PROGRAMACIÓN

La programación es una disciplina cuyo objetivo fundamental es la resolución de problemas mediante la formulación de los pasos necesarios para obtener la solución, codificados en un lenguaje que pueda ser interpretado por una computadora.-

Podemos decir también que un programa es una secuencia de instrucciones cuya ejecución producen una serie de acciones que cambian o transforman el estado inicial del ambiente, pasan por diversos estados intermedios y finalmente arriban a un estado final, el cual debe ser la solución del problema.-

Al tal efecto consideramos las siguientes definiciones:

- **Acción:** Una acción es un evento o acontecimiento que ocurre provocado por un determinado actor, que tiene lugar durante un período de tiempo finito y que produce un resultado bien definido y previsto.-

- **Variables de estado:** Sus valores describen en todo momento el estado de un programa.-

- **Cambio de estado**: Las acciones cambian los valores de las variables de estado, existiendo por lo tanto un estado anterior a la acción o inicial y un estado posterior o final, definidos ambos por el conjunto de valores anteriores y posteriores a la acción.-

- **Ambiente:** El ambiente está representado en cada caso particular por un conjunto de variables de estado u objetos. En todo momento los valores de las variables de estado definen la imagen o estado del ambiente, Esta imagen del ambiente cambia o se transforma por efecto de las acciones, las cuales normalmente se plantean bajo la forma de instrucciones u órdenes a la computadora.-

RESOLUCIÓN DE PROBLEMAS

En general la resolución de un problema no implica una única estrategia de solución, por lo cual es conveniente adoptar un método para su ordenamiento de forma tal de asegurarnos efectuar un profundo y detallado análisis del problema considerado, para luego sintetizar las acciones a seguir . A tal fin se proponen los siguientes pasos :

a) Análisis del enunciado del problema.

b) Estrategia de solución

   b1) Método de solución.
   b2) Descripción del ambiente.
   b3) Algoritmo.

c) Síntesis y Codificación.

d) Puesta en marcha y validación de los resultados.

a) Formulación o enunciado del problema:

Consiste en preparar una completa y exhaustiva descripción del problema, incluyendo los datos y los resultados deseados, de modo de disponer todo lo necesario para la implementación del método de solución.-

b) Estrategia de solución:

La estrategia de solución consiste en el conjunto de pasos necesarios a implementar para poder pasar de la descripción del problema a la obtención de un algoritmo computacional o solución software.-

Como se expresara precedentemente los pasos a considerar son los siguientes:

b1) Método de solución

Consiste en la representación del problema mediante una formulación metodológica y precisa y la implementación de un método de solución que sin lugar a dudas no es único.-

b2) Descripción del ambiente:

Consiste en la formulación del mismo escribiendo los objetos o variables que participan en esta solución. El ambiente representa el estado del sistema y en todo momento queda descripto o definido por estado o valor de sus objetos o variables.-

En la solución de un problema el ambiente evolucionará de un estado inicial a un estado final que **es la solución buscada**, a través de una serie de transformaciones.-

b3) Algoritmo:

Un algoritmo es un conjunto de acciones, que producen sobre él ambiente tales transformaciones que conducen a la solución del problema.-

Obviamente se podrán esbozar distintos tipos de algoritmos sobre un mismo ambiente para implementar el mismo método de solución planteado, teniendo todos ellos dos estados en común: el inicial y el final.-

También debemos tener presente que la descripción o formulación del ambiente tiene una fuerte incidencia sobre el algoritmo de solución ya que define los datos o estructuras de datos a utilizar.-

c) Codificación:

Consiste en la transcripción del algoritmo descripto mediante algún tipo de esquema gráfico, como por ejemplo los diagramas estructurados de Nassi, o escrito en pseudocódigo, al código reconocido por la máquina.-

d) Puesta en marcha y validación de resultados:

Consiste en eliminar todos aquellos errores sintácticos de modo de obtener una ejecución sin problemas y luego mediante un conjunto de datos validar los resultados obtenidos eliminando los errores de lógica o errores metodológicos.-

LENGUAJES DE PROGRAMACIÓN:

Un lenguaje de programación debe ser un medio para la comunicación entre el hombre y la máquina. Existen distintos tipos de lenguajes, dentro de los cuales podemos mencionar los siguientes:

- Lenguaje de máquina.
- Lenguaje ensamblador.
- Lenguaje de alto nivel.

A. Lenguaje de máquina y ensamblador:

Una computadora digital almacena toda la información en memoria en forma binaria (ceros y unos) de modo de tanto las instrucciones como los datos se hallan codificados.

Al estado de un programa codificado mediante ceros y unos, se lo denomina lenguaje de máquina o binario. El ordenador solamente entiende lenguaje de máquina, pero aunque teóricamente es posible para el programador escribir programas directamente en éste lenguaje, lo cierto es que será terriblemente laborioso y por lo tanto poco práctico.

Aparece entonces el concepto de máquina **virtual**. Si a una máquina dada la dotamos de un software especial, de modo que vista desde el exterior se comporte como si admitiera otro lenguaje de programación diferente de su propio lenguaje binario, luego estamos en presencia de una máquina virtual que trabaja con un nuevo lenguaje.-

El más sencillo de éstos lenguajes (el más cercano al de máquina), resulta ser el lenguaje ensamblador (assembler). El assembler es, en realidad un lenguaje que incorpora las mismas instrucciones que dispone el ordenador (es decir, el lenguaje de máquina), pero permite una escritura en forma simbólica.

Se deduce por lo tanto, que el idioma assembler es particular de cada procesador. Sin embargo, se comprende fácilmente que éste lenguaje simbólico es mucho más fácil de utilizar por parte del programador.

A los lenguajes assembler se los denomina de bajo nivel porque una instrucción de éste, se convierte en una instrucción de lenguaje de máquina tal como fue escrita. Es decir a nivel de potencia de instrucción, el assembler no aporta nada superior al lenguaje de máquina.

LENGUAJE DE ALTO NIVEL:

Es obvio pensar que una máquina programable en assembler es más cómoda de utilizar que otra que debiera programarse en idioma de máquina, pero en realidad se trata de las mismas instrucciones básicas (sólo que escritas en lenguaje simbólico), y por lo tanto escribir un programa resulta más cómodo, pero continua siendo una tarea muy tediosa.-

Surge ahora la necesidad de los lenguajes de alto nivel, es decir, aquellos en que por su potencia, una instrucción del programa fuente se convierte en general, en un gran número de instrucciones binarias o de máquina. Así cuando en un lenguaje de alto nivel escribimos la instrucción:

A = B + C (i, j) + LOG (x)

su ejecución puede suponer una serie de instrucciones, sin embargo esto resulta transparente para el usuario.-

A. Compilación

El pasaje del programa fuente creado por el usuario en lenguaje de alto nivel al lenguaje de máquina, se realiza mediante el uso de un programa traductor provisto por el fabricante que se denomina compilador.-

Es muy habitual que el módulo generado por este compilador sea expresado en lenguaje assembler y por lo tanto precise de un paso posterior de ensamblado.-

B. Librerías:

Para simplificar la compilación, los compiladores no generan código auto suficiente, sino que con frecuencia deben hacer llamados a rutinas que realizan una función determinada (por ejemplo leer un registro desde un dispositivo de entrada, imprimir una línea sobre un dispositivo de salida, efectuar un cálculo, etc...).-

Éstas rutinas auxiliares deben estar disponibles externamente agrupadas en librerías, para que luego a su debido tiempo sean citadas e incorporadas al resto del programa mediante el uso de un editor de ensamblado (link).-

C. Intérpretes:

Es otra alternativa para implementar un lenguaje de alto nivel en una máquina. En este caso lo que facilita el fabricante es un programa intérprete de aquel lenguaje que coexistirá con el programa del usuario en la memoria del ordenador. De este modo, nuestro programa no se ejecutará directamente traducido a idioma de máquina, sino a través de la interpretación que producirá al ejecutarse el programa intérprete en una especie de traducción simultánea.-

PROGRAMACIÓN ESTRUCTURADA - TÉCNICAS

La experiencia ha demostrado que un programa que presente dificultades para ser modificado está condenado a la muerte informática. Debemos procurar, por lo tanto, que los programas sean a la vez flexibles y transportables: flexibles para que se adapten con facilidad a cualquier cambio; transportables de forma que cualquier nuevo proceso pueda utilizar sus subprogramas (procedimientos y funciones) sin introducir grandes cambios.-

Conviene para ello, emplear técnicas de programación que faciliten el desarrollo de software fácilmente modificable.

De esta forma el programador que utilice el software desarrollado anteriormente no tendrá que efectuar dos tareas muy tediosas:

- Escribir partes del programa ya escritas.
- Probar subprogramas ya probados.

PROGRAMACIÓN MODULAR (Técnica TOP - DOWN)

Todas estas consideraciones se acercan a la idea de programación modular: cada problema debe descomponerse en una serie de problemas más pequeños hasta llegar a un nivel en que cada uno de ellos no pueda reducirse más. En ese momento se ha llegado al nivel más bajo del análisis.

