde una variable se compara con varios valores. Para esto, están los operadores lógicos.

Por esto es común tener expresiones que combinan tanto los operadores lógicos como relacionales. En estas expresiones se evalúa más de una condición o relación, por medio de operadores lógicos.

Los operadores lógicos más utilizados son:

Y
O
No

Conjunción (and, &&)
Disyunción (or, ||)
Negación (not, ~,!)

7.2.- Condiciones Compuestas

Una condición compuesta se define como dos o más condiciones simples unidas por los operadores lógicos. En muchas ocasiones es necesario plantear más de una condición para su evaluación al computador.

Por ejemplo: que el computador muestre la boleta de un alumno, si este estudia la carrera de medicina y su promedio de calificaciones es mayor de 70.

Otro ejemplo: para entrar a un parque infantil el niño debe entre 2 y 8 años de edad. Para este caso la condición se plantea así:

$$(\text{edad} \geq 2) \text{ Y } (\text{edad} \leq 8)$$

Ahora, el caso en el que para entrar un espectáculo, los niños menores de 8 años y los adultos mayores de 65 años están exentos de pagar la entrada. La estructura selectiva sería:

```
Si (edad < 8) O (edad > 65) Entonces  
    Costo_Eentrada=0  
FinSi
```

Si se quisiera hacer más específica la verificación condicional para la entrada preferencial a las personas de la 3ra edad: mujeres a partir de 55 y hombres a partir de 60, la condición sería:

(sexo="F" Y edad>=55) O (sexo="M" Y edad>=60)

De ser necesario, debe usarse paréntesis para priorizar evaluación de condiciones compuestas. A mayor complejidad de la expresión lógica, más determinante será el uso de dichos símbolos.



Capítulo 8. TABLA DE LA VERDAD

8.1.- Concepto

Una tabla de verdad, o tabla de valores de verdad, es una tabla que despliega el valor de verdad de una proposición compuesta, para cada combinación de valores de verdad que se pueda asignar a sus componentes.

Considérese 2 proposiciones: A y B. Cada una puede tomar uno de dos valores de verdad: o V (verdadero), o F (falso). Por lo tanto, los valores de verdad de A y B pueden combinarse de 4 maneras distintas:

- Ambas son verdaderas,
- A es verdadera y B falsa,
- A es falsa y B verdadera, o
- Ambas son falsas.

Esto puede expresarse con una tabla simple:

A	B
V	V
V	F
F	V
F	F

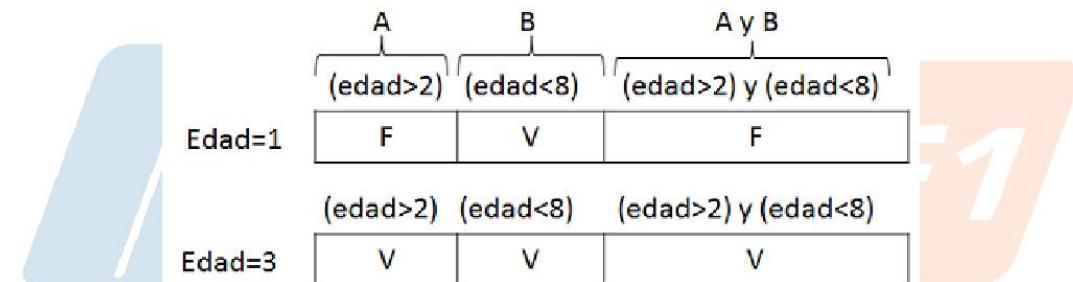
8.2.- Tabla de la Conjunción

La conjunción (Y) es un operador lógico que opera sobre dos valores de verdad, típicamente los valores de verdad de dos proposiciones, devolviendo el valor de verdad verdadero cuando ambas proposiciones son verdaderas, y falso en cualquier otro caso.

La tabla de verdad de la conjunción es la siguiente:

A	B	$A \wedge B$
V	V	V
V	F	F
F	V	F
F	F	F

Véase el ejemplo donde para entrar a un parque infantil el niño debe tener de 3 a 7 años.



AC

	A $(edad > 2)$	B $(edad < 8)$	$A \wedge B$ $(edad > 2) \text{ y } (edad < 8)$
Edad=1	F	V	F
Edad=3	V	V	V
Edad=15	V	F	F
Edad=2	F	V	F
Edad=8	V	F	F

Véase varias condiciones que debe cumplir una lámpara eléctrica para que pueda encender:

- el interruptor está conectado

- la lámpara no está fundida

En cualquier otro caso la lámpara no se encenderá. La condición se escribiría así:

```
Si (interruptorConectado=verdadero) Y (lamparaFundida=falso)
    Entonces EncenderLampara()
FinSi
```

Ahora ilustrando el ejemplo en la tabla de la verdad:

```
Si (interruptorConectado=verdadero) Y (lamparaFundida=falso) entonces
    encenderLampara()
```

A (interruptorConectado=verdadero)	B (lamparaFundida=falso)	A y B (interruptorConectado=verdadero) y (lamparaFundida=falso)
V	V	V
(interruptorConectado=verdadero)	(lamparaFundida=verdadero)	(interruptorConectado=verdadero) y (lamparaFundida=verdadero)
V	F	F
(interruptorConectado=Falseo)	(lamparaFundida=falso)	(interruptorConectado=Falseo) y (lamparaFundida=falso)
F	V	F
(interruptorConectado=Falseo)	(lamparaFundida=verdadero)	(interruptorConectado=Falseo) y (lamparaFundida=verdadero)
F	F	F

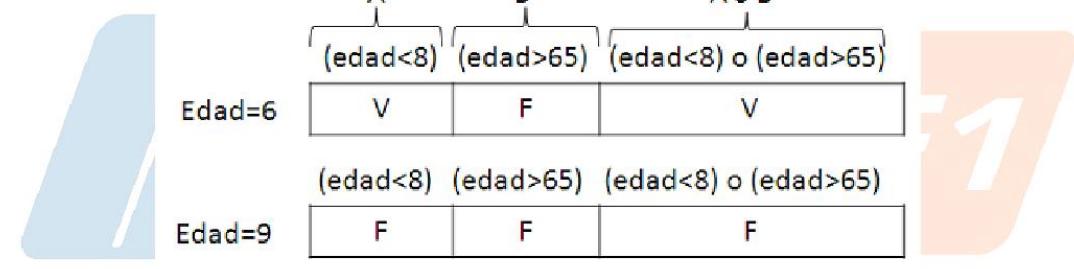
8.3.- Tabla de la Disyunción

La disyunción (\vee) es un operador lógico que opera sobre dos valores de verdad, devolviendo el valor de verdad verdadero cuando una de las proposiciones es verdadera, o cuando ambas lo son, y falso cuando ambas son falsas. Una disyunción es un enunciado con dos o más elementos optativos.

La tabla de verdad de la disyunción es la siguiente:

A	B	$A \vee B$
V	V	V
V	F	V
F	V	V
F	F	F

Ahora, el ejemplo en el que la entradas para entrar un espectáculo es gratis para los niños menores de 8 años y los adultos mayores de 65 años



	A $(edad < 8)$	B $(edad > 65)$	$A \vee B$ $(edad < 8) \vee (edad > 65)$
Edad=6	V	F	V
Edad=9	F	F	F
Edad=66	F	V	V
Edad=8	F	F	F
Edad=65	F	F	F

Nótese el ejemplo: si voy al cine o voy a cenar entonces gastaré algo de dinero; la conclusión de gastar algo de dinero es verdadera si la persona va al cine aunque no vaya a cenar, pero también es verdadera si solamente va a cenar y no va al cine. La única

forma en que la conclusión de gastar algo de dinero sea falsa, es que no vaya al cine ni a cenar, en este caso es una condición compuesta con dos variables distintas.

