Unidad 2
METODOLOGÍA DE LA PROGRAMACIÓN
Desarrollo de la unidad : 27 h

Prácticas y Ejercicios :
Diseñar con el Word ordinogramas y organigramas
Problemas con pseudocódigo, seguimientos, Elaboración
Ejercicios con tablas de verdad
Conceptos:
Diagramas de flujo, Organigramas, Ordinogramas, pseudocódigo, tablas de decisión
Instrucciones simples, compuestas, Estructuras de control: secuencial, alternativa, repetitiva, variables auxiliares

Metodología: Métodos, normas, herramientas utilizados para la elaboración de programas

Herramientas de Diseño:

- **Herramientas gráficas o visuales**

Diagramas de Flujo
Conjunto de símbolos normalizados, conectados mediante líneas de flujo que muestran la secuencia de pasos de un programa, las acciones que realiza o el origen y destino de los datos

Tipos:
Organigramas (Visión externa del programa)
Ordinogramas (Visión interna, detallada del programa)
Diagramas de transición de estados.

- **Herramientas textuales**

Pseudocódigo: Lenguaje de intermedio entre el lenguaje natural y el lenguaje de programación.

ORGANIGRAMAS

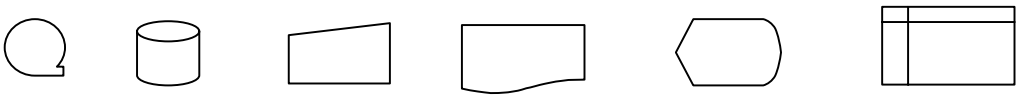
- También llamados diagrama de flujo del sistema (Externo)
- Representación gráfica de los ficheros y dispositivos físicos que interviene en un proceso.
- Visión general del programa, E/S del programa, dispositivos utilizados

Normas

1. Arriba dispositivos de entrada
2. A Izda/dcha Dispositivos de salida
3. Abajo dispositivos de salida
4. Centro el nombre del proceso, tarea, programa

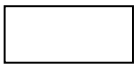
Elementos

Símbolos de soporte E/S

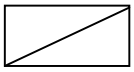


Cinta Disco Teclado Impresora Pantalla Almacenamiento interno

Símbolos de proceso



Líneas de conexión



Salida o Entrada Salida o entra por una línea de comunicación Módem, dispositivo de conexión

Ejemplos y ejercicios

1. Proceso de consulta de un usuario de una agencia de viajes sobre vuelos almacenados en un ordenador central
2. Modificación de la bases de datos de clientes en una aplicación comercial
3. Selección y impresión de informes de contabilidad
4. Consulta de una pagina web (conectado o no conectado)
5. Emisión de facturas
6. Anotar pedidos y enviarlos a la central

ORDINOGRAMAS

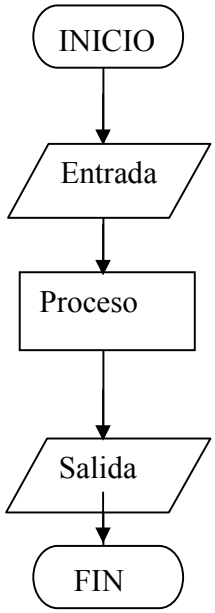


Diagrama de flujo de programa (Interno)
Secuencia lógica detallada de las operaciones que realiza el programa

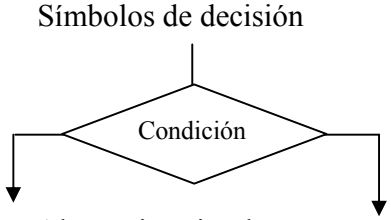
Normas
- Debe existir un principio y un final.
- La secuencia detallada de operaciones se realizan de arriba a abajo.
- Todos los elementos están conectados por líneas de flujo de datos sin que las líneas se crucen entre sí.
- El ordinograma en conjunto debe guardar una cierta simetría.

Esquema básico :
Inicio – Entrada de datos – Proceso – Salida de Resultados – Fin

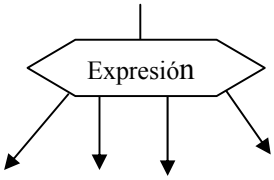
Elementos Básicos

- Símbolos de operación o proceso

- Símbolos de decisión

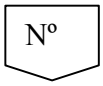


Alternativa simple



Alternativa múltiple

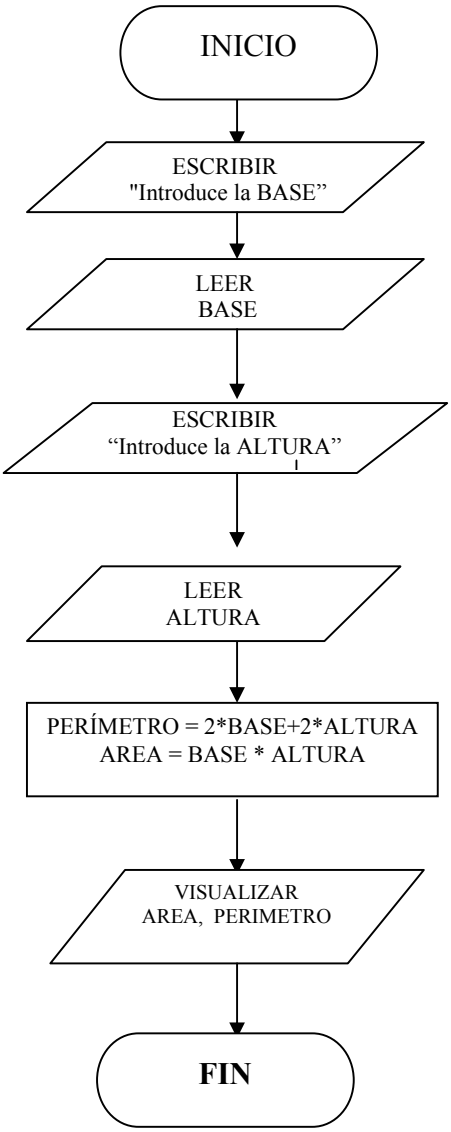
Elementos de interconexión:

SIMBOLOS	FUNCIÓN
	**Conector de reagrupamiento :** Utilizado para el reagrupamiento de líneas de flujo
	**Conector de una misma página**: Sirve para conectar dos puntos o partes del ordinograma en la misma página.
	**Conector de distintas páginas**: Sirve para conectar dos puntos del ordinograma en páginas diferentes.

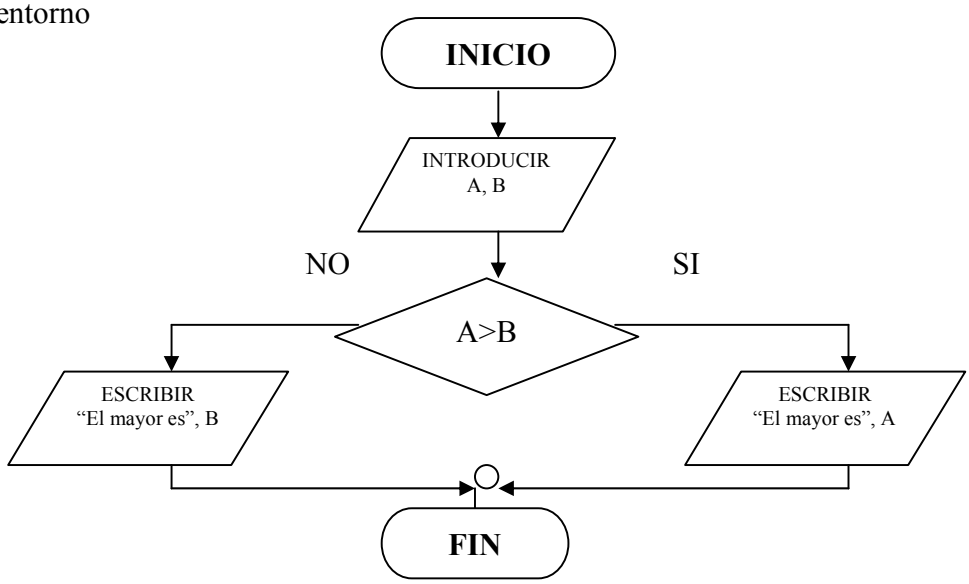
Los ordinogramas son bastante fácil de entender y seguir pero laborioso de realizar y modificar, se necesita mucho espacio para representar un programa completo. Hoy están en desuso, sólo se utiliza en casos muy sencillos como en la descripción de funcionamiento en una guías para usuario, de instalación, etc, pero no es habitual en programación donde se utiliza mayoritariamente los pseudocódigos.

Ejemplo 1: Ordinograma del algoritmo que calcula el perímetro y el área de un rectángulo:

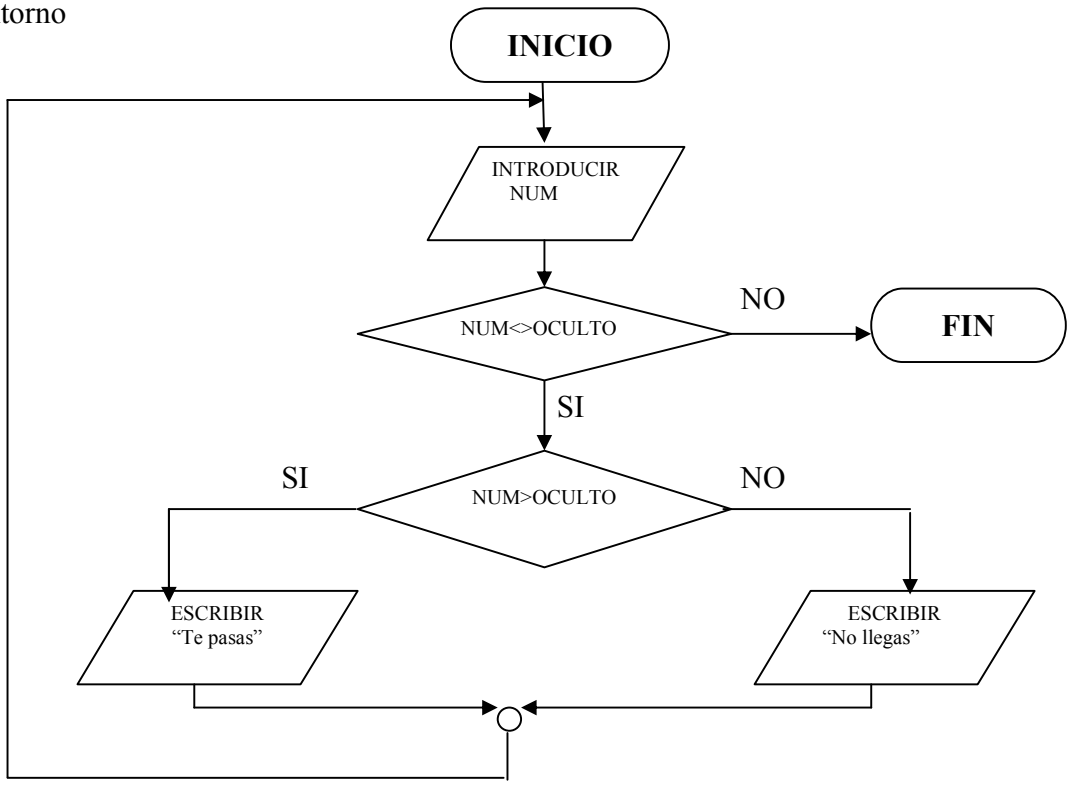
```
Algoritmo REC_AREA_PERIMETRO
        Entorno
                Entero BASE, ALTURA, PERIMETRO, AREA
        fin_entorno
```