Es entonces cuando realmente se puede resolver el problema planteado al principio. Cada uno de estos problemas de orden superior puede usar, para su resolución, problemas mínimos, comunes a varios niveles.

Una vez demostrada la necesidad de descomponer un problema general en **problemas mínimos**, resulta obvio que estos no son sino los módulos de que consta el problema. Se está, de esa forma, haciendo a la vez programación modular y programación estructurada: el software obtenido es modular, mientras que las técnicas empleadas para desarrollarlo son estructurales.-

CARACTERÍSTICAS DE UN PROGRAMA ESTRUCTURADO:

El desarrollo de programas modulares requiere un soporte software adecuado: el grado de modularidad obtenido depende del intérprete o del compilador empleado. En este sentido, resulta muy útil contar con instrucciones flexibles de ejecución de subprogramas o módulos.

Para conseguir que estos programas sean transportables es necesario programar en base a módulos de pequeño tamaño, cada uno de los cuales deben contar con toda la documentación posible sobre su funcionamiento. Es preciso que con un simple vistazo del listado de un módulo, cualquier programador comprenda su funcionamiento. Esto facilita además cualquier modificación posible.-

Por otra parte, los algoritmos de un programa estructurado deben ser muy sencillos. Es preferible utilizar varias instrucciones separadas y visibles que una, con muchos niveles de paréntesis, operaciones complicadas, etc... Una sentencia con cinco o seis instrucciones de tratamiento de cadenas, o con varias funciones definidas por el usuario, puede provocar el desconcierto en cualquier programador que intente comprender su funcionamiento.-

Otro factor muy importante que determina la legibilidad de un módulo es la linealidad de la secuencia de sus instrucciones: en éste sentido no resulta aconsejable el uso de sentencias **GO TO**, pues cada vez que un programador se encuentra con una de ellas tiene que reconstruir mentalmente el organigrama del programa. Si los módulos de un programa estructurado están bien construidos, cada uno de ellos ejecutará una sola tarea, y no efectuará ningún tipo de saltos a puntos alejados del programa.-

TIPOS DE ESTRUCTURAS A UTILIZAR:

El teorema de la programación estructurada afirma que puede construirse cualquier programa mediante tres tipos de básicos de estructuras. Estos son:

- **Secuenciales.**
- **Condicionales.**
- **Repetitivas.**

Las instrucciones **secuenciales** son las más comunes en un programa estructurado. Representan una operación o acción ejecutada dentro del programa.

La **bifurcación o transferencia de control** (originada por una instrucción condicional) es la operación por la cual el ordenador selecciona la acción a ejecutar dentro de un conjunto de posibilidades. Esta elección está determinada por el valor que tomen determinadas variables, definidas anteriormente por medio de sentencias lineales.

Por su parte, cuando se repiten varias instrucciones hasta que una variable alcance un determinado valor, se está en presencia de una **estructura repetitiva.**

SUBPROGRAMAS: PROCEDIMIENTOS Y FUNCIONES.

Aunque con estos tres tipos de sentencias se puede construir cualquier clase de programa estructurado, no se garantiza su legibilidad. Para asegurarla se crean secuencias lineales independientes. Cuando un bloque de instrucciones se maneja como si fuera **una única instrucción**, se ha creado un procedimiento. Este, a su vez, puede constar de otros tantos bloques independientes.-

Por otra parte, cada bloque de instrucciones utiliza variables que no deben ser siempre las mismas. Las variables de trabajo de un bloque son transferidas desde el bloque superior a través de los **argumentos** de entrada al procedimiento o función. Los resultados obtenidos pueden, a su vez, emplearse por otros módulos.-

Las variables implicadas en un procedimiento o función pueden ser de uso exclusivo de este bloque, o compartirse con el módulo que los llama. Para asegurarse la flexibilidad de los módulos, cada una de las partes de que se compone de ser autónoma y contener, por tanto sus propias variables independientes. Las modificaciones sucesivas no presentarán de esta forma problemas, ya que cada variable se usa sólo en un módulo concreto.-

VARIABLES Y CONSTANTES - TIPOS DE DATOS
CONSTANTES Y VARIABLES

Una constante es una cantidad que permanece con un valor fijo durante toda la ejecución del programa. Las constantes según el tipo de dato que representen las podemos clasificar también en enteras, reales, lógica, carácter, etc...
Las constantes al igual que las variables pueden adoptar un nombre simbólico o identificador que las represente.

Una variable adopta siempre un nombre simbólico o identificador el cual representa una posición de memoria, cuyo contenido puede cambiar durante la ejecución del programa. Las reglas que rigen la composición de los nombres de variables, o de constantes, pueden cambiar ligeramente, de un lenguaje de programación a otro.

En general se exige que los nombres comiencen con una letra, después de este primer carácter, los siguientes son optativos, se pueden seleccionar de un conjunto que contiene letras, dígitos y algunos caracteres especiales.

Ejemplos de nombres:

A , B31 , Q34X, Z100, SUM_POT, etc...

Los nombres elegidos deben decir algo sobre las cantidades que representan. El programador cuando escribe un programa, es el encargado de seleccionar los mismos, de acuerdo a su criterio y gusto personal. Sin embargo, nombres bien elegidos facilitarán la lectura y comprensión de los algoritmos desarrollados, y simplificarán notablemente su mantenimiento, o sea todos los cambios y modificaciones que debamos introducir en el futuro.-

La declaración de tipo de las variables es usada por el compilador para determinar cuanto espacio de memoria debe reservar para cada tipo de dato y para establecer algunos límites a los valores que pueden asignárseles.-

DATOS - TIPOS

Dato es una expresión general que describe los **objetos** con los cuales opera una computadora. Existen diferencias básicas entre los lenguajes de programación, según el tipo de datos que usan, las operaciones primitivas sobre estos datos, y el modo en que pueden organizarse para formar estructuras más complejas. Así podemos clasificar los datos en **simples o escalares, estructurados y dinámicos (punteros).-**

El usuario puede además crear sus propios tipos en función de tipos ya establecidos primitivos o no.-

En general los lenguajes de programación poseen un conjunto fijo de tipos de datos llamados **datos primitivos.**

Cada tipo de dato define el conjunto de valores que puede asumir una variable. Los tipos de datos primitivos, también llamados básicos o simples y que manipulan la mayor parte de los ordenadores son:

- Tipo numérico (enteros, reales).
- Tipo lógico.
- Tipo carácter.

TIPO NUMÉRICO

Los datos de tipo numérico los podemos clasificar en enteros y reales. Reales con y sin exponente.-

ENTEROS

Una constante entera es un número entero, escrito sin punto decimal, que puede ser positivo, negativo o cero. El signo positivo es optativo.

Los números enteros que pueden ser manejados por una computadora están siempre comprendidos entre un valor máximo y un mínimo. Los valores mínimos y máximos mencionados podrían ser, por ejemplo, -32768 y +32767, por lo tanto valores superiores o inferiores a estos, no podrán ser representados por este tipo de entero en la máquina.

Ejemplos de constantes enteras: 27, 31, -12, 93, etc...

REALES

Una constante real se compone de una mantisa y eventualmente un exponente (potencia de 10). El número máximo de dígitos significativos, depende en general del ordenador, pero un valor típico suele ser siete.-

Ejemplos de constantes reales sin exponente:

12, 31, 1, 78, -45, 781, -0.213, 0.00945

Ejemplos de constantes reales con exponente:

15×10^3	10 . E03	15000
15.73×10^{-2}	15. 73E - 02	0. 1573
-0.21765×10^3	-0. 21765E+03	217.65

EXPRESIONES Y ASIGNACIÓN

Los lenguajes de programación proporcionan en general los siguientes operadores aritméticos:

Operador	Función
+	Suma
-	Resta
*	Producto
/ ¨	División Potenciación

EXPRESIONES ARITMÉTICAS

Una expresión aritmética se define como una secuencia de una o más variables y/o constantes unidas por cualquier operador aritmético, la cual da un solo valor numérico como resultado.