Esto se escribiría así:

```
Si (cine=verdadero) O (cenar=verdadero) entonces
    gastoDinero()
FinSi
```

En las siguientes evaluaciones se aprecia el resultado de evaluar cada condición simple sobre las variables cine y cenar, y luego la evaluación del resultado de la disyunción.

Nótese que:

- Siempre que al menos una es verdadera, la disyunción es verdadera.
- La disyunción sólo es falsa, cuando ambas expresiones son falsas.



A	B	A o B
(cine=verdadero)	(cenar=verdadero)	(cine=verdadero) o (cenar=verdadero)
V	V	V
(cine=verdadero)	(cenar=falso)	(cine=verdadero) o (cenar=falso)
V	F	V
(cine=falso)	(cenar=verdadero)	(cine=falso) o (cenar=verdadero)
F	V	V
(cine=falso)	(cenar=falso)	(cine=falso) o (cenar=falso)
F	F	F

Capítulo 9. MODULARIDAD. PARTE 1

9.1.- Concepto

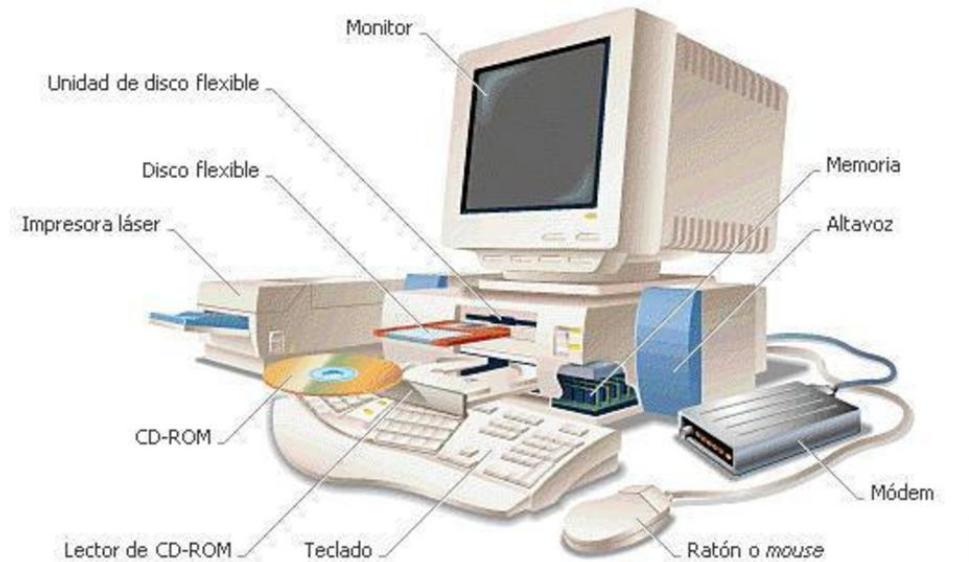
La solución de un problema complejo puede requerir muchos pasos. En la medida que el problema es más complejo o más grande, se hace más difícil crear el algoritmo que lo resuelve. Es necesario aplicar alguna estrategia. La estrategia más idónea es dividir el problema en sub-problemas más fáciles de resolver, que el problema principal. A este método se le denomina: divide y vencerás.

La modularidad es la capacidad que tiene un sistema de ser estudiado, visto o entendido, como la unión de varias partes que interactúan entre sí y que trabajan para alcanzar un objetivo común, realizando cada una de ellas, una tarea necesaria para la consecución de dicho objetivo.

Cada una de esas partes en que se divide un programa, recibe el nombre de Módulo. Idealmente, un módulo debe poder cumplir las condiciones de caja negra, es decir, ser independiente del resto de los módulos y comunicarse con ellos.

La computadora es un ejemplo de un equipo hecho de forma modular, debido a que cada una de las partes que la conforman se unen para lograr un fin común, pero cada una de ellas es independiente de la otra y se comunican entre sí para trabajar en conjunto.

Muchos de estos módulos poseen sub-módulos que lo componen, por ejemplo: el case posee internamente el CPU, el disco duro, la memoria, etc. El disco duro internamente puede también dividirse en módulos y así sucesivamente.



La modularidad tiene muchas ventajas, entre ellas las más notables:

- Simplifica el diseño.
- Disminuye la complejidad de los algoritmos.
- Ahorra en tiempo de programación gracias a la reusabilidad del código.
- Favorece el trabajo en equipo.
- Facilita la depuración y prueba.
- Facilita el mantenimiento.

9.2.- Definición de Módulos

En el contexto de la programación, un módulo (también conocido como sub-rutinas, funciones, procedimientos, sub-programas o sub-procesos) es un conjunto de instrucciones encerradas o enmarcadas bajo un nombre, que será usado posteriormente para invocar su ejecución.

La palabra reservada que se usará durante el curso para la definición de un módulo es la palabra "subproceso", seguido del nombre del módulo (cada lenguaje de programación

tiene una palabra reservada y/o forma particular de definir módulos), seguidamente, la lista de parámetros encerrados en paréntesis. Dentro de cada módulo, se colocan las instrucciones que serán parte de este. Finalmente, se coloca la palabra "fin subproceso", que indica el final de las instrucciones del módulo. No puede haber 2 módulos con el mismo nombre dentro del mismo programa. El siguiente es un ejemplo:

```
subProceso ejemplo1()
    // aqui van las instrucciones que va a ejecutar el modulo
Fin subProceso
```

En pseudocódigo los módulos son declarados normalmente antes del cuerpo principal. Un programa puede tener cualquier cantidad de módulos, los cuales pueden tener cualquier nombre. El orden en que se definen es relevante en algunos lenguajes de programación, cuando se usan entre ellos.

Lo más recomendable es que cada módulo tenga un nombre significativo y que se utilicen verbos en infinitivo, por ejemplo: calcular_montos, mostrar_salidas, etc. A continuación una serie de módulos de un Algoritmo Rutina_Diaria:

ACADEMIA DE SOFTWARE

```
subProceso vestir()
    buscarRopa
    plancharRopa
    ponerRopa
Fin subProceso

subProceso comer()
    calentarComida
    servirComida
    masticarComida
Fin subProceso

subProceso bañar()
    quitarRopa
    mojar_enjabonar
    secar
Fin subProceso
```

Un módulo debe resolver un problema específico del gran problema. La cantidad de módulos en un programa es muy relativa a la complejidad y amplitud del problema.

Los módulos deben ser pequeños para que sean claros y de poca complejidad. Cada módulo a su vez puede dividirse en varios módulos y así sucesivamente hasta que quede un módulo indivisible. Los criterios para dividir un módulo en sub-módulos es subjetivo, pero se pueden considerar los siguientes criterios:

- si el módulo es muy largo.
- si es muy complejo.
- si es un candidato a ser reusable.

9.3.- Llamado de Módulos

Definir los módulos es una parte de la modularidad. La otra parte es el llamado o invocación de los módulos. Llamar un módulo es la instrucción para indicar que se necesita que se ejecuten las instrucciones que están adentro del módulo. Si un módulo no es llamado, no se ejecuta.