Ejemplo 2: Introducir dos números y visualizar el mayor.
Entorno
    Real A, B
Fin_entorno



Ejemplo 3:Introducir un número por teclado, si no se adivina el número oculto, borrar pantalla, indicar si el nº es superior o inferior. Repetir la operación hasta acertar.
Entorno
    Entero NUM
    Constante OCULTO=19
Fin_entorno



Ejercicios sobre ordinograma:

1. Leer dos números, sumarlos y escribir su resultado
2. Leer dos números e indicar cual es el mayor
3. Leer tres números e indicar cual es el mayor
4. leer tres números e indicar cual es el mayor y cual el menor
5. leer dos números y una operación (+ - * /) y mostrar el resultado
6. Un programa que lea 4 número y calcule la media
7. Muestre el mensaje "procesando datos... desea continuar (s/n)", si el usuario introduce 'S', se repite el mensaje y si dice 'N' muestra el mensaje adiós y termina.
8. Un programa que lea un número, que compruebe que está comprendido entre 10 y 100, que lo muestre por pantalla o que lo vuelva a leer en el caso que no cumpla la condición.
9. Un programa que lea un valor N, que indica cuantos números va a leer, y calcule la suma y la media.
10. Un programa que lea números, los sume hasta que el usuario introduzca el número 0, entones los muestra la suma y la media.
11. Un programa que lea 200 números y me indique cual el máximo y el mínimo.
12. Un programa que sume el valor de los número pares de 1 a 30
13. Realizar un programa que muestre el valor de una factura telefónica sabiendo que cada paso consumido se cobra a 0.10 Euros y que cuando se consumen más de 1000 pasos se aplica un descuento del 18 % sobre el total de la factura. El número de pasos consumidos se solicita al usuario. Hay que chequear que este valor siempre es mayor que 0.
14. Elaborar un programa que muestre el precio de un billete de autobús, que se calcula en base a los kilómetros de trayecto (0.30 Euros por Km), pero si el recorrido supera los 80 Km se aplica un 15 % de descuento y que si el trayecto se realiza en día laborable ('L') hay un 5 % de descuento respecto si es día festivo ('F'). Datos ha introducir: kilómetros de recorrido y tipo de dia.

PSEUDOCÓDIGO

- Lenguaje intermedio entre natural y de programación
- Es más libre que un lenguaje de programación concreto pero mantiene las reglas de programación estructurada
- Independiente del lenguaje de programación
- Permite un diseño Top-Down (Niveles de abstración)
  General a lo concreto, Divide y vencerás.
- Fácil de entender, corregir y modificar

El pseudocódigo es una técnica de representación de algoritmos no gráfica que nos permite describirlos mediante un lenguaje intermedio entre el lenguaje natural que normalmente utilizamos en nuestra comunicación escrita (el español) y el lenguaje de programación que posteriormente vamos a utilizar (C, C++, Java, etc).

Esta característica permite escribir la solución de un problema utilizando palabras y frases en lenguaje natural sujetas a unas determinadas reglas que luego facilitan la traducción del algoritmo a un programa escrito en el lenguaje de programación determinado.

Ventajas del pseudocódigo:

- Permite que durante la fase de diseño, los programadores se centren en la lógica y estructuras de control del algoritmo (descripción de la secuencias de pasos que hay que llevar a cabo) y se olviden de las reglas y restricciones sintácticas (como hay que escribir dicha secuencia) que impone un determinado lenguaje de programación.
- La descripción o representación de los algoritmos que obtenemos es más fácil de crear y de entender, pues está realizada en el lenguaje que utilizamos habitualmente, no siendo necesario por tanto el conocimiento de un lenguaje de programación.
- La descripción o representación de los algoritmos que obtenemos es totalmente independiente de lenguaje de programación que posteriormente vayamos a utilizar.
- Facilita la realización de futuras correcciones o actualizaciones gracias a que no es un sistema de representación rígido.

La descripción del algoritmo que obtenemos mediante pseudocódigo, que **no es ejecutable por un ordenador**, se considera como un primer borrador del programa que vamos a desarrollar en la fase de codificación, ya que como ya hemos dicho, la descripción obtenida en pseudocódigo es fácilmente traducible a un programa.

Estructura o partes de un algoritmo:
Cabecera o Identificación
Cuerpo
- Entorno: Definición de datos
- Proceso: Definición de instrucciones

EJEMPLO:

ALGORITMO: Descuento en compra
/* Este algoritmo calcula el tanto por ciento de descuento que han hecho al realizar una determinada compra */
ENTORNO:
/* En este bloque irán la declaración de los distintos elementos u objetos que vamos a utilizar en un algoritmo: tipos definidos por el usuario variables , constantes, funciones y procedimientos, etc.*/
PROCESO:
INICIO
/* En este bloque se representarán las distintas instrucciones que formen parte del bloque de instrucciones del algoritmos que estamos describiendo .*/
FIN

ENTORNO DEL ALGORITMO:

Se definen todas la variables y constantes que vamos ha utilizar en el programa.
Cada variable y constante debe tener un nombre y pertenecer a un tipo de dato**.**