Este resultado puede ocasionar en algunos casos, un error de desbordamiento (overflow o underflow) cuando el resultado de una operación es demasiado grande o demasiado pequeño para poder almacenarlo en el espacio de memoria reservado. Por lo tanto es tarea del programador estimar los resultados de las operaciones de modo de evitar cualquier error de esta naturaleza.-

Las operaciones dentro de una expresión se ejecutan de acuerdo con el siguiente orden de prioridades o de precedencia:

1	**^**	**Potenciación**
2	*** , /**	**Producto y división**
3	**+ , -**	**Suma y resta**

Cuando en una expresión existen operaciones con igual prioridad, se resuelve de izquierda a derecha. Sin embargo, en muchos casos es conveniente alterar la jerarquía normal de las operaciones, lo cual es posible colocando pares de paréntesis en lugares adecuados de las expresiones, logrando de esta forma que primero se efectúen las operaciones incluidas entre los paréntesis más internos, luego las de los segundos más internos y sí sucesivamente, conservando siempre entre cualquier pareja de paréntesis la jerarquía normal de las operaciones, a menos que entre éstos también existan paréntesis.-

Veamos en que orden se desarrolla las siguientes expresiones:

```
T + R - W + ( S - D ) + 45 - ( C - H )
```

1.- S - D = Z1
2.- C - H = Z2
3.- T + R = Z3
4.- Z3 - W = Z4
5.- Z4 + Z1 = Z5
6.- Z5 + 45 = Z6
7.- Z6 - Z2 = Z7 (Resultado)

```
B + ( ( A + B) * C ) - A ¨ 2
```

1.- A + B = Z1
2.- Z1 * C = Z2
3.- A ^ 2 = Z4
4.- B + Z2 = Z3
5.- Z3 + Z4 = Z5 (Resultado)

En la práctica, si existen dudas en cuanto al orden en que se desarrollará una expresión, es recomendable utilizar paréntesis para forzar la evaluación de la expresión según el orden determinado.-

FUNCIONES INTERNAS

Además de las operaciones básicas como suma, resta, producto, división y potencia, existe otro conjunto de operaciones especiales que e denominan internas, las cuales proporcionan una forma rápida de evaluar muchas funciones matemáticas y llevar a cabo algunas operaciones lógicas. Estas funciones internas son rutinas prescritas que están incluidas como parte integral de los lenguajes.

Se tiene acceso a cada función simplemente haciendo referencia a su nombre, seguido de un argumento cuyo tipo y rango de variabilidad dependerá de cada función en particular.-

Se incluye a continuación de tales funciones internas:

Nombre de la función	Significado
ABS (x)	Valor absoluto
RC (x)	Raíz cuadrada
LOG (x)	Logaritmo natural
LOG 10 (x)	Logaritmo decimal
EXP (x)	Exponencial base 10
SEN (x)	Seno
COS (x)	Coseno
TAN (x)	Tangente
TRUNC (x)	Truncamiento
REDON (x)	Redondeo
RESTO (X/Y)	Resto

* * En las funciones trigonométricas el argumento debe expresarse en radianes. * *

DATOS DE TIPO LÓGICO

Los valores de tipo lógico son VERDADERO (V) y FALSO (F).

En programación, la mayoría de las operaciones, suelen depender de los resultados de la evaluación de una condición, la que puede resultar de la comparación de dos valores del mismo tipo, como por ejemplo, dos valores numéricos o dos valores de tipo carácter.-

Para las comparaciones antes señaladas, se utilizan los operadores relacionases, los que se consignan en la siguiente tabla:

Op. relacional	Significado
=	Igual
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual
><	Distinto

Si A y B son dos variables numéricas cuyos valores son 3 y 15 respectivamente, la tabla siguiente ejemplifica el resultado de algunas expresiones condicionales:

Exp. relacional	Valor
A >= B	Falso
A = B	Falso
B > A	Verdadero

También es posible la comparación entre cadenas de caracteres. Para estas, la comparación se efectúa carácter a carácter, de izquierda a derecha en función de una tabla (código ASCII), en la cual cada carácter le corresponde un número entero positivo según su ubicación dentro de ella. Así por ejemplo SUMAS es mayor que la palabra SUMAR, ya que la letra R se ubica antes en la tabla ASCII que la letra S.-

Las expresiones relacionales pueden a su vez combinarse, mediante conectores u operadores lógicos, para formar condiciones compuestas o expresiones lógicas.-

Operador lógico	**Significado**
(Y) (and)	Conjunción
(O) (or)	Unión lógica
(NO) (not)	Negación

La evaluación de las expresiones lógicas sigue las leyes del álgebra de Boole. Para simplificar y ordenar la evaluación de estas expresiones lógicas construiremos lo que se denominan **tablas de verdad**.-

X	Y	X and Y
V	V	V
V	F	F
F	V	F
F	F	F

X	Y	X or Y
V	V	V
V	F	V
F	V	V
F	F	F

X	not X
V	F
F	V

El resultado de una expresión relacional o lógica puede ser asignado a una variable de tipo lógico, la cual contendrá entonces verdadero (V) o falso (F).-

Consideremos las siguientes expresiones de asignación sabiendo que A y B contienen los valores 7 y 12 respectivamente:

Exp. Asignación	**Valor asignado**
W ← A > B	W ← (F)
W ← A = B	W ← (F)
W ← A < B	W ← (V)
W ← (F) (*)	W ← (F)

(*) En esta última expresión, asignamos directamente una constante lógica, en lugar del resultado de una expresión lógica como en las tres anteriores.-

TIPO CARÁCTER

El tipo carácter está constituido por un conjunto de caracteres que el procesador reconoce. Los caracteres alfanuméricos y los símbolos especiales son aquellos que no tienen valor numérico, sino que representan para la computadora simplemente un símbolo y nada más.-

También existe un grupo de caracteres denominados especiales. En general, todas las computadoras reconocen un conjunto de caracteres más o menos standard, que es el siguiente:

Conjunto de carac. Reconocidos	**Códigos ASCII**
Letras mayúsculas (A ... Z)	(65 ..... 90)
Letras minúsculas (a ....z)	(97 ..... 122)
Dígitos numéricos (0 ... 9)	(48 ..... 57)
Carac. Espec. (!, #, @ , $, ^, * ...)	(33, 35, 64, 36, 94, 15)

Un elemento del conjunto descrito, delimitado mediante apóstrofes representa una constante carácter, como por ejemplo las siguientes:

"@ ", " A ", " $ ", " 1 ", "/ ", " $ "

Una constante de tipo carácter se escribe entre apóstrofes para evitar confundirlo con los nombres de las variantes, operadores, etc...

Cada carácter reconocido por la computadora, tiene un valor decimal asociado en un código denominado ASCII (código standard americano para intercambio de información).-

Así cuando comparamos dos valores carácter, lo que se comparan son sus correspondientes códigos ASCII, por esta razón podemos decir:

" a " es menor que " b "	(97 < 98)
" A " es menor que " a "	(65 < 97)
" 4 " es mayor que " & "	(52 > 38)

El tratamiento de cadenas o tiras formadas por estos caracteres, las operaciones posibles, y las operaciones de asignación serán tratados posteriormente en la Unidad 9.-

ESTRUCTURAS SECUENCIALES

Las estructuras secuenciales están compuestas por acciones que se ejecutan en secuencia, es decir unas detrás de otras en el orden natural en que se encuentran escritas en el texto del programa. Las instrucciones de entrada / salida y las de asignación son ejemplos de éstas.-

En una computadora la entrada y salida representa la interfase final entre el usuario y un programa que ejecuta. Las instrucciones de entrada de un programa hacen que la computadora lea o ingrese datos durante la ejecución del mismo, lo que permite al programador escribir programas generales que pueden ser utilizados repetidamente para diferentes conjuntos de datos, permitiendo así, una participación activa del programador en el tiempo de ejecución.-

Por otra parte, las instrucciones de salida permiten a la computadora sacar o escribir la información contenida en su memoria.-

Por ejemplo, la instrucción **LEER** o **INGRESAR base**, toma un valor de un medio periférico, por ejemplo el teclado y lo asigna a una posición de memoria rotulada o identificada mediante el nombre **base**, destruyendo su contenido anterior.-

Por su lado, la instrucción **ESCRIBIR** o **MOSTRAR suma,** extrae o escribe el valor contenido en la posición **suma** sobre un medio periférico de salida, como podría ser el monitor de video.-

Debemos destacar siempre al respecto, que en la ejecución de estas instrucciones de entrada y salida, el flujo de información se establece entre un medio periférico y la memoria central de la máquina y viceversa.-

Con ayuda de las instrucciones de entrada/salida y la de asignación ya vistas, podemos ahora preparar nuestro primer programa.-

```
PROGRAMA  :      SUMA

VARIABLES ( A ,  B ,  C ):  enteras

COMIENZO

                INGRESAR "A"
                INGRESAR "b"
                C ← A + B
                MOSTRAR C

FIN
```

<------Instrucciones Declarativas

En el pequeño programa escrito antes, debemos destacar en primer lugar la declaración de las variables "A" y "B", mediante la cual informamos al programa cuales son las variables que vamos a utilizar y cual es su tipo. Luego, entre las palabras COMIENZO y FIN encontramos las instrucciones ejecutables las cuales componen lo que se denomina cuerpo principal del programa.-

Analicemos ahora cada una de las instrucciones ejecutables por separado:

INGRESAR "A" : Ingresa o lee la variable A

INGRESAR "B" : Ingresa o lee la variable B

C ← A + B : Calcula la expresión A + B y asigna el valor a C

MOSTRAR "C" : Saca o escribe la variable C

De modo que, si a la variable A le damos el valor 3, y a la variable B el valor 5, el programa nos proporcionará el valor 8 que corresponde a la variable C.