Los módulos pueden llamarse desde:

- Otros módulos del algoritmo: en algunos lenguajes de programación el módulo que es llamado debe estar definido antes del módulo que lo llama.
- El cuerpo principal del algoritmo: el orden en que se crean los módulos no es importante.

Cuando se ejecuta un programa, se ejecuta cada instrucción que esté en el cuerpo principal. Cuando el cuerpo principal llama a un módulo, se pasa el control del programa a ese módulo y se ejecuta cada instrucción del mismo. Cuando se ejecuta la última instrucción del módulo, se devuelve el control al cuerpo principal, para que ejecute la siguiente instrucción.

Para que un módulo se ejecute, se hace el llamado desde donde sea conveniente. Se hace a través del nombre del módulo seguido de los parámetros (si los tiene).

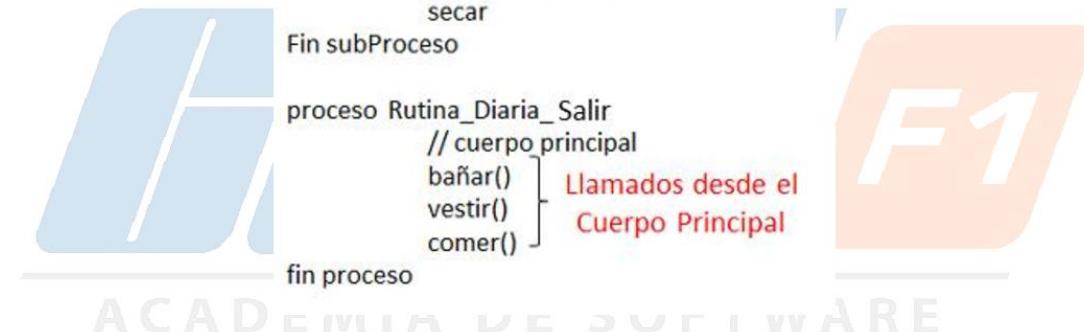
A continuación se muestra como llamar los módulos sin parámetros desde el cuerpo principal del programa:

```
subProceso vestir()
    buscarRopa
    plancharRopa
    ponerRopa
Fin subProceso

subProceso comer()
    calentarComida
    servirComida
    masticarComida
Fin subProceso

subProceso bañar()
    quitarRopa
    mojar_enjabonar
    secar
Fin subProceso

proceso Rutina_Diaria_Salir
// cuerpo principal
bañar()
vestir()
comer() ] Llamados desde el
                     Cuerpo Principal
fin proceso
```



Como llamar un módulo sin parámetros desde otro módulo:

```

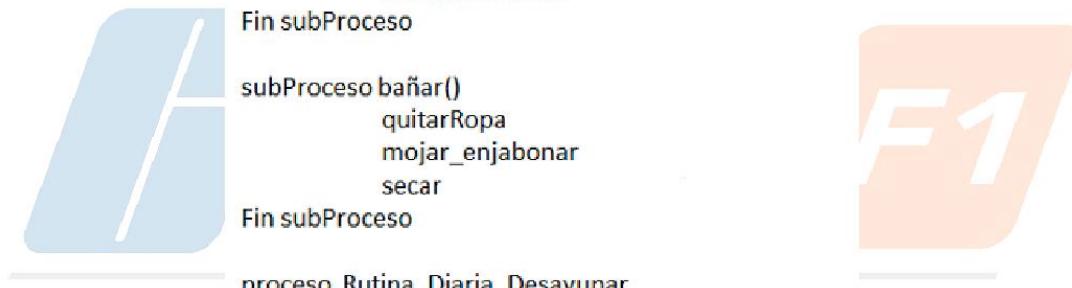
subProceso vestir()
    buscarRopa
    plancharRopa
    ponerRopa
Fin subProceso

subProceso calentarComidaMicroondas()
    ingresarComida
    seleccionarTiempo
    retirarComida
fin Subproceso
Llamado desde otro modulo
subProceso comer()
    calentarComidaMicroondas()
    servirComida
    masticarComida
Fin subProceso

subProceso bañar()
    quitarRopa
    mojar_enjabonar
    secar
Fin subProceso

proceso Rutina_Diaria_Desayunar
    // cuerpo principal
    bañar()
    vestir()
    comer()
fin proceso

```



Un módulo puede ser llamado a ejecutarse tanta veces sea necesario. Puede hacerse desde el cuerpo principal o desde otros módulos (incluso en un mismo módulo puede llamarse varias veces). A esto se le conoce como reusabilidad.

No es necesario definir el módulo las veces que se necesite que se ejecute. Es suficiente con definirlo una vez y hacer los llamados tantas veces como sea necesario.

```

subProceso vestir()
    buscarRopa
    plancharRopa
    ponerRopa
Fin subProceso

subProceso calentarComidaMicroondas()
    ingresarComida
    seleccionarTiempo
    retirarComida
fin Subproceso

subProceso comer()
    calentarComida Microondas()
    servirComida
    masticarComida
Fin subProceso

subProceso cepillar()
    untarPastaDentalCepillo
    cepillarDientes
    enjuagar
Fin subProceso

proceso Rutina_Diaria
    cepillar() ← Reusabilidad
    vestir()
    comer()
    cepillar() ←
fin proceso

```



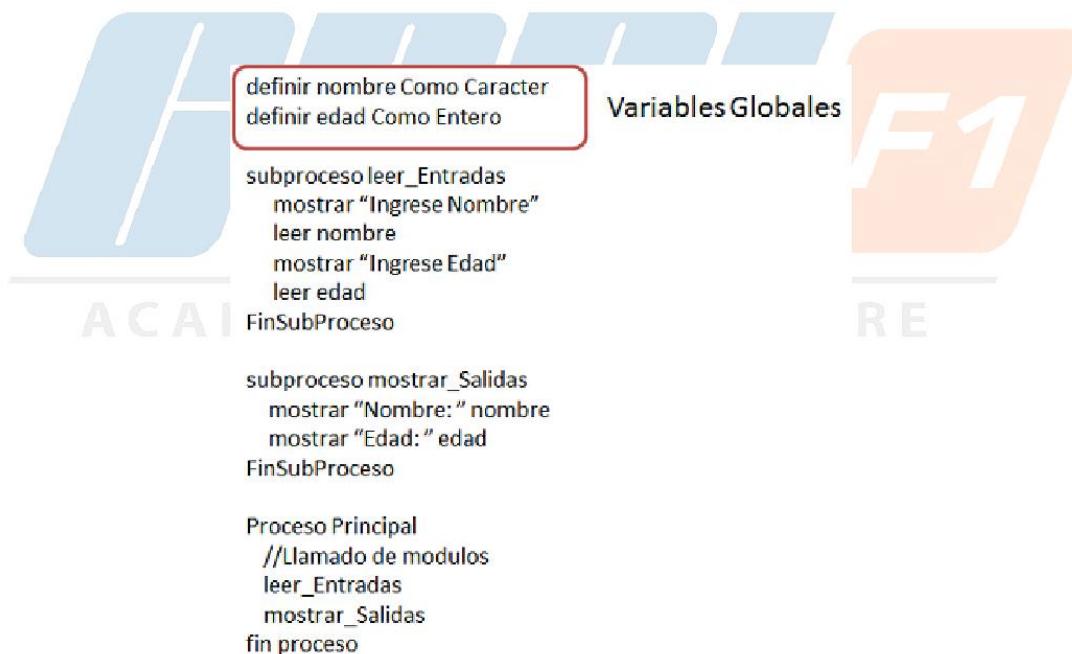
Capítulo 10. VARIABLES: GLOBALES Y LOCALES

10.1.- Variables Globales

Son variables que se declaran afuera de los módulos y pueden ser usadas en cualquier módulo del programa y/o en el cuerpo principal del programa. Una variable global ocupa espacio en la memoria RAM durante la ejecución de todo el programa.