Constantes
Constante tipo1 NombreConstante11,...., Constante NombreConstante1n;
Constante tipo2 NombreConstante21,..., Constante NombreConstante2n;
...
Constante tipom NombreConstantem1,...,Constante NombreConstantemn;
Variables
tipo1 NombreVariable11,....NombreVariable1n;
tipo2 NombreVariable21,...,NombreVariable2n;
...
tipom NombreVariablem1,...,NombreVariablemn;

Ejemplo:

Constantes
Real Pi = 3.1416;
Entero Numelementos = 10;

Variables
Real Radio, Superficie;
Entero contador;
Carácter letra1;

PROCESO DE UN ALGORITMO:

- Descripción detallada de la secuencia de instrucciones que realiza el algoritmo

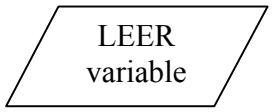
Tipos de instrucciones:

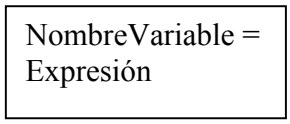
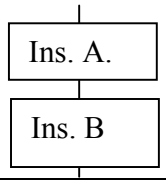
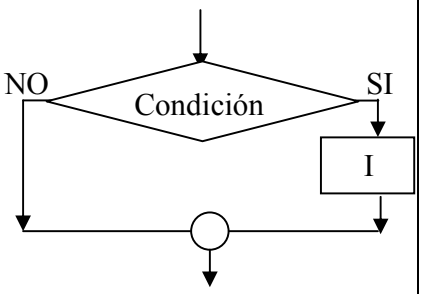
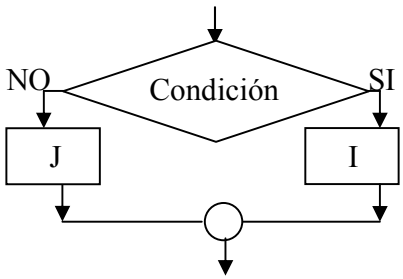
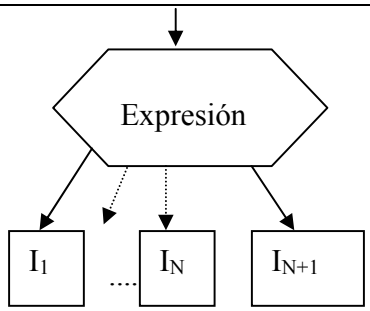
Básicas:

- Entrada
- Salida
- Asignación

Instrucciones de control / Estructuras de Control

- Estructura Secuencial
- Estructuras alternativas / condicional
  Simple
  Doble
  Múltiple
- Estructuras repetitivas / iterativas
  Mientras
  Hasta
  Para

Categoría de Instrucción	Representación en un ordinograma	Representación en un pseudocódigo
Instrucción de Entrada		LEER variable;
Instrucciónes de Salida		ESCRIBIR "texto"; ESCRIBIR "texto", variable ESCRIBIR "texto", variable; "texto";

Tipo de Instrucción	Representación en un ordinograma	Representación en un pseudocódigo
Instrucción de asignación		NombreVariable = Expresión
Secuencia de instrucciones		Ins. A Ins. B
Alternativa Simple		**SI** Condición **ENTONCES** I ; **FIN-SI**
Alternativa Doble		**SI** Condición **ENTONCES** I; **SINO** J; **FIN-SI**
Alternativa Múltiple		**SEGÚN** Expresión Valor1: I_1 ; Valor2: I_2 ; ..... ValorN: I_N ; [**En otro Caso**: I_{N+1} ;] **FIN-SEGUN**

EJEMPLOS BÁSICOS I.

1. Leer dos números, sumarlos y escribir su resultado

```
PROGRAMA Ejemplo1
  /* Lee dos número los suma y me muestra el resultado */
ENTORNO
  Variables
     Entero A,B, Suma
 INICIO
     Leer A
     Leer B
     Suma = A +  B
    Mostar Suma
FIN
```

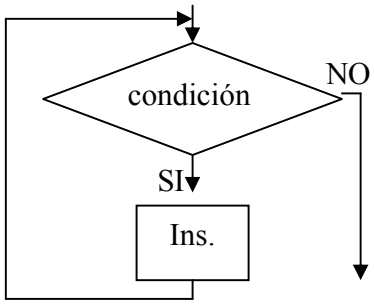
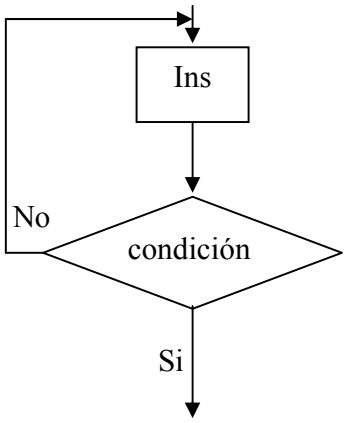
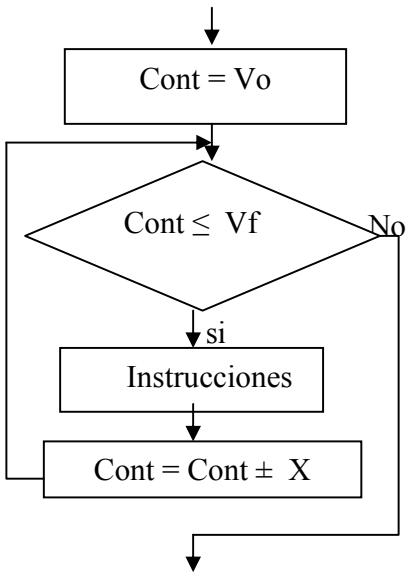
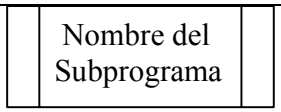
2. Leer dos números e indicar cual es el mayor

```
PROGRAMA Ejemplo2
  /* Leer dos número e indicar cual es el mayor */
ENTORNO
   Variables
        Entero  A, B
INICIO
   Leer A
   Leer B
   SI ( A > B)
          ENTONCES
             Mostrar A
          SINO
             Mostrar B
   FIN-SI
INICIO
```