Finalmente, y a fin de facilitar la utilización del programa, documentaremos las instrucciones de entrada y salida mediante el agregado de leyendas que informarán durante la ejecución al usuario, cuales variables se leen y cuales de escriben.-

```
PROGRAMA   :   SUMA
VARIABLES  :   A, B, C : enteras
```

<------Instrucciones Declarativas

```
COMIENZO

   LEER   * Ingrese A  :  *   ,  A
   LEER   * Ingrese B  :  *   ,  B

   C  =  A  +  B

   ESCRIBIR  * La suma es  :  *   ,  C

FIN
```

<------Instrucciones Ejecutables

PROGRAMACIÓN MODULAR - SUBPROGRAMAS

También además de las instrucciones de entrada y salida y las de asignación, podemos considerar como sentencias lineales o que se ejecutan en secuencia, aquellas que se agrupan formando un módulo separado e independiente ubicado en un nivel subordinado, al cual denominamos **subprograma**. Así, cuando un grupo de instrucciones se maneja como si fuera una única instrucción, se ha creado un subprograma.-

Luego podemos definir un subprograma como un conjunto de sentencias de un programa que realiza una determinada tarea y que puede ser ejecutada desde uno o más puntos del programa principal con la simple mención de su nombre.-

Al efectuarse el llamado, el control del programa se transfiere a este módulo independiente el cual después de realizar la tarea encomendada retorna nuevamente el control al programa llamador o principal continuando éste la ejecución normal del resto de sus instrucciones.-

Este módulo independiente o subprograma posee todas las características propias de un programa, salvo que no se puede ejecutar por sí mismo, sino mediante el llamado de otro módulo que hace las veces de programa principal.-

Con el auxilio de éste último concepto podemos encarar la resolución de complicados problemas mediante su división en problemas de menor complejidad o subproblemas.-

Estos subproblemas pueden dividirse en a su vez en otros subproblemas más pequeños, hasta llegar a subproblemas fáciles de solución.-

Este proceso de subdividir en tareas de menor envergadura a una tarea compleja, es el método denominado de **Refinamiento sucesivo** o también conocido como diseño descendente **TOP-DOWN.-**

El refinamiento sucesivo es por lo tanto, el proceso mediante el cual podemos desglosar un problema en problemas más pequeños que pueden tratarse y desarrollarse independientemente entre sí.-

Las soluciones derivadas de un diseño descendente de problemas complejos pueden implementarse fácilmente en cualquiera de los lenguajes de alto nivel hoy disponibles, como pueden ser: Pascal, Fortran, C, Qbasic, Clipper, etc...

La división antes mencionada de un programa en módulos independientes o subprogramas, exige que un módulo de ellos actué como coordinador el cual debe controlar y relacionar a todos los demás, siendo este el denominado programa principal o **MAIN.-**

Siguiendo Ésta metodología de programación modular se logra las siguientes ventajas:

- Los programas modulados son más fáciles de escribir y depurar, así como de mantener y modificar.-

- Los subprogramas permiten dividir un programa en unidades de programa más pequeñas (subprogramas) de forma tal que cada una de ellas pueda ser desarrollada de modo independiente e incluso por programadores diferentes.-

- Los subprogramas usados en un programa pueden ser reutlizados en otros programas.-

- Los módulos permiten simplificar notablemente el código, al reemplazar el conjunto de instrucciones que lo componen por una simple instrucción de llamado en cada oportunidad en que se requiera la tarea que ellos desarrollan.-

ESTRUCTURAS DE CONTROL

La gran potencialidad de un procesador reside en su capacidad de tomar decisiones y de determinar que acción debe efectuar en el momento de la ejecución, sobre la base del estado de ciertas variables residentes en su memoria.-

Llamaremos flujo de control de un algoritmo, al orden en el cual se deben ejecutar las acciones del mismo. Dentro del flujo de control, podemos considerar el **flujo lineal**, es decir, donde la ejecución de las acciones se efectúa en secuencia, desde la primera a la ultima.-

Ahora corresponde analizar las estructuras mediante las cuales es posible alterar este orden natural de ejecución de las acciones, las cuales se denominan estructuras condicionales.-

ESTRUCTURA CONDICIONAL SIMPLE

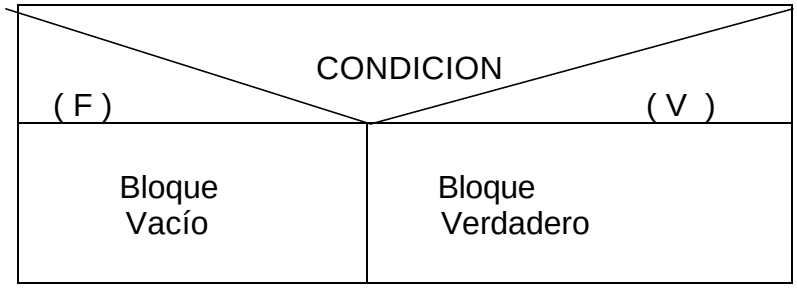
Esta condición permitirá cuando la condición se verdadera, que el flujo de control acceda a un grupo de acciones, denominado **bloque verdadero**, mientras que en el caso de que la condición sea falsa deberá omitirlas, dicho de otro modo, un grupo de acciones de un programa se ejecutara en un caso y en otro no.-

SI (condición)

entonces

.....
.....
.....

FIN - SI



Por lo tanto, si la condición es verdadera corresponde ejecutar un grupo adicional de acciones, que en caso de ser falsa se omitirá y el programa continuara con la ejecución de la siguiente instrucción después del fin de la estructura SI.-

ESTRUCTURA CONDICIONAL COMPUESTA O DOBLE

Esta estructura también denominada alternativa, a diferencia de la condicional simple, ejecutará un grupo de instrucciones si la condición es verdadera y otro si la condición es falsa, correspondiendo a las opciones entonces y sino respectivamente.-

SI (condición)

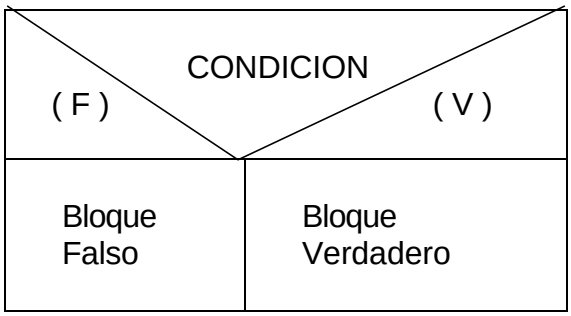
entonces

…… …… bloque verdadero

sino

…… …… bloque falso

FIN - SI



ESTRUCTURA MULTIPLE SELECTIVA

Esta estructura permitirá seleccionar la ejecución de un grupo de acciones de un conjunto de varias alternativas. Esto se lograra mediante la utilización de una expresión numerable, cuyo resultado hace las veces de una **llave** que habilita la ejecución de un determinado grupo o bloque de instrucciones.-

```
SEGÚN  DÍA  HACER
     1:  ESCRIBIR   " Domingo"
     2:  ESCRIBIR   " Lunes"
     3:  ESCRIBIR   " Martes"
     4:  ESCRIBIR   " Miércoles"
     5:  ESCRIBIR   " Jueves"
     6:  ESCRIBIR   " Viernes"
     7:  ESCRIBIR   " Sábado"
FIN  - SEGÚN
```

D Í A						
1	**2**	**3**	**4**	**5**	**6**	**7**
DOM	**LUN**	**MAR**	**MIE**	**JUE**	**VIE**	**SAB**

ESTRUCTURAS DE REPETICIÓN

Analizaremos ahora las diferentes maneras de preparar un ciclo, también denominado corrientemente lazo, bucle o iteración, con el cual podremos repetir un bloque de una o más acciones. Este bloque de acciones constituye el **rango** del ciclo que podrá repetirse un número determinado de veces según una expresión condicional que podremos ubicar antes o después del mismo.-

Nos encontramos entonces con dos tipos de estructuras de repetición:

ESTRUCTURA MIENTRAS FIN - MIENTRAS

Su forma general es:

```
mientras ( condición ) hacer
                    acción 1
                    acción 2
                    .....          } rango
                    acción n
fin - mientras
```

Recordemos que la condición es una expresión relacional o lógica cuya evaluación puede arrojar un resultado **verdadero o falso**, del cual dependerá en este caso la repetición del rango o bloque de acciones. Si al evaluarse por primera vez la expresión condicional resulta **falsa,** el ciclo no será ejecutado y el flujo de control será transferido a la instrucción al **fin - mientras.-**

Si en caso contrario, la condición resultara **cierta,** entonces se ejecutará el rango y el flujo retornará nuevamente a la sentencia de control del ciclo, donde se evaluará una vez más, repitiéndose el proceso.-

De aquí se desprende que en éste tipo de estructura el rango pude no ejecutarse nunca, ya que la sentencia de control del ciclo se encuentra antes que este.-