El uso de variables globales no es recomendado, debido a que puede hacer que los programas sean más difíciles de entender y se den ambigüedades.

Este sería un ejemplo de variables globales, sin embargo, PSeInt no admite el uso de estas variables:



10.2.- Variables Locales

Son variables que se declaran dentro de un módulo y solo pueden usarse en el módulo donde fueron declaradas. El objetivo de una variable local es apartar la memoria necesaria para las variables sólo en el momento en que se van a usar. El siguiente es un ejemplo de declaración de variables locales:

```
subproceso calcularSueldo  
    definir bono, aumento como real // variables locales  
  
    bono = nrohijos*5  
    aumento = sueldobase * 0.1  
    sueldoNuevo = sueldobase + aumento + bono  
  
fin subprocesso
```

En el ejemplo anterior, las variables "bono" y "aumento" son variables locales del módulo "calcularSueldo" y solo pueden ser usadas dentro de ese módulo. Las variables locales son usadas normalmente para hacer cálculos auxiliares previos a los cálculos finales.

IMPORTANTE:

Para entenderlo de un modo práctico, si se declara una variable, pero ella no está catalogada como entrada o una salida del programa, debe ser declarada como variable local del módulo que la utiliza. Dos o más módulos pueden tener variables locales con el mismo nombre sin que esto genere conflictos.

10.3.- Ámbito de Las Variables

Una de las características de la programación modular es la independencia de los módulos. Esta independencia es importante, no sólo porque puede ser diseñado sin conocimiento del diseño de otros módulos, sino también porque la ejecución de un subproceso particular no afectará a los valores de las variables que pertenecen a otros subprocesos. Esto se vuelve más complejo en los lenguajes de programación que permiten el anidamiento de módulos.

Por esta razón, es preciso definir un nuevo concepto de gran importancia: el ámbito de una variable. Se conoce como ámbito de una variable la parte de un programa donde ésta es relevante, en otras palabras, todos aquellos lugares de un programa donde se puede utilizar la variable.

En relación al ámbito, se cumplen las siguientes reglas:

- Una variable local solo puede ser usada en el bloque (módulo) donde se declara.
- En un mismo bloque no se pueden duplicar los nombres de las variables.
- Una variable se puede declarar con el mismo nombre en bloques diferentes (módulos).
- Si una variable local tiene el mismo nombre de una variable global, al hacer referencia al nombre de la variable dentro del módulo, se accede a la variable local.

En el siguiente ejemplo se puede demostrar el funcionamiento del ámbito de las variables. Las salidas del siguiente programa serían:

```
50 , 20 , 25      (Cuerpo Principal)
25 , 10 , 40 , 50 (Modulo A)
30 , 45 , 60 , 10 (Modulo B)
25 , 45 , 25      (Cuerpo Principal)
```

Programa
variable x,y,z

Modulo_A
variable y,w,z
y<-10
x<-25
w<-50
z<-40
Imprimir x,y,z,w

x al no ser una
variable local,
modifica la
variable global
(x=25)

Modulo_B
variable x,t,z
x<-30
z<-60
t<-10
y<-45
Imprimir x,y,z,t

y al no ser una
variable local,
modifica la
variable global
(y=45)

x<-50
y<-20
z<-25
Imprimir x,y,z
Modulo_A
Modulo_B
Imprimir x,y,z



Capítulo 11. COMUNICACIÓN ENTRE MÓDULOS. PARTE 1

11.1.- Concepto

Todo lo que existe en un módulo es inaccesible al resto de los módulos o al programa principal. Por esta razón, la programación modular exige una comunicación entre el módulo llamador y el módulo llamado. Esta comunicación se realiza a través de unas variables de enlace que se denominan parámetros.

Un parámetro es un dato que se envía o que es retornado por un módulo del programa. Un módulo puede no tener parámetros, puede tener un solo parámetro o varios parámetros. La cantidad de parámetros del módulo dependerá del objetivo que cumple el módulo.

En la modularidad, lo que se busca es que cada módulo tenga un solo objetivo bien definido. Para lograr este objetivo posiblemente necesite datos provenientes de otros módulos.

Por ejemplo, si se necesita crear un módulo para enviar mensajes de texto, se define un módulo con el nombre "EnviarMensaje", el cual necesitará dos parámetros: el número de teléfono al cual se va a enviar el mensaje y el mensaje mismo. Estos dos (2) datos pueden provenir de un módulo que solicite al usuario estos datos, o de un módulo que los lea de una base de datos externa o pueden provenir de otro módulo que lee mensajes, etc.

Entonces, los parámetros son variables cuyos valores provienen de otros módulos o tal vez sean determinados adentro del módulo y sean retornados a otros módulos. En el mismo ejemplo de un módulo "EnviarMensaje", el módulo podrá retornar un parámetro que indique si el mensaje fue enviado o no.

El uso de parámetros en la modularidad permite que la reusabilidad de código sea más provechosa, ya que se puede usar el mismo código en diversas situaciones que son idénticas, donde lo único que cambia son los datos. Si se necesita enviar 100 mensajes, el procedimiento es el mismo, lo único que cambiará serán los números de teléfono y el mensaje.

Sin reusabilidad, se tendría que escribir el mismo código 100 veces, donde lo único que cambiaría serían estos dos (2) datos: número de teléfono y mensaje. Es por esto que la modularidad, haciendo uso de la reusabilidad, permite ahorrar mucho código, y por ende, tiempo.

11.2.- Parámetros Formales

Cuando se define un módulo, si se determina que éste necesita parámetros o retorna parámetros, en la definición del módulo deben indicarse. A esto se le llama "definición de los parámetros". Los parámetros que se colocan en la definición de los módulos se les llaman parámetros formales. Ahí se deben colocar entre paréntesis al lado del nombre módulo, con el respectivo tipo de dato de cada parámetro. Si son varios parámetros, se separan con comas (,). La forma general de colocar los parámetros es la siguiente:

```
subProceso nombreModulo ( parametro1 : tipoDatos, parametro2 : tipoDatos )  
.....  
fin subprocesso
```

En Pselnt no es necesario colocar el tipo de dato del parámetro. La forma de declarar los parámetros es:

```
subProceso ejemplo1 ( parametro1, parametro2, parametro3, ..., parametro n )  
.....  
fin subprocesso
```

Siguiendo el ejemplo del envío de mensajes:

```
subproceso EnviarMensaje ( nroTelefono , mensaje)  
.....  
fin subprocesso
```