3. Leer dos números y una operación (+ - * /) y mostrar el resultado

```
PROGRAMA Calculadora
/* Lee dos número, un carácter operación y realiza el cálculo indicado */
ENTORNO
  Variables
          Carácter Operador
          Entero A, B
          Entero Resultado
INICIO
   Leer A
   Leer B
   Leer Operador
   SEGÚN  Operador
          '+': Resultado = A + B
          '-' : Resultado = A - B
          '*': Resultado = A * B
          '/' : Resultado = A / B
   FIN-SEGÚN
   Mostrar Resultado
FIN
```

ESTRUCTURAS REPETITIVAS:

Tipo de Instrucción	Representación en un ordinograma	Representación en un pseudocódigo
Instrucción Mientras:		**MIENTRAS** (Condición) Instrucciones ..... **FIN-MIENTRAS**
Instrucción Repetir		**REPETIR** Instrucciones ..... **HASTA** (Condición)
Instrucción Para		**PARA** Cont desde Vo **HASTA** Vf **Con [Incremento | Decremento]** X Instrucciones **FIN-PARA**
Llamada a Subrutina, realizar un subprograma		**Nombre (Paramétros)** **Datos Entrada :**E1,E2.. **Datos Salida :**S1, S2..

EJEMPLOS BÁSICOS II

1. Muestre el mensaje "procesando datos... desea continuar (s/n)", si el usuario introduce 'S', se repite el mensaje y si dice 'N' muestra el mensaje "adiós" y termina.

```
PROGRAMA MensajeSN
/* Pide confirmación para continuar */
ENTORNO
  Variables
        Carácter letra
INICIO
        REPETIR
            Mostrar "Procesando datos...desea continuar (s/n):"
            Leer letra
    HASTA (letra = 'N')
    Mostar "Adiós"
FIN
```

2. Un programa que lea números, los sume y termine cuando que el usuario introduzca el número 0, mostrando la suma y la media.

```
PROGRAMA Sumador-Mientras
/* Suma una serie de número enteros hasta leer el número cero */
ENTORNO
  Variables
      Entero Número, Contador, Suma, Media
INICIO
  Contador = 0
  Suma = 0
  Leer Número
  MIENTRAS ( Número ≠ 0 )
    Suma = Suma + Número
    Contador = Contador + 1
    Leer Número
  FIN-MIENTRAS
  Media = Suma / Contador
  Mostrar Suma
  Mostrar Media
FIN
```

3. Un programa que lea un valor N, que indica cuantos números va a leer, y calcule la suma y la media.

```
PROGRAMA Sumador-Para
/* Suma una serie de número determinados */
ENTORNO
  Variables
      Entero N, Número, Contador, Suma, Media
INICIO
  Suma = 0
  Leer N
  PARA Contador = 1 HASTA N HACER
    Leer Número
    Suma = Suma + Número
  FIN-PARA
  Media = Suma / Contador
  Mostrar Suma
  Mostrar Media
FIN
```

Más ejemplos:

1. Dado un número por teclado, decidir si es par o impar.

```
Algoritmo PAR_IMPAR
        //Determina si un número introducido es par
Entorno
        Entero A
fin_entorno
Inicio
        Escribir "Introduce un número:"
        Leer A
        Si (A % 2=0) entonces
                Escribir "el número",A,"es par"
        sino
                Escribir A,"es un número impar"
        fin_si
fin
```

2. Dados dos números por teclado, decir si un número es múltiplo de otro:

```
Algoritmo MULTIPLO
        //Determina si un número es múltiplo de otro
Entorno
        Entero A,B
fin_entorno

Inicio
        Escribir "Introduzca dos números:"
        Leer    A,B
        Si (  (A>B) and (  (A % B) = 0))
            entonces
                Escribir  B,"es múltiplo de",A
        sino
                Si ( (B>A) and ( (B %A) = 0))
                    entonces
                        Escribir A,"es múltiplo de",B
                sino
                        Si (A=B)
                            entonces
                                Escribir   "son iguales"
                        sino
                                Escribir   "no son múltiplos"
                        fin_si
                fin_si
        fin_si
fin
```

3. Dado el día, mes y año de nacimiento de una persona y dado también el d, m, a actual, nos diga su edad:

```
Algoritmo EDAD
        //Determina la edad de una persona conocida la fecha de nacimiento
Entorno
        Entero D, M, A, DIA, MES, AÑO /*datos actuales: D, M, A y
                                                de nacimiento: DIA, MES, AÑO  */
        Entero X                                // Edad
fin_entorno

Inicio
        Escribir  "Introduce la fecha actual día mes año: "
        Leer D, M, A
        Escribir  "Introduce tu fecha de nacimiento día mes año: "
        Leer DIA, MES, AÑO
        X=A-AÑO
    Si (M>MES)
        entonces
            Escribir  "Tienes ", X ," años."
    sino
            Si (M<MES)
                entonces
                    Escribir  "Tienes ", X-1 ,"años."
            sino
                    Si ((M=MES)and(D>=DIA))
                        entonces
                            Escribir "Tienes ", X ," años."
                    sino
                            Escribir "Tienes ", X-1 ," años"
                    fin_si
            fin_si
    fin_si
fin
```