Consideremos el siguiente ejemplo:

```
Leer n                                  (1)
k  = 0                                  (2)
 mientras ( k < n ) hacer               (3)
                k = k +1                (4)
                 escribir ' estoy girando '   (5)
 fin - mientras                         (6)
```

En el ejemplo anterior, se conoce de antemano el número de giros que pretendemos efectúe el bucle, ya que a "n" le asignamos un valor mediante la instrucción de lectura (1).-

El número de repeticiones está controlado por la condición (k < n) de la instrucción (3), la cual permitirá un nuevo giro siempre que la variable " k " resulte menor que " n ". De esta forma en cada giro imprimirá la leyenda **Estoy girando.-**

La variable " k " denominada **índice o contador** del bucle, es inicializada en la instrucción (2) e **indexada** en la (4). Siendo esta última instrucción parte del rango, se garantiza la terminación del proceso iterativo ya que permitirá que la condición resulte en algún momento falsa e impida un nuevo giro.-

ESTRUCTURA REPETIR HASTA QUE

Su forma general de este nuevo ciclo es:

```
Leer n                                  (1)
 k  = 0                                 (2)
repetir                                 (3)
                k = k +1                (4)
                 escribir ' estoy girando '   (5)
hasta que ( k = n )                     (6)
```

Aquí la última sentencia es la sentencia de control del ciclo, por lo tanto en esta estructura el rango se ejecuta **siempre** por lo menos una vez. Al final de cada iteración, se evalúa nuevamente la condición y si es **verdadera,** la ejecución del ciclo termina y se pasa a ejecutar la sentencia siguiente a la condición **hasta que**.

Si por el contrario, su valor es **falso** el ciclo se ejecuta de nuevo.-
Podemos por lo tanto observar, que la ejecución de este ciclo se diferencia del ciclo anterior **mientras**, en que se ejecuta por lo menos una vez y finaliza el número de giros cuando la condición es **verdadera** en lugar de falsa como ocurría antes.-

BUCLE CONTROLADO POR CONTADOR

Un bucle controlado por **contador** es un ciclo que se ejecuta un número especificado de veces. Es necesario para el diseño correcto del bucle un mecanismo que controle el número de veces que se ejecuta el mismo. Para ello se utiliza una **variable de control del bucle (índice)** que actúa como un contador.-

Un bucle controlado por contador consta de tres partes, además del **cuerpo o rango** y de la condición de salida.-

(1) * Inicialización de la variable de control.-
(2) * Verificación de la variable de control.-
(3) * Incremento de la variable de control.-

BUCLE CONTROLADO POR VARIABLE CENTINELA

Los bucles o lazos iterativos se utilizan con frecuencia para leer y procesar listas de datos sin conocer el número de ellos por anticipado. Cada vez que una iteración se ejecuta, una nueva lectura de datos se realiza. Para finalizar este proceso se utiliza un valor especial de alguno de los datos leídos, al cual denominamos **centinela.** Su valor solo sirve para indicar el fin del lazo iterativo y **no se procesa como dato válido,** por lo tanto su acción es controlar el bucle.-

El bucle se ejecuta mientras no se lea el valor **centinela**. En el momento que se lee este valor centinela y que se reconoce en la condición inicial, se sale o termina el bucle.-

Mes = 2 febrero, no puede tener día = 31,
condición imposible, se adopta como valor centinela.

```
mientras NO ( mes = 2 [y] día = 31 ) hacer
                    acción 1
                    acción 2
                    .....          } rango
                    acción n
fin - mientras
```

BUCLE CONTROLADO POR VARIABLE BANDERA

Una **bandera o flag** es una variable que se utiliza para conservar el estado, verdadero o falso de una condición. Se denomina bandera o interruptor debido a que puede ser asociado a un interruptor o a una bandera que se encuentra arriba o abajo.-

Algunos idiomas no disponen de variables lógicas, por lo tanto, se deben simular mediante variables enteras a las que se asignan los valores verdadero (1 o -1) o falso (0).-

Con frecuencia se desea repetir una serie de acciones mientras que una determinada condición representada por la variable **bandera o flag** sea verdadera. Entonces el bucle se controla por el valor verdadero de la condición.-

```
flag = [V]

Mientras flag hacer

        SI  ( condición )
                Flag = [F]
         SINO
                acción 1
                .....
                acción n
        FIN - SI
fin - mientras
```

BUCLES INFINITOS

Un bucle cuyo bloque de sentencias se ejecuta indefinidamente se denomina **bucle infinito o sin fin**. La condición que termina un bucle en una estructura **MIENTRAS O REPETIR** o la variable que controla el contador en lazo **VARIAR**, constituyen la condición terminal o de fin de bucle. Normalmente esta variable debe ser modificada por alguna de las sentencias del cuerpo del lazo iterativo, de lo contrario se convertiría en un proceso sin fin.-

Para poner término al proceso iterativo que se desarrolla en un lazo infinito, se dispone de la siguiente sentencia:

* SI (cond) ENTONCES SALIR DEL LAZO

las que producen una transferencia del control al exterior del lazo iterativo exactamente a la línea siguiente a la de fin del lazo, con lo que se logra interrumpir el lazo infinito.-

ITERAR

La instrucción **ITERAR** constituye un lazo infinito que cuenta para finalizar con una transferencia de control al exterior del mismo exactamente a la línea siguiente al FIN - ITERAR. Sin embargo, en algunos idiomas no se cuenta con dicha instrucción de transferencia. Por lo tanto, para finalizar el lazo nos veremos obligados a rotular una línea y a utilizar la instrucción de transferencia de control GO TO.-

```
ITERAR
        .....
        .....
SI (Condición ) SALIR
        .....
        .....
FIN - ITERAR
```

REGLAS DE USO DEL CICLO VARIAR o PARAR

V i = Valor inicial
V f = Valor final
Inc = Incremento

```
VARIAR i = V i HASTA V f PASO Incr
        ..............
        ..............
FIN - VARIAR
```

- La sentencia FIN - VARIAR marca el final del bucle y el punto en que la variable de control del bucle se incrementa después e cada iteración. El bucle finaliza cuando i alcanza el valor V f.-

DATOS DEFINIDOS POR EL USUARIO - CONSIDERACIONES

La resolución de cualquier problema, ya sea que use o no programación, involucra datos. Estos pueden muchas formas diferentes: notas, palabras, números, imágenes, etc... Los datos pueden emplearse con diferentes propósitos, para resumir hechos relacionados con el problema, para proporcionar la solución de un problema o para obtener cierta información intermedia deducida de los hechos y que resulta necesaria para la solución.-

El primer paso en la resolución de un problema es determinar los datos mismos (las entradas) así como los que se obtendrán con la solución (las salidas). El segundo paso es determinar la manera en la que se obtendrán las salidas a partir de las entradas. En general, lo anterior implica la manipulación de datos.

Existen muchas formas de hacerlo, entre las que se incluyen las operaciones aritméticas, el ordenamiento, las operaciones con texto, con gráficas, de presentación en pantalla, de impresión y de referencia. El manejo particular de los datos cambia de un problema a otro y depende de lo que se necesite obtener para generar las salidas a partir de las entradas.-

En todos los problemas, es necesario encontrar la manera de representar los datos relacionados. La representación que se seleccione puede incidir mucho en la organización de los datos y ser de gran ayuda en su manejo.

En general los idiomas proporcionan varios modos de representar los datos, los cuales dan lugar a lo que se conoce con el nombre de **estructuras de datos.**

Por lo tanto, uno de los aspectos más importantes de los idiomas es la posibilidad de definir tipos de datos a medida o sea según las necesidades. Al adaptar las estructuras específicas de un programa se puede aumentar la legibilidad del programa y simplificar su mantenimiento.-

DATOS ESTRUCTURASDOS - ARREGLOS

Los datos pueden organizarse en diferentes formas. El modelo matemático o lógico de una organización particular de datos recibe el nombre de **estructura de datos.** Una estructura de datos es, en esencia, una colección de datos, donde la forma en que están relacionados unos con otros determina el tipo de estructura.-

Las estructuras de datos se clasifican en dos grandes grupos: **estáticos y dinámicos.** Por el momento nos ocuparemos de las estructuras estáticas que son aquellas en que la dimensión ocupada en memoria se declara previamente y no es posible modificar.-

En particular trataremos ahora los **arreglos** que están compuestos por una colección o conjunto de posiciones de memoria que tienen el mismo nombre de variable, es decir, un conjunto de datos numéricos o de cadena que tienen el mismo nombre. Los valores individuales de un arreglo se denominan **elementos.**

Cuando un arreglo es referenciado por un nombre de variable, los elementos de los arreglos son también variables y por lo tanto se los puede utilizar en todos los usos que se destina normalmente a cualquier variable no estructurada.-

TIPOS DE ARREGLOS

Podemos clasificar los arreglos atendiendo a sus dimensiones de las siguiente forma:

- Arreglos unidimensionales (listas o vectores).
- Arreglos bidimensionales (tablas o matrices).
- Arreglos multidimensionales.