Como ejemplo se tiene un módulo encargado de solicitar los datos de salida al usuario. Estos datos de entrada son los parámetros del módulo, por tanto deben listarse en la definición del módulo:

```

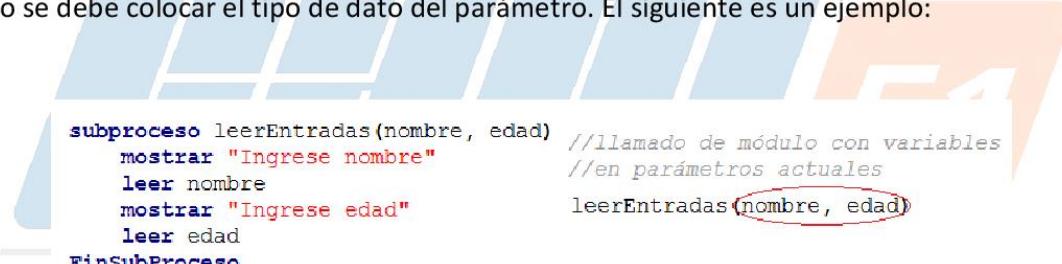
subproceso leer_entradas(nombre, edad)
    mostrar "Ingrese nombre"
    leer nombre
    mostrar "Ingrese edad"
    leer edad
FinSubProceso

```

↑
parámetros formales

11.3.- Parámetros Actuales

Cuando un módulo tiene parámetros en su definición (parámetros formales), el llamado de este módulo también debe tener parámetros. En el llamado se deben colocar los parámetros que el módulo necesita. A estos parámetros se les denomina parámetros actuales. Pueden variables o valores literales. A diferencia de los parámetros formales, no se debe colocar el tipo de dato del parámetro. El siguiente es un ejemplo:



```

subproceso leerEntradas(nombre, edad)
    mostrar "Ingrese nombre"
    leer nombre
    mostrar "Ingrese edad"
    leer edad
FinSubProceso

subproceso mostrarInf(nombre, edad)
    mostrar "El alumno: " nombre
    mostrar "tiene " edad " años"
FinSubProceso

```

//llamado de módulo con variables
//en parámetros actuales
leerEntradas(**nombre**, **edad**)

//llamado de módulo con valores
//en parámetros actuales
mostrarInf("Jose Luis", 16)

11.4.- Relación Entre Los Parámetros

Entre los parámetros actuales y formales existe una relación muy estrecha. Se deben cumplir las siguientes condiciones:

1.- Deben tener la misma cantidad: si un módulo tiene 5 parámetros formales, en el llamado del módulo deben colocarse obligatoriamente 5 parámetros actuales. Ni más ni menos. De lo contrario ocurrirá un error. Veamos un ejemplo de errores por cantidad de parámetros:

```

subproceso leer_Entradas(nombre)
    mostrar "Ingrese Nombre"
    leer nombre
FinSubProceso

subproceso procesar(edad, bono)
    si edad<18 entonces
        | bono=500
    Sino
        | bono=250
    FinSi
FinSubProceso

```

```

subproceso mostrar_Salidas(bono)
mostrar "Bono: " bono
FinSubProceso

```

```

Proceso Principal
+ leer_Entradas(nombre, edad)
+ procesar(edad) ←
    mostrar_Salidas(bono)
fin proceso

```

Error. Hay mas parámetros actuales que formales

Error. Hay mas parámetros formales que actuales

Correcto

2.- Deben ser del mismo tipo: si el módulo define que recibe 2 parámetros tipo real, no se pueden colocar en el llamado del módulo 2 parámetros alfanuméricos. Se deben colocar 2 parámetros actuales del mismo tipo que los formales. Errores por no coincidir el tipo de dato:

```

subproceso leer_Entradas(nombre, edad)
mostrar "Ingrese Nombre"
leer nombre
mostrar "Ingrese Edad"
leer edad
FinSubProceso

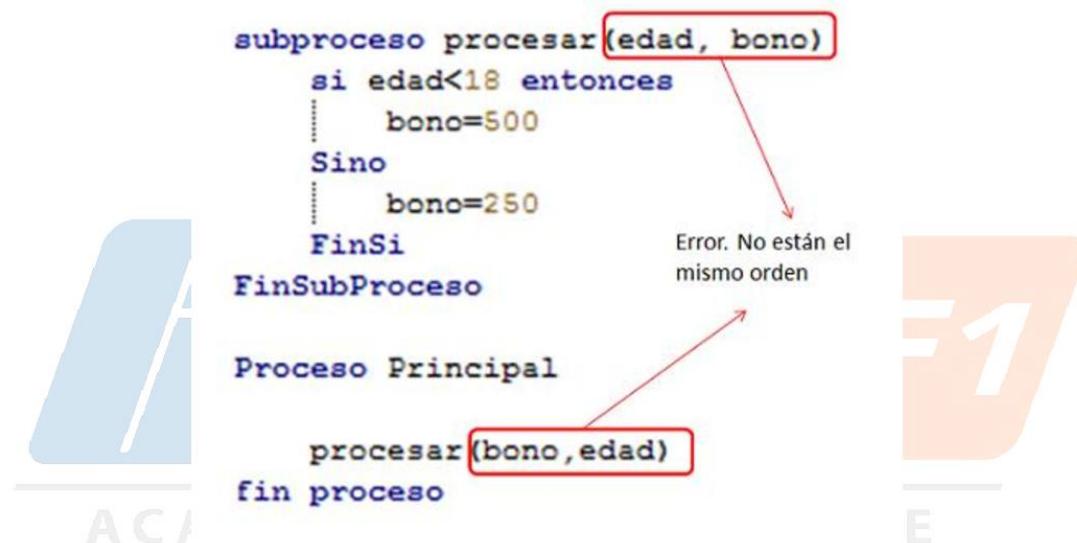
Proceso Principal
definir nombre como carácter
definir edad como entero
leer_Entradas(edad, nombre)

fin proceso

```

No coinciden los tipos

3.- Deben estar en el mismo orden: si un módulo define 2 parámetros reales, uno indica la estatura y el otro el peso, al ser llamado el módulo, deben colocarse los parámetros actuales en el mismo orden, primero la estatura y de segundo el peso. De no respetar el orden, aunque no ocurrirá un error de compilación, los resultados del módulo muy probablemente serán incorrectos. Ejemplo de no colocar los parámetros formales y actuales en el mismo orden:



No es necesario que tengan el mismo nombre, aunque inicialmente se usa el mismo nombre para evitar confusión. Es decir, los parámetros formales y actuales pueden tener nombres distintos, aun cuando se refieren al mismo dato.

Ejemplo: se puede definir un módulo con los parámetros formales nombre, apellido, edad. Y al invocar este módulo se puede hacer usando los parámetros actuales n, a, e.

Para que el compilador sepa quién se refiere a quién, es suficiente con cumplir las tres condiciones que establece la relación entre parámetros.

```

proceso sin_titulo

    entradas(n,a,e)

Finproceso

subproceso entradas(nombre, apellido, edad)
    mostrar "Ingresar Nombre"
    leer nombre
    mostrar "Ingresar Apellido"
    leer apellido
    mostrar "Ingresar edad"
    leer edad
FinSubProceso

```

Que los parámetros formales y actuales pueden tener distintos nombres juega un papel importante en la reusabilidad.