Este programa podría realizarse de forma más simple agrupando condiciones:

```
Si ((M>MES) or ((M=MES)and(D>=DIA)))
        entonces
            Escribir  "Tienes ", X ," años."
 sino
            Escribir  "Tienes ", X-1 ,"años."
fin_si
```

Cálculo del factorial de un número realizado con los tres tipos de ciclos: Mientras, repetir y para:

N! = N * (N-1) * (N-2)*....* 1.

```
Algoritmo FACTORIAL
        // Algoritmo que calcula el factorial de un número natural
Entorno
        Entero NUM, INI, FAC
fin_entorno
Inicio
   FAC =1
   Escribir "Introduce un número mayor que 1"
   Leer NUM
   INI=NUM
   Mientras NUM > 1
            FAC = FAC * NUM
            NUM = NUM - 1
    Fin_mientras
     Escribir "El factorial de", INI , " es ", FAC.
Fin
```

```
Algoritmo FACTORIAL
        // Algoritmo que calcula el factorial de un número natural
    Entorno
            Entero NUM, FAC
    Fin_entorno
    Inicio
       FAC = 1
       Escribir "Introduce un número mayor que 1"
       Leer  NUM
       Repetir
            FAC = FAC * NUM
            NUM = NUM - 1
       Hasta NUM = 1
       visualizar NUM
    fin
```

```
Inicio
   Escribir " Dame un nº: "
   Leer NUM
   FAC = 1
   Para I = 1 hasta NUM incremento 1
        FAC = FAC * I
   Fin_para
   Escribir "El factorial  es : ",FAC
Fin
```

```
Inicio
   Escribir " Dame un nº: "
   Leer NUM
   FAC = NUM
   Para I = NUM-1 hasta 1 decremento 1
        FAC = FAC * I
   Fin_para
    Escribir "El factorial es: ",FAC
Fin
```

VARIABLES AUXILIARES DE UN ALGORITMO O PROGRAMA

Son objetos que utiliza un algoritmo y que por la frecuencia con la que se utilizan dentro de un algoritmo y por la función que realizan dentro del mismo toman un nombre especial: Contadores, acumuladores e interruptores

Contadores.

Un contador es una variable destinada a almacenar un valor que se irá incrementando o decrementando en una **cantidad constante**.

Se suelen utilizar mucho en procesos repetitivos, para contabilizar el número de veces que se repite un conjunto de acciones o eventos, es decir en los cuerpos de las instrucciones repetitivas.
Sobre un contador se realizan dos operaciones básicas:

Inicialización:
Todo contador se debe inicializar con un valor inicia (0, 1...)
contador = Valor_Inicial
Incremento
Cada vez que aparezca el evento a contar se ha de incrementar o decrementar en una cantidad fija (I, D respectivamente) el valor del contador.
contador = contador+ I;
contador = contador- D;

Ejemplo: Cuenta el números de A's que un usuario introduce hasta finalizar con la pulsación de un *

```
INICIO
        contadorAs = 0
        Escribir "Introduce un caracter"
        Leer letra
        MIENTRAS (letra <> "*") HACER
                SI  letra = "A"
                entonces
                        contadorAs = contadorAs + 1
                FIN SI
                Escribir "Introduce un caracter"
                leer letra
        FIN MIENTRAS
        Escribir  contadorAs
Fin.
```

Acumuladores

Un acumulador o totalizador es una variable cuya misión es acumular cantidades sucesivas obtenidas al realizar la misma operación. Realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante como en el caso del contador.

El uso más habitual de un acumulador es obtener sumas y productos. Al igual que con los contadores, para poder utilizar un acumulador hay que realizar sobre ellos dos operaciones básicas:
Inicialización: SumaTotal=0; ProductoFinal=1;
Acumulación:
Una vez obtenido y almacenado en una variable la cantidad a acumular la añadimos a la variable acumulador :

```
SumaTotal = SumaTotal + cantidad;
ProductoFinal = ProductoFinal * cantidad;
```

Ejemplo:
Leer números y muestra su suma hasta que el usuario pulse S para terminar

```
INICIO
   Suma = 0
   Leer Número
   REPETIR
      Suma = Suma + Número
      Mostrar " Desea continuar (s/n):"
      Leer Continuar
   HASTA ( Continuar = 'S')
    Mostrar " El total es : " Suma
FIN
```

Interruptores , conmutadores bandera o switches, flag, indicadores

Un interruptor, conmutador, bandera o switch es una variable que puede tomar dos posibles a lo largo de la ejecución del programa. Los valores que toma son normalmente 1 o sí (encendido/abierto) y 0 o no (apagado/cerrado) (de ahí su nombre de interruptor). Se utilizan principalmente para:
a) Recordar en un determinado lugar del programa la ocurrencia o no de un un suceso:

Ejemplo: Algoritmo que lee una secuencia de notas (hasta que se introduzca el valor -1) , nos calcula la media y nos dice si hubo o no una nota con valor diez:
Programa: Ejemplo de Conmutador 2

```
ENTORNO
        Real nota
        Lógico HaySobresaliente;
INICIO
        HaySobresaliente = falso;
        REPETIR
                Escribir " Introduce una nota"
                Leer nota
                SI (nota = = 10)
                        HaySobresaliente = verdadero
                FIN SI
        HASTA (nota = -1)
        SI  ( HAySobresaliente = verdadero)
        Entonces
                Escribir " Al menos hay una nota que es un 10"  ;
        Sino
                Escribir " Ninguna nota ha sido un10";
        FINSI
FIN
```

b) Realizar de forma alternativa e independiente dos procesos alternativos
Ejemplo: Sumar por un lado los números pares comprendidos entre 1 y100 y por otro los impares:

```
Programa: Ejemplo de Conmutador 2
ENTORNO
```