Cada valor de un arreglo se denomina elemento del arreglo y para acceder a él se debe utilizar un **subíndice**, normalmente numérico que identifica la posición de un valor dado del arreglo.-

En general, los lenguajes de programación permiten definir arreglos de cualquier tipo: numérico, cadena, lógicos o registro. A su vez los numéricos pueden ser enteros, reales simples o reales de precisión extendida.-

VECTORES - ARREGLOS UNIDIMENSIONALES

Un **arreglo unidimensional o lineal**, también llamado lista o vector, es una lista de un número finito de datos (elementos) del mismo tipo, que se referencian por un nombre común y un número de orden (subíndice) y que son consecutivos normalmente : 1, 2, 3, ... n.-

Las variables que representan los arreglos se denominan **variables con subíndices o variables subcriptas.** A cada elemento de estos arreglos se les puede asignar un valor y se los puede manipular de igual forma que cualquier otra variable. La dimensión de un vector es el número de elementos o posiciones de memoria que son reservados para este arreglo.

Así para el caso de un vector A (5) que consta de cinco elementos, B (4) de cuatro y C (3) de tres, lo podemos representar de la siguiente forma:

VECTOR A

A (1) A (2) A (3) A (4) A (5)

VECTOR B

B (1)
B (2)
B (3)

VECTOR C

C (1) C (2) C (3)

En el primer caso del vector A el subíndice podrá tomar los valores comprendidos entre 1 y 5, mientras que en el segundo y tercer caso, el subíndice podrá variar entre 1 y 4 y 1 y 3 respectivamente.-

MATRICES - ARREGLOS DE DOS DIMENSIONES

Recordemos que un arreglo de una dimensión o vector es, básicamente, una serie de posiciones contiguas de memoria que se identifican por un nombre, dicho de otra forma, una columna o una fila de datos. Cada posición del vector viene dada por un número llamado **subíndice o índice** que lo distingue de los demás.-

Un arreglo de **dos dimensiones**, denominado también **matriz o tabla,** es un arreglo con filas y columnas. Así una matriz o tabla de 3 filas y 5 columnas se representa de la siguiente forma:

Matriz A

12	67	89	78	33
22	02	14	55	87
11	59	31	74	45

Para representar una matriz o tabla bidimensional, se necesita una variable con dos subíndices, el primero de los cuales especifica el número de filas y el segundo el número de columnas. Por consiguiente, el elemento A (2, 1) representa la posición ubicada en la segunda fila y la primera columna de la matriz A.-

Por otra parte, la matriz A formada por 3 filas y 5 columnas, necesitará memoria suficiente para acomodar los 15 elementos que la componen.-

ARREGLOS MULTIDIMENSIONALES

Hasta aquí solo hemos tratado arreglos de dos y tres dimensiones, pero podemos usar también arreglos con más dimensiones, por ejemplo de 3, 4 ... n.

Básicamente se aplican las mismas reglas ya vistas. El orden de almacenamiento dado para los arreglos de dos dimensiones puede generalizarse en aquellos arreglos que poseen más.

Es decir, los elementos de los arreglos se almacenan en posiciones consecutivas de memoria en un orden tal que cada uno de los subíndices recorre todo su rango de valores para cada uno de los valores del rango del subíndice situado inmediatamente a su izquierda.-

Arreglos con más de tres dimensiones, son normalmente poco usados por los programadores puesto que son más difíciles de visualizar y por consiguiente de manipular.-

REGISTROS

Los arreglos ya vistos permiten agrupar bajo un nombre común varios datos que tienen como características ser del mismo tipo. Sin embargo, en muchas aplicaciones, es útil describir grupos de datos de diferentes tipos. En la mayor parte de estas aplicaciones, resulta de gran utilidad describir los datos mediante el uso del tipo de datos registro.

Consideremos por ejemplo, la información referente a una persona. Nombre y apellido, número de documento, edad, sexo, dirección y teléfono. Cada uno de estos datos corresponde a un tipo diferente. Así podríamos pensar en la siguiente estructura:

```
TIPO persona = REGISTRO

    Nombre    :  cadena (25)
    Docum.    :  cadena (10)
    Edad      :  entera
    Sexo      :  lógica
    Dirección :  cadena (40)
    Teléfono  :  entera
FIN - REGISTRO
```

Esta estructura se parece a un arreglo en el sentido de que ésta consiste en una secuencia de datos. Sin embargo, es diferente de un arreglo porque los datos tienen tipos distintos. Por lo tanto podríamos dar la siguiente definición:

El acceso a un componente de la estructura de datos de logra añadiendo al nombre de ésta, el del campo en cuestión separado mediante un punto. Las variables que identifican los componentes de un registro pueden manejarse de la misma manera que cualquier variable.

SUBPROGRAMAS

En programación es muy frecuente que un determinado procedimiento de cálculo, definido por un grupo de sentencias, tenga que repetirse varias veces a lo largo del programa, lo cual implica que tenga que escribirse tantos grupos de aquellas sentencias como veces aparezca dicho proceso. Tal vez la herramienta más potente con que se cuenta para facilitar, reducir y dividir el trabajo en programación, es escribir aquellos grupos de sentencias **una sola y única vez** bajo la forma de un **subprograma**.-

El criterio clásico mencionado de que si un trozo de programa se usa muchas veces, entonces está justificado el aislarlo formando un subprograma, es realmente cierto. Pero cada vez se impone más la idea de que un programa constituye una tarea muy compleja de resolver y por lo tanto, es conveniente que esté bien estructurado y dividido en subtareas. Con éste último enfoque, si un grupo de sentencias realiza una tarea bien definida, entonces se justifica aislarlo formando un subprograma, aunque solo se llame o se use una sola vez.

Este modo de proceder, no solo hará más sencilla la tarea de desarrollo del programa, sino que además facilitará también enormemente su mantenimiento, o sea tanto las modificaciones como las ampliaciones futuras.

Visto así, un subprograma es una parte autónoma del programa que realiza una tarea o función definida, la cual puede ser llamada o activada, desde otras partes del programa cada vez que se necesite desarrollar dicha función.

Los subprogramas los podemos clasificar en **funciones y procedimientos** y sus características y funciones las describiremos a continuación.-

FUNCIONES

Una función es un grupo de sentencias de un programa que forman un bloque separado e independiente. Este bloque de sentencias, como ya comentamos, constituye una subtarea que realiza un conjunto determinado de operaciones sobre un grupo de argumentos dados y devuelve un único resultado.

Cada vez que se llama a la función, se transfiere el control al bloque de sentencias definido por esa función, después de que las sentencias han sido ejecutadas, el control retorna a la instrucción desde donde fue invocada o llamada la función.

El bloque de sentencias que componen una función constituye lo que se llama definición de la misma. El control se transfiere desde el exterior mediante una invocación de la función, escrita de la siguiente forma:

```
Nombre - Función ( arg1,  arg2, ... )
```

(1) llamado a la función

Los argumentos que aparecen en el llamado a la función, se escriben entre paréntesis a continuación del nombre y constituyen la información de entrada o datos suministrados a la misma, los cuales pueden ser constantes, variables o expresiones.

Además de las sentencias que componen el cuerpo principal de la función, la definición de la misma también incluye la definición de sus parámetros.

La forma general de tal definición es:

```
FUNCIÓN nombre (parm1,  parm2 ...)  :  tipo
 VARIABLES
            Declaración de variables
 COMIENZO
            Cuerpo principal de la función
 FIN
```

(2) Definición de la función

Los argumentos del llamado de la función (1) y los parámetros de su definición (2), deben coincidir tanto en número y tipos, como en ubicación.

El tipo indicado en la cabecera de la definición califica el tipo de valor devuelto por la función, el cual podrá ser entero, real, lógico, carácter, cadena, etc...

Las sentencias propiamente dichas de la función, constituyen el cuerpo principal de la misma y al igual que en los programas está delimitado o comprendido por las palabras **Comienzo y Fin.**

Dentro de éste conjunto de sentencias, debe existir por lo menos una sentencia de asignación donde figure el **nombre de la función** a la izquierda del signo igual, con lo que asignaremos el valor a devolver por la función.

El retorno al programa que hizo el llamado, tiene lugar cuando se ejecutara la última sentencia del cuerpo de la función o cuando se encuentra con una instrucción **retorne.** El control pasa entonces al punto del programa en cual tuvo lugar la llamada de la función, reemplazando el nombre por su valor.