Ya que se puede definir un módulo con parámetros y que pueda usarse varias veces pasándole parámetros distintos en información. Solo hay que cumplir las 3 condiciones ya mencionadas.

```

subproceso mostrarPromedio(nombre,nota1,nota2,nota3)
    definir promedio como real
    promedio=(nota1+nota2+nota3)/3
    mostrar "Alumno: " nombre
    mostrar "Promedio: " promedio " Pts."
FinSubProceso

//llamado de cuerpo principal o desde un modulo
    mostrar "Ingresar nombre "
    leer n
    mostrar "Ingresar Nota 1, Nota 2, Nota 3"
    leer n1, n2, n3
//llamado de modulos
    mostrarPromedio(n, n1, n2, n3)
    mostrarPromedio("Juan", 18,15,20)

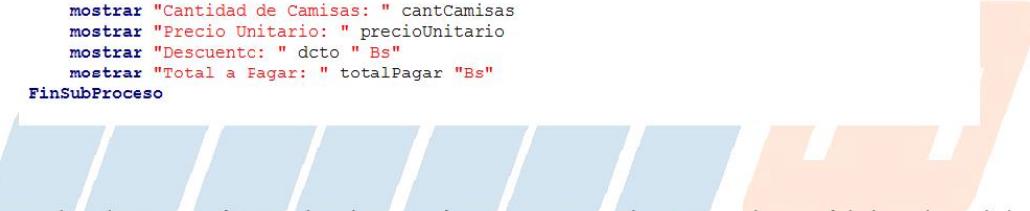
```

Capítulo 12. COMUNICACIÓN ENTRE MÓDULOS. PARTE 2

12.1.- Parámetros Por Valor

Los parámetros por valor (o parámetros de entrada) son los más frecuentemente usados. Un parámetro "por valor" indica que el valor del parámetro entrará al módulo, pero no sufrirá ningún cambio adentro de éste. Para que un parámetro sea pasado "por valor", no se debe escribir nada en particular en la definición ni en el llamado de los módulos. Al tener definido uno o varios parámetros formales en un módulo, ya ellos son, por defecto, parámetros por valor.

```
SubProceso mostrarFactura(nombre, direccion, cantCamisas, precioUnitario, dcto, totalPagar)
    mostrar "Nombre: " nombre
    mostrar "Direccion: " direccion
    mostrar "Cantidad de Camisas: " cantCamisas
    mostrar "Precio Unitario: " precioUnitario
    mostrar "Descuento: " dcto " Bs"
    mostrar "Total a Pagar: " totalPagar "Bs"
FinSubProceso
```



El caso donde son más usados los parámetros por valor es en los módulos de salidas, debido a que en estos módulos los parámetros nunca son modificados, solamente son mostrados en pantalla al usuario final. En el ejemplo del módulo que envía mensajes de texto, los parámetros "numero" y "mensaje" serían parámetros de entrada, porque adentro del módulo no serán modificados. En Pselnt, por defecto los parámetros se pasan por valor, es decir no es necesario hacerles una marca especial pero si se desea especificarlos basta con escribir al lado del parámetro "por valor".

Ambos módulos son equivalentes:

<pre>subproceso mostrarI(bono) mostrar "El bono a recibir es:" mostrar bono FinSubProceso</pre>	<pre>subproceso mostrarI(bono por valor) mostrar "El bono a recibir es:" mostrar bono FinSubProceso</pre>
---	--

Si un módulo tiene parámetros por valor en su definición, cuando se haga el llamado de este, los parámetros pueden ser variables que ya contengan información o valores

literales. En el caso de que el valor literal sea tipo alfanumérico, se debe hacer uso de las comillas dobles.

```

SubProceso mostrarFactura (nombre, cantCamisas, precio)
    mostrar "Nombre: " nombre
    mostrar "Cantidad de Camisas: " cantCamisas
    mostrar "Precio Unitario: " precio
    mostrar "Total a Pagar: " cantCamisas * precio "Bs"
FinSubProceso

Algoritmo ejemplo
    //llamado con valores literales
    mostrarFactura ("Ana Garcia", 24, 2500)
    nombre="Juan Gonzales"
    cant=6
    //llamado con 2 variables y 1 valor literal
    mostrarFactura (nombre, cant, 2500)
FinAlgoritmo

```

12.2.- Parámetros Por Referencia

Los parámetros por referencia (de salida), son aquellos parámetros que si su valor es modificado adentro del módulo, su valor debe salir para estar disponible a otro módulo. Para que un parámetro sea pasado por referencia en PSeInt se debe colocar "Por Referencia" luego del parámetro, solo en la definición del módulo, el llamado no requiere algo en especial. Esta palabra reservada varía de un lenguaje a otro.

Uno de los usos más clásicos de estos parámetros es el módulo donde se leen las entradas, debido a que es ahí donde toman el valor leído por el teclado, para luego pasar esos valores a los demás módulos.

```
subproceso leer_datos(nombre por referencia, edad por referencia)
    mostrar "Ingrese nombre"
    leer nombre
    mostrar "Ingrese edad"
    leer edad
FinSubProceso
```

Los parámetros por referencia en el llamado de un módulo solo deben ser variables.

Proceso ejemplo

```
definir nombre1, nombre como caracter
definir edad1 Como entero
leer entradas(nombrer1, edad1)

//Error, el segundo parametro no puede ser un valor literal
leer entradas(nombre, 28);

FinProceso

subproceso leer_entradas(nombre por referencia, edad por referencia)
    mostrar "Ingrese nombre"
    leer nombre
    mostrar "Ingrese edad"
    leer edad
FinSubProceso
```

12.3.- Módulos Con Ambos Parámetros

Un módulo puede tener sólo parámetros "por referencia", o sólo parámetros "por valor" o ambos. No deben estar en un orden especial, es decir, no tienen que estar primero los "de referencia" y luego los "por valor", o viceversa. Incluso pueden estar intercalados.

Los módulos que se encargan de los procesos, en la mayoría de casos, deben tener parámetros de entrada y salida (por valor y por referencia), ya que deben recibir información, procesarla y enviar información nueva a otros módulos

```
Proceso ejemplo
  definir nombrel, nombre como carácter
  definir edad1 Como entero
  definir bono como real

  leer_entradas(nombrel, edad1)
  procesar(edad1, bono)

FinProceso
subproceso procesar(edad, bono por referencia)
  si edad<18
    bono=500
  Sino
    bono=1000
  FinSi
FinSubProceso
```

ACADEMIA DE SOFTWARE

Capítulo 13. MODULARIDAD. PARTE 2

13.1.- El Cuerpo Principal

El código fuente de un programa se escribe en una sección denominada "Cuerpo Principal" del programa. En algunos lenguajes de programación se conoce como el "main". Cuando un programa se ejecuta, se ejecutan las instrucciones que están en esta sección. En la programación secuencial, todas las instrucciones de los programas se colocan en esta sección, lo que implica que cuando un algoritmo tiene muchas líneas de código sería difícil de entender, mejorar y corregir errores.

Ahora bien, en la programación modular se busca que en el cuerpo principal se coloquen la menor cantidad de instrucciones posibles. La mayoría de las instrucciones deberían estar en los módulos. No existe una regla general sobre crear programas modulares, generalmente se crean primero los módulos y luego las instrucciones que estarán en el cuerpo principal.

Entonces, un algoritmo modular se basa en uno o varios módulos y un único cuerpo principal.