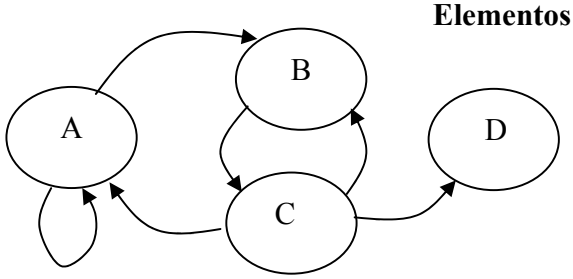
```
        Entero SumaPares SumaImpares, I
        Booleano Es_Par;
INICIO
        SumaPares=0;
        SumaImpares=0;
        Es_Par=falso;
        PARA I=1 hasta 100 incremento 1
                SI (Es_Par = = falso )
                Entonces
                        SumaImpares=SumaImpares+ I;
                        Es_Par=verdadero
                FIN_SI
                SI (Es_Par = = verdadero)
                Entonces
                        SumaPares=SumaPares+ I;
                        Es_Par=falso;
                FIN_SI
        FIN_PARA
        Escribir " La suma de los primeros 100 números impares es:"  SumaImpares
        Escribir " Y la de los primeros 100 números ares es:" SumaImpares
Fin
```

DIAGRAMAS DE TRANSICIÓN DE ESTADOS

Permiten definir los distintos estados por los que pasa un sistema en función de una serie de eventos, así como las operaciones que realiza.

Elementos: Estados, Eventos, Operación

Estados:
-Inicial
-Intermedios
-Final




```
PROGRAMA  : Autómata1
 Reconocer la secuencia AAA y termina
ENTORNO
 TipoEstados = (INICIO,A1,A2,FIN)
 Carácter  Letra
 TipoEstados  Estado
INICIO
 Estado = INICIO
REPETIR
   Leer Letra
   SI (Letra = 'A')
     ENTONCES
        SEGUN Estado
          INICIO : estado <- A1;
          A1 : estado <-  A2;
          A2 : estado <-  FIN;
        FIN-SEGUN
     SI NO
       Estado <- INICIO
   FIN-SI
HASTA  (Estado = FIN)
FIN
```

```
PROGRAMA : Autómata2
Contar las palabras introducidas hasta
encontrar un punto
ENTORNO
   Lógico   Haypalabra
   Carácter  Letra
   Entero   Contapalabra
INICIO
    Haypalabra = NO
    Contapalabra = 0
   Leer Letra
   MIENTRAS (Letra <> '.' )
        SI  (Letra >= 'A'  AND  letra <= 'z')
              Haypalabra = SI
       si no
              SI Haypalabra = SI
                    Contapalabra =
                              Contapalabra  +1
                    Haypalabra = NO
              FIN SI
        FIN SI
       Leer Letra
   FIN MIENTRAS
   SI Hayplabra = SI
      Contaplabra = Contapalabra + 1
   FIN SI
   Mostrar " Total de palabras ",
Contapalabra
FIN
```

Ejercicios

1. Mostrar todos los caracteres introducidos hasta el carácter, que no estén entre paréntesis, se supone que no existen paréntesis anidados.
2. Leer caracteres hasta reconocer la secuencia "SOS"
3. Leer caracteres hasta recibir el carácter '.' y contar cuantas veces aparece la secuencia "PTS"
4. No mostrar los comentarios de un programa en C (Entre /* y */

PROGRAMACIÓN CONVENCIONAL

No está limitada por ninguna estructura de control, puede utilizar libremente la instrucción de salto. Implica poca claridad de los programas, difícil de entender y corregir, sin metodología, los programas son obras de artesanía, no de ingeniería.

PROGRAMACIÓN ESTRUCTURADA

Desarrollada a partir de los trabajos de E.W.Dijkstra:

"se puede realizar cualquier algoritmo con un conjunto limitado de estructuras de control y se considera la instrucción goto o salto incondicional perjudicial para la compresión de los programas".

Los autores Bohm y Jacopini demostraros que cualquier programa propio se puede escribir empleando sólo con tres tipos de estructuras básicas de control.

- La secuencia de instrucciones
- La sentencia condicional
- La sentencia repetitiva

Para poder entender bien este teorema, vamos a introducir previamente dos conceptos que intervienen en su enunciado de una forma directa o indirecta.

1º) Denominamos programa propio, a aquel programa que cumple las siguientes condiciones:

- Posee un solo principio y un solo fin.
- Todo elemento del programa es accesible, es decir, existe al menos un camino desde el inicio al fin que pasa a través de él.
- El programa no posee bucles infinitos.

2º) Decimos que dos programas son equivalentes si realizan, ante cualquier situación de datos, el mimo trabajo pero de distinta forma.

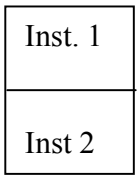
Teniendo en cuenta estos dos conceptos, el Teorema de Básico de la Programación Estructura dice:

"*Todo programa propio, realice el trabajo que realice, tiene siempre al menos un programa propio equivalente que sólo utiliza las estructuras básicas de control: Secuencial, Alternativa y Repetitiva, para representar las distintas acciones que deben llevarse a cabo cuando se ejecute el programa."*

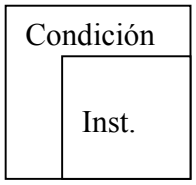
En definitiva, el teorema de estructura nos viene a decir que diseñando programas con instrucciones primitivas (lectura, escritura y asignación) y las estructuras de control básicas anteriores, no sólo podremos hacer cualquier trabajo sino que además conseguiremos mejorar la creación, lectura, comprensión y mantenimiento de los programas.