En el ejemplo mostrado abajo, la función recibe mientras los parámetros enteros n1 y n2, dos valores que según su condición serán asignados al nombre de la función.-

n1 y n2 : parámetros formales

```
FUNCIÓN   máximo ( n1  ,  n2   : entero  )  :   entero

COMIENZO

 Si   (  n1 >  n2  )
                  Entonces
                       máximo = n1
                  sino
                       máximo = n2

 FIN
```

Una llamada típica de la función podría ser:

x e y : parámetros actuales

```
Valor < - - -  máximo  ( x, y )
```

Donde las variables "x" e "y" , debieron haber sido declaradas en el programa que llama y al momento del llamado de la función también deben estar definidas.-

Al ejecutarse la instrucción de llamado (1), se transfiere el control a las sentencias de la función donde tiene lugar el paso de los argumentos a los parámetros " n1 " y " n2 ", los que permitirán a la función asignar el mayor de ellos al nombre de la misma y retornar.-

A modo de ejemplo consideremos el siguiente :

```
X =  27
Y =  13
 Valor =  máximo ( x ,  y  )
```

Ejecutadas estas instrucciones, la variable valor contendrá el mayor de ambos valores, o sea 27.-

PROCEDIMIENTOS

Un procedimiento está constituido por un bloque de sentencias similar a una función, pero tiene diferencias importantes en el modo de llamado y en la información retornada. Mientras que en el caso de una función, se obtiene un único resultado y los parámetros representan solo datos que se suministran a la función, con un procedimiento se pueden obtener tantos resultados como se desee (uno, varios o ninguno) y sus parámetros cuando existan, podrán ser unos datos y otros resultados.

Una segunda diferencia se observa en el modo de llamado o activado. Para una función su activado se realiza mediante el llamado desde una expresión, mientras que para el caso de un procedimiento, **su llamado es una instrucción más** y retorna la ejecución a la inmediata siguiente.-

La forma general es :

```
CALCULAR ( x,  y, z,  t,  u  )
```

Donde :

- Los parámetros x , y son solo **datos de entrada**
- Los parámetros z , t son solo **resultados de salida**
- El parámetro u se emplea como **entrada y salida**

PARÁMETROS - PASAJE POR VALOR Y POR REFERENCIA

Como mencionamos anteriormente, cada vez que llamamos a una función, **se establece una correspondencia automática entre los parámetros formales y los actuales.** Los métodos que analizaremos son:

- Pasaje de parámetros por valor
- Pasaje de parámetros por referencia

PASAJE DE PARÁMETROS POR VALOR

Esta es la forma más sencilla de pasar los parámetros, en el momento de efectuarse la llamada a la función, los parámetros formales que son locales a la función, reciben como valores iniciales, los valores de los parámetros actuales y, con éstos valores, comienza la ejecución de las instrucciones descriptas en el cuerpo de la función. Los parámetros actuales podrán ser en este caso tanto variables, constantes o expresiones.-

Al ejecutarse la siguiente instrucción en el programa principal

```
T = MICODI ( x, y )
```

Los parámetros formales **a y b** reciben los valores 10 y 15 respectivamente. Los cambios producidos en las variables a y b, **locales** a la función, no modifican los valores de las variables **x e y** del programa principal. Aunque los parámetros formales reciben los valores iniciales de los parámetros actuales, no existe otra conexión entre ambos. De hecho, todos los parámetros formales son solo de **entrada.-**

Programa principal

```
PROGRAMA : COMÚN

Comenzar

X = 10
Y = 15

T = MICODI ( x. Y )

Escribir x, y, t

FIN
```

Subprograma Función

```
FUNCIÓN MICODI ( a, b )

COMIENZO

Mientras ( a <> b )
 Si a > b
          Entonces
                A = a - b
           Sino
                B = b - a

    Fin_si

Fin_mientras

     MICODI = a

FIN
```

PASAJE DE PARÁMETROS POR REFERENCIA

En lugar de pasar el valor del parámetro actual como valor inicial para su respectivo parámetro formal, el pasaje de parámetro por referencia, también llamado por dirección, establece una conexión directa entre ambos parámetros.

Dicho de otro modo, el parámetro formal utiliza la dirección de memoria del parámetro actual, de esta forma cualquier cambio que se efectúe en las variables **a y b** alterarán los correspondientes valores de **x e y.**

Podemos observar entonces, que las variables a y b se comportan como variables **globales** en lugar de cómo **locales** que ocurría en el paso por valor.

Además debemos tener presente, que éste caso solo podremos usar variables como parámetros actuales y no valores constantes o expresiones.-

VARIABLES GLOBALES

Las variables declaradas en un programa que contenga funciones o procedimientos, son válidas (está disponibles) en cualquier parte del programa, tanto dentro de las funciones o procedimientos, como así fuera de ellos. A estas variables se las denomina **globales.-**

En cambio, las variables declaradas dentro de los procedimientos o funciones, son variables **locales o privadas** de los mismos y solo están disponibles en su propio ámbito.-

EFECTOS LATERALES

Con este nombre nos referimos a las modificaciones que pueden introducirse dentro de una función o procedimiento sobre los valores de determinadas variables globales. La recomendación es no utilizar, en lo posible este tipo de acceso, ya que es una forma muy peligrosa de programar aunque en ocasiones pueda resultar más cómoda y rápida.

En general cuando nos referimos a **efectos laterales,** nos referimos a acciones **indeseables o erróneas** normalmente muy difíciles de detectar. La forma de evitar estos efectos se logra mediante la utilización solamente de variables locales en las funciones y procedimientos, declarando como parámetros variables solamente aquellos que deban devolver un resultado y el resto como parámetros por valor o **protegidos**.-

RECURSIVIDAD VS. ITERATIVIDAD

La recursividad es un concepto lógico-matemático que permite la posibilidad de definir una función para determinados valores y utilizar estos valores para definir la función para el resto de sus valores. El concepto de recursividad es posible aplicarlo en aquellos idiomas que disponen de estructuras dinámicas (tipo pilas).-

Un procedimiento o una función es recursiva si puede llamarse a sí misma. La recursión puede ser usada como una alternativa de la iteración (repetición). No todos los problemas son aptos para ser expresados en términos recursivos y muchas veces estos son menos eficientes que los iterativos. Sin embargo, en muchos casos, el uso de la recursión permite implementar una solución simple y natural para resolver problemas que por otros procedimientos seria difícil de resolver. Por esta razón, la recursión es una importante y potente herramienta para la resolución de problemas complejos.-

FUNCIÓN FACTORIAL

Un ejemplo típico de una función recursiva es la función factorial, la cual expresamos matemáticamente como n!, E1 factorial de **n** es el producto de los enteros de 1 a n.-

$$n! = n \times (n-1) \times (n-2) \times .... 3 \times 2 \times 1$$

Así los sucesivos valores son:

$1! = 1 = 1$
$2! = 2 \times 1 = 2$
$3! = 3 \times 2 \times 1 = 6$
$4! = 4 \times 3 \times 2 \times 1 = 24$
$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

sin embargo, n! es posible expresarlo como

$$n! = n \times (n-1)!$$

Luego la expresión recursiva surge fácilmente:

6! = 6 x 5 x 4 x 3 x 2 x 1 = 6 x 5!
5! = 5 x 4 x 3 x 2 x 1 = 5 x 4!
4! = 4 x 3 x 2 x 1 = 4 x 3!
3! = 3 x 2 x 1 = 3 x 2!
2! = 2 x 1 = 2 x 1!
1! = 1

Para poder diseñar una función o procedimiento recursivo se precisa considerar **una condición de terminación o de salida,** ya que en caso contrario la recursión conduciría a un anidamiento sucesivo e infinito.-

La cláusula de escape o condición de salida en el caso de la función factorial es 1!=1, donde se define el proceso recursivo ya que la solución de 1! Es simplemente 1 y no está expresada en función de otro llamado a la función, como ocurría en casos anteriores.-

Luego la definición de una función o procedimiento recursivo consta de 2 partes a saber:

1.- Condición de escape o terminación donde se proporciona uno o más valores de la función.

2.- Un paso inductivo en el que los valores restantes quedan definidos en base a los anteriores.

La función factorial ya vista se puede expresar del siguiente modo:

n!	1 (si n = 0)
	n x (n -1) .... x 3 x 2 1 (si n > 0)

Aquí hemos tratado el concepto de recursión. De hecho, una llamada recursiva a un procedimiento o a una función, es una clase de ciclo en que se hace la secuencia de llamadas repetitivas. Muchas de las aplicaciones que utilizan la recursión, también pueden resolverse mediante iteración, por lo que ambos métodos pueden compararse.

Las llamadas recursivas son fáciles de describir y, por lo general, el código necesario para implementarlas es más pequeño. Por otro lado, suelen ser difíciles de depurar las llamadas recursivas que los procedimientos iterativos.

Las definiciones recursivas ahorran código en comparación con la iteración, pero necesitan en general mucha más memoria **RAM** para ubicar la pila. Sin embargo, cuando se programa de manera eficiente, los procedimientos recursivos por lo general se ejecutan con mayor rapidez, por que para cada problema habrá que analizar cual es la solución más apta a adoptar.-

CADENAS

Una cadena (string) es simplemente una lista o secuencia de caracteres que se tratan como una unidad. Las cadenas se identifican por estar delimitadas entre doble comillas o apóstrofes. Por ejemplo:

" Ingrese nombre y apellido "
o
´ Ingrese nombre y apellido ´

Estas tiras de caracteres son constantes cadenas. En esencia, una constante de cadena es un conjunto de caracteres encerrado entre comillas. La longitud de una cadena es el número de caracteres encerrado entre comillas.