La división de un algoritmo se hace en base a los requerimientos del mismo. Un algoritmo sencillo debería tener al menos 3 módulos: leer_entradas, procesar_datos y mostrar_salidas (no necesariamente con esos nombres). En algunos casos, el módulo procesar_datos es el que tiende a ser sub-dividido debido a que es en este módulo es donde está la parte más compleja del programa. En general, un programa modular quedará de la siguiente forma:

```

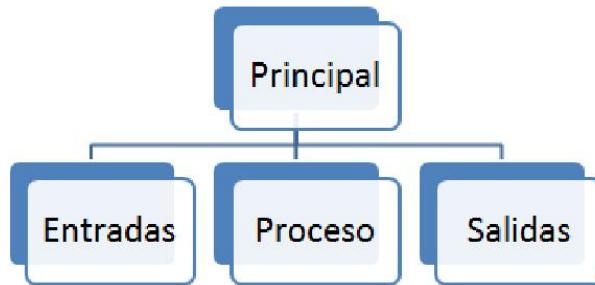
Algoritmo sin titulo
    //definicion de variables
    definir var1 como caracter
    definir var2, var3 como real
    definir var3, var5, var6 como entero
    //llamado de modulos
    procesol( var2, var5, varn )
    proceso2( var3, var1, var6, varn )
FinAlgoritmo

subproceso procesol(parametro1, parametro2, parametron )
    //cuerpo del proceso 1
FinSubProceso
subproceso proceso2(parametro1, parametro2, parametron )
    //cuerpo del proceso 2
FinSubProceso

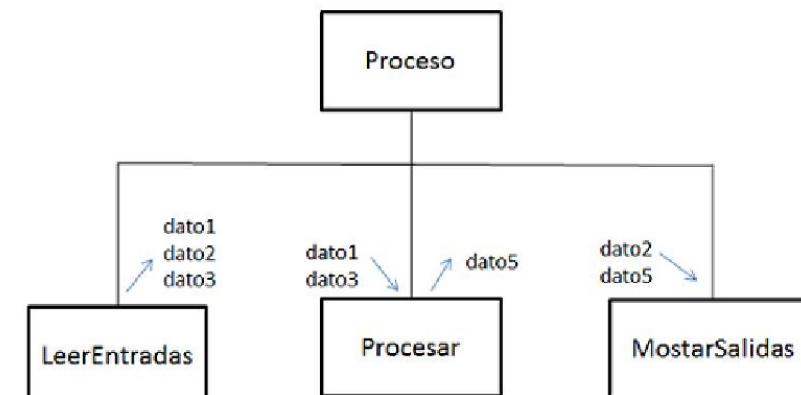
```

13.2.- Estructura Jerárquica

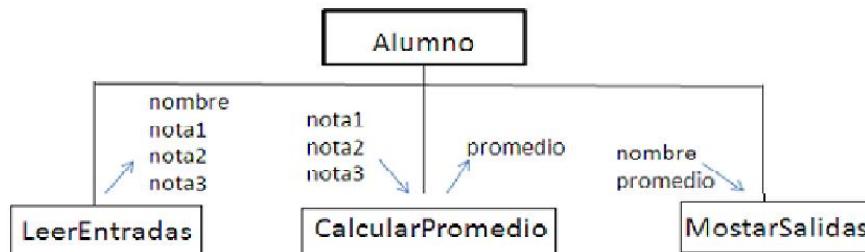
Si un algoritmo es sencillo se divide en tres módulos: entradas, proceso y salidas. Esta división puede plasmarse en un dibujo que muestre la relación que existe entre los módulos. A esta representación gráfica se le llama "Estructura Jerárquica". A continuación, se presenta un ejemplo de la división modular mínima de un programa, a través de un diagrama jerárquico:



Entre los módulos debe existir una comunicación que se hace a través de los parámetros. Entonces cada módulo, debe contar con una cantidad de datos para que pueda funcionar. Estos parámetros pueden ser de entrada y/o de salida y se pueden mostrar en el diagrama jerárquico, haciendo uso de flechas para distinguir los parámetros de entrada de los parámetros de salida.



Por ejemplo, para resolver el siguiente enunciado: "Se leen nombre de un alumno y sus 3 notas parciales. Calcular, mostrar el nombre y la nota promedio del alumno", será necesario crear 3 módulos: LeerEntradas, CalcularPromedio y MostrarSalidas. La siguiente imagen muestra el diagrama jerárquico para resolver el problema:



En el módulo LeerEntradas se solicitan los datos de entrada al usuario: nombre, nota1, nota2, y nota3. Estos deben salir de este módulo para ser usados en otros módulos. En el módulo CalcularPromedio se reciben los datos nota1, nota2, y nota3; los cuales vienen del módulo LeerEntradas. Con estos datos se calculará el promedio y debe salir de este módulo para ser usado en el último módulo MostrarSalidas, donde va a ser mostrado por pantalla junto con el dato nombre que viene del módulo LeerEntradas.

Luego de crear el diagrama, se debe escribir el código fuente del programa, haciendo que sea coherente con el mismo. El algoritmo correspondiente al diagrama jerárquico del ejemplo anterior es el siguiente:

```

Proceso ejemplo
    definir nombre Como Real
    definir nota1, nota2, nota3 como entero
    definir promedio como real
    LeerEntradas(nombre, nota1, nota2, nota3)
    CalcularPromedio(nota1, nota2, nota3, promedio)
    MostrarSalidas(nombre, promedio)
FinProceso

subproceso LeerEntradas(nombre por referencia, n1 por referencia,
                           n2 por referencia, n3 por referencia)
    mostrar "Ingrese Nombre"
    leer nombre
    mostrar "Ingrese Notas: "
    leer n1, n2, n3
FinSubProceso

subproceso CalcularPromedio(n1, n2, n3, p por referencia)
    p=(n1+n2+n3) /3
FinSubProceso

subproceso MostrarSalidas(n, p)
    mostrar "Alumno: " n
    mostrar "Promedio: " p "puntos"
FinSubProceso

```

13.3.- Reusabilidad Con Parámetros

Ya se comprobó que no es estrictamente necesario que los parámetros formales y actuales de un módulo tenga el mismo nombre. Entonces se puede aprovechar esa ventaja para reutilizar un módulo con parámetros, y pasarle en cada llamado información (parámetros actuales) diferente. Por ejemplo:

```
Algoritmo sin_titulo
    //empleado1
    b=5000
    d=1500
    calcularSueldo(b,d, sf)
    mostrar "Sueldo Final Empleado 1: " sf
    //empleado2
    calcularSueldo(9000,3000, sf)
    mostrar "Sueldo Final Empleado 1: " sf
finAlgoritmo

subproceso calcularSueldo(bono, deduccion, sueldo por referencia)
    sueldo=200000+bono-deduccion
FinSubProceso
```



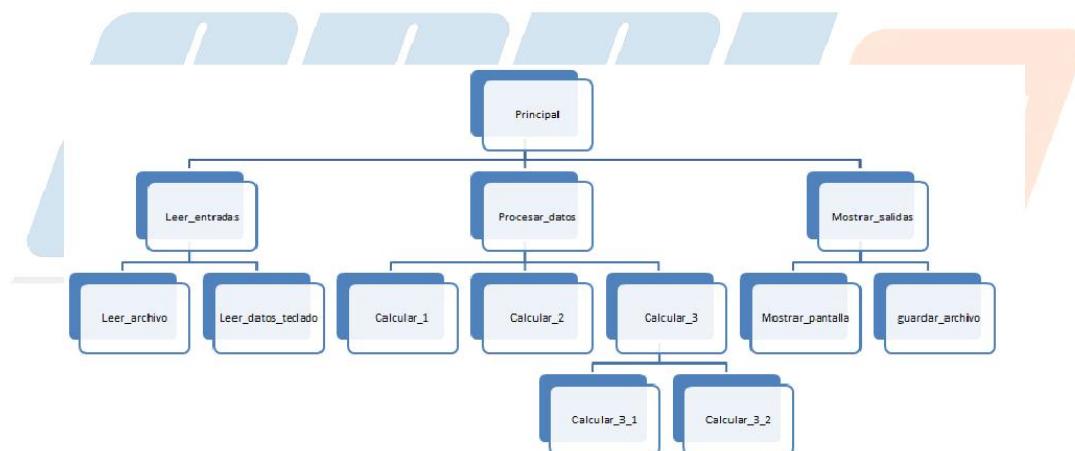
Capítulo 14. MODULARIDAD. PARTE 3

14.1.- Creación de Sub-módulos

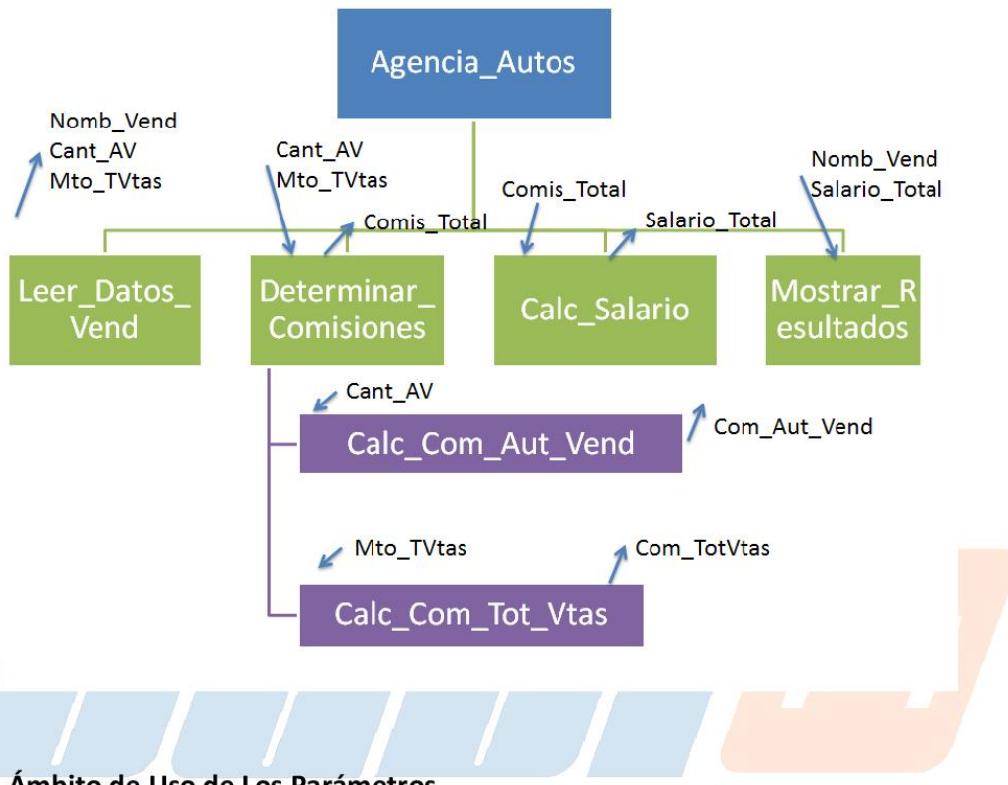
Cuando los problemas son más grandes o más complejos, será necesaria la sub-división de los módulos en varios sub-módulos. Esta división puede hacerse tantas veces como sea necesario hasta lograr que cada sub módulo sea más sencillo de resolver.

En esos casos no siempre la invocación a los módulos se hace desde el cuerpo principal del algoritmo.

En la siguiente imagen se muestra un ejemplo de un algoritmo cuya división se ha hecho más granular dada su complejidad:



Veamos la estructura jerárquica del caso de la Agencia de Autos creando sub-modulos del módulo Determinar_Comisiones, dentro del cual gracias la división del trabajo en 2 sub-modulos se hace más detalladas las determinaciones de las comisiones parciales que determinan la comisión total del vendedor.



14.2.- Ámbito de Uso de Los Parámetros

Como se ha visto, no todos los parámetros se utilizan en el cuerpo principal, ya que algunos de ellos pueden tener un ámbito de uso que se limita los módulos padre de los cuales derivan sus sub-módulos.

Capítulo 15. MÓDULOS COMO FUNCIONES

15.1.- Concepto

Ya se ha visto que los lenguajes de programación poseen módulos predefinidos para realizar operaciones típicas de los algoritmos. Es decir, poseen funciones.

Básicamente una función es un módulo que retorna un solo valor. En el nivel anterior se vieron las funciones, tales como: longitud, mayúsculas, minúsculas y azar, entre otras, que son funciones que vienen con el lenguaje. Pero en ocasiones, se necesitan crear funciones particulares.

15.2.- Declaración

La declaración de una función es diferente a la de un módulo tradicional. Cada lenguaje de programación tiene sus particularidades. En el caso de PseInt, la función se declara igual que un subprocesso, pero colocando el nombre de la variable de retorno que se calculará o determinará, el operador de asignación y luego el nombre de la función.

```
funcion var_retorno<- nombre_funcion(parametro1, parametro2)
    //cuerpo de la funcion
FinFuncion
```

En el siguiente ejemplo se puede observar la definición de una función, cuyo objetivo es calcular y devolver el doble de un número pasado a la función como un parámetro.

```
funcion res<- calc_doble(num)
    res=num*2
FinFuncion
```

En el siguiente ejemplo se puede ver la diferencia de un módulo y de un módulo como función, en cuanto a parámetros, desarrollo y uso. En el ejemplo se puede notar cómo en el módulo tradicional, el parámetro "notatotal" debe pasarse por referencia, en cambio, en la función, ese parámetro no existe debido a que ese es el valor que calculará la función:

```
funcion prom <- calc_promedio(examen1, examen2, examen3)
    prom=(examen1+examen2+examen3)/3
FinFuncion

subproceso calc_promedio(prom por referencia,examen1, examen2, examen3)
    prom=(examen1+examen2+examen3)/3
FinFuncion
```

15.3.- Llamado de Funciones

El llamado de las funciones es particularmente diferente al de los módulos tradicionales, debido a que ellas retornan un valor, este valor retornado puede usarse para almacenarse en una variable, para mostrarse en pantalla o para ser comparado, entre otras opciones.

ACADEMIA DE SOFTWARE

```
Proceso sin_titulo
    //mostrar en pantalla el valor que devuelve la funcion
    escribir nombre_funcion(parametro)

    //asignar a una variable el resultado de llamar a la funcion
    variable = nombre_funcion(parametro)

    //llamado desde una condición
    si (nombre_funcion(parametro)>0)
    |
    FinSi
FinProceso
```

En el siguiente ejemplo se puede observar algunos de los usos de las funciones, es decir, como se pueden llamar desde el cuerpo principal (main) para asignar el resultado de su llamado a otra variable, comparar (dentro de una condición) o mostrar el resultado directamente.

```
10 Proceso ejemplc2
11
12     definir numero, doble como entero
13
14     escribir "ingrese un numero "
15     leer numero
16     escribir "El doble del numero ",numero," es: ",CalcularDoble(numero)
17
18     doble = CalcularDoble(numero)
19
20     escribir "El doble es: ",doble
21
22     Si(CalcularDoble(numero) > numero) Entonces
23
24         Escribir "Calculamos el doble "
25
26     FinSi
27 FinProceso
```



ACADEMIA DE SOFTWARE