" UTN MENDOZA" (cadena de 11 caracteres)

Una cadena se denomina nula o vacía si no contiene ningún carácter. En este caso no es lo mismo que la cadena con un blanco, cuya longitud es uno. La cadena nula se representa por dobles comillas seguidas.

"" (Cadena nula o vacía)

" " (Cadena de longitud 1 en blanco)

de igual modo, se debe tener cuidado con las cadenas que representan informaciones numéricas. Por ejemplo, "3" es una cadena de caracteres con la que no se puede operar aritméticamente, como puede ser sumarla, restarla, multiplicarla, etc...y 3 es una constante numérica que puede actuar en operaciones aritméticas.

Las cadenas y los números se almacenan de modo diferente en la memoria de la computadora. A su vez, las cadenas pueden ser variables o constantes.

Los nombres de las variables cadenas siguen en general las mismas reglas de formación que el resto de los nombres de variables.

Algunos idiomas tienen la facilidad de manejar cadena de longitud variable, las que pueden ampliarse o contraerse para almacenar diferentes números de caracteres.

Por su parte las cadenas de **longitud fija** contienen un número determinado de caracteres por lo que siempre tendrán la misma longitud independiente del tamaño de lo que se haya almacenado en ellas.

La declaración de cadena es:

Nombre	:	cadena [20 [	Declara a nombre con 20 caracteres.
Dirección	:	cadena [40 [	Declara a dirección con 40 caracteres.

Si el contenido de las cadenas no llega a ser igual a la longitud prefijada, las cadenas se justifican a la izquierda, y si es necesario se truncan a la derecha.-

OPERACIONES CON CADENA

Las operaciones básicas con cadenas son **Concatenación y Comparación,** ampliables a las operaciones basadas en función de cadena, como es el caso de las subcadenas.-

CONCATENACIÓN

La operación de **concatenación** de cadenas consiste en añadir una cadena a otra. Para tal fin utilizaremos el símbolo (+) como operador de Concatenación.

Como ejemplo de Concatenación, consideraremos lo siguiente;

Cad1 = " UNIVERSIDAD "
Cad2 = " TECNOLÓGICA "

Cad3 = Cad1 + Cad2

El signo (+) indica que Cad1 se ha concatenado con Cad2 y el resultado ha sido asignado a la variable Cad3.

COMPARACIÓN DE CADENAS

Las cadenas se comparan carácter a carácter de acuerdo a sus códigos numéricos ASCII. Un carácter es mayor que otro si su código ASCII es mayor.

Por ejemplo, el código ASCII del carácter **espacio en blanco es 32,** el correspondiente al **número 1 es 49,** el de la **letra "A" es 65** y el de la **letra "a" es**

97; eso significa que un espacio en blanco es menor que el dígito 1, que a su vez es menor que la letra "A" y ésta menor que la "a".-

" 2 " < " 7 "	Verdadero
" a " < " b "	Verdadero
" b " < " D "	Falso
" A " < " 5 "	Falso

La comparación se lleva a cabo de izquierda a derecha, tomando un carácter de cada cadena cada vez y comparando sus valores ASCII. Si los valores son diferentes, la cadena que contiene el carácter con el valor más bajo se considera menor que la otra cadena y la comparación se termina. Si los valores son iguales, se compara el siguiente par de caracteres, y así sucesivamente. Si todos los caracteres son iguales en valor, las cadenas son iguales.-

Por último si el final de una cadena se alcanza antes del final de la otra y todos los valores hasta ese punto han sido iguales, la cadena más corta se considera menor que la cadena más larga.

"Antonio" < "Bartolomé"	Verdadero
"no" > "NO"	Verdadero
"Marga" < "margarita"	Verdadero
"12 de octubre" > "Luis"	Falso
"aaaaa" > "aa"	Falso

FUNCIONES CADENA

En general todos los idiomas disponen de un grupo de funciones para el tratamiento de cadenas no habiendo en ellas una normalización.-

Una función cadena es aquella cuyo argumento es una cadena y devuelve un resultado tipo cadena, numérico o lógico.-

A continuación se detallan un conjunto de funciones cadenas adoptadas a nuestro lenguaje pseudocódigo.

CAR (x) **:** Acepta un parámetro entero y devuelve un ASCII equivalente carácter.

ASCII (c) **:** Acepta un parámetro carácter y devuelve un valor numérico entero correspondiente en código ASCII.

MAYUS (c) : Acepta un carácter alfabético en minúscula y lo devuelve en mayúscula. Si el carácter ya está en mayúscula no lo altera.

MINUS (c) : Idem a MAYUS pero convierte a minúscula.

COPIA (Cad, Pi, Nc) : Esta función extrae una subcadena de una cadena más larga. Copia de la cadena Cad desde la posición Pi un número de caracteres Nc.

BORRAR (Cad, Pi, Nc): Al igual de COPIA borra caracteres de una cadena. Se deben especificar el punto de comienzo (Pi) y el número de caracteres a borrar (Nc).

INSERTA (Cad1, Cad2, n): Inserta una subcadena Cad1 dentro de la cadena Cad2 comenzando por el carácter (n).

LARGO (Cad) : Devuelve el número de caracteres que contiene en ese momento la cadena Cad.

UBICA (Cad1, Cad2) : Devuelve la posición donde se encuentra la subcadena Cad1 en la cadena Cad2 si no es encontrada devuelve cero.

VALOR (Cadena, num, cad) : Este procedimiento acepta tres parámetros: la cadena, la variable numérica que va a recibir el valor NUM, y una variable entera que se utiliza para verificar la conversión.

TIRA (Num, Cadena) : Este procedimiento es inverso al VALOR, convierte un valor numérico en una cadena. Siempre será posible la conversión

ARCHIVOS

Un archivo o fichero es un conjunto de objetos o registros de la misma naturaleza relacionados entre sí y que se tratan como una sola unidad. Cada uno de estos registros se dividen a su vez en partes llamadas Campos, las cuales pueden contener por ejemplo, el nombre de una persona, su número de documento, su domicilio, su edad, etc... Los registros se identifican normalmente por un campo llamado Campo clave, los cuales suelen tener información vital para el registro o archivo.-

Los archivos son normalmente almacenados sobre discos, CD o diskette aunque, en ciertos casos, pueden ser almacenados sobre cintas magnéticas, siendo en éste caso su organización obligadamente secuencial.

ORGANIZACIÓN DE ARCHIVOS

Los archivos según el modo en que están ordenados sus registros, se los puede clasificar de la siguiente forma:

- Secuenciales.
- Relativos o directos.
- Indexados.

ORGANIZACIÓN SECUENCIAL

Los registros de un archivo de organización secuencial son ordenados físicamente en secuencia. Cada registro, excepto el primero, tiene otro registro que lo procede, y cada registro excepto el último, tiene otro que lo antecede.

El orden físico en el cual los registros aparecen es usualmente el mismo en que fueron originalmente escritos o grabados en el archivo.

ORGANIZACIÓN RELATIVA

Un archivo de organización relativa está compuesto por un conjunto de celdas de longitud fija ordenada en secuencia física, las cuales están numeradas desde su primer registro "1" hasta el registro enésimo "n", en el cual cada número representa la localidad o ubicación relativa del registro referida al comienzo del mismo.

Cada celda puede contener así un simple registro o directamente están en blanco (vacía). El número de celdas (número de registros) es usado para reverenciar un especificado registro del archivo durante su acceso.-

ORGANIZACIÓN INDEXADA

Los registros de un archivo indexado están ordenados por algún campo del mismo designado como clave. Luego, una clave es un campo de datos dentro del registro del archivo.-

Cuando se crea un archivo indexado, debemos decidir que campo o campos de los registros del archivo deben ser considerados como clave; el contenido de estos campos son usados para identificar un determinado registro.-

La longitud y posición de la clave para los distintos registros del archivo debe ser idéntica en todo el archivo.

Debemos definir al menos una clave para un archivo indexado. Esta clave se denomina clave primaria del archivo, y usualmente es único valor para cada registro.-

OPERACIONES TIPIAS SOBRE ARCHIVOS

Creación **:** Consiste en la escritura o grabación de los registros que van a conformar el fichero. Los datos pueden introducirse por teclado desde otro fichero o como resultado de algún proceso intermedio.

Consulta **:** Lectura de uno o más registros.

Actualizar **:** Consiste en añadir, modificar o dar de baja algún registro.

Clasificar **:** Consiste en reubicar los registros de tal forma que quedan Ordenados por algún campo determinado.

Fusión **:** A partir de dos ficheros de idéntica estructura, obtener un nuevo fichero con los registro de ambos.

Partición : Descomponer un fichero en dos o más atendiendo a cierta Característica de alguno de sus campos.

Borrado **:** Eliminación física del fichero sobre el soporte, dejando libre el espacio que ocupaba.