Un programa estructurado se puede leer directamente de principio a fin, va tener un sólo inicio y una sola terminación, utilizando exclusivamente los tres tipos de instrucciones de control.

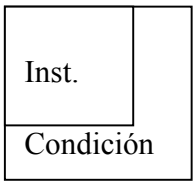
Diagramas Nassi-Shneiderman (Chapin) son una herramienta gráfica que obliga a la programación estructurada:



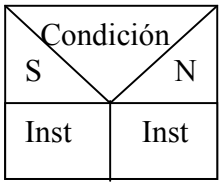
Secuencia



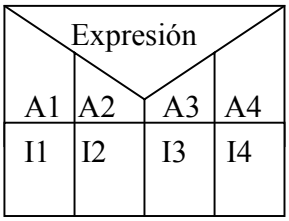
Mientras



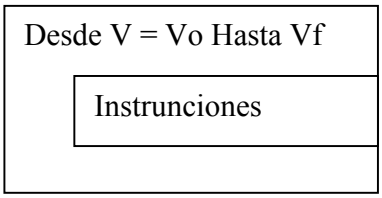
Hasta



Si



Según



Para

Ejemplos. Elaborar el ordinogramas, Pseudocódigo y Diagramas N-S de los siguiente programas:

1. Leer números hasta que se introduzca el valor 0 calculando la media de los valores positivos

2. Leer Letras hasta que se introduzca el valor * y calcular cuantas vocales se han introducido.

```
PROGRAMA Media Positivos (1)
ENTORNO
   Entero  Suma, Contador
INICIO
  Suma <- 0;
  Contador <- 0;
  Leer Num
  MIENTRAS (Num <> 0)
      SI ( Num > 0)
          ENTONCES
             Suma <- Suma + Num
            Contador <- Contador +1
     FIN-SI
     Leer NUM
  FIN-MIENTRAS
  Mostrar  "La media es igual a ", Suma / Num
FIN
```

PROGRAMACIÓN MODULAR

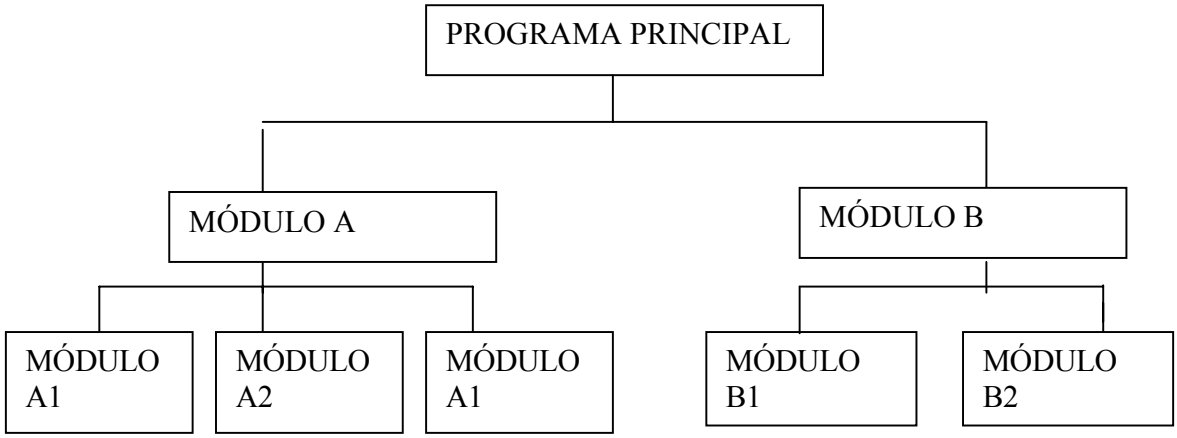
Principio: Descomponer el problema en distintos módulos o subprogramas que resuelvan cada uno de ello una parte del problema, para ello la programación modular utiliza dos métodos de trabajo: Diseño Top–Down y Recursos abstractos

- Diseño Top – Down (Diseño descendente)

Descomponer el problema en distintos subproblemas en un primer nivel, descomponer esos subproblemas en otros más simples, y continuar descendiendo definiendo distintos niveles de detalle hasta que cada subproblema sea suficientemente sencillo para que pueda ser resulto directamente desarrollando su algoritmo.

- Recursos Abstractos

Suponer que tenemos un ordenador maravilloso que admite cualquier tipo de instrucciones, Definimos el problemas utilizando esas instrucciones abstractas que nos resuelven el programa, A continuación se definen cada una de las anteriores instrucciones utilizando nuevas instrucciones abstractas que solucionen el subproblema. Continuamos sucesivamente hasta que las instrucciones sean directamente realizables por el lenguaje de programación elegido.



ESTRUCTURA DE UN PROGRAMA MODULAR

Teniendo en cuenta la descomposición modular, un programa quedará constituido por dos partes claramente diferenciadas:

1.- Programa o módulo Principal:

Describe la solución completa del problema y consta principalmente de **llamadas** a los distintos módulos en los que se divide el programa. Estas llamadas son indicaciones al procesador de que debe continuar la ejecución del programa en el módulo llamado, regresando al punto de partida una vez lo haya concluido. El programa principal puede contener además de las llamadas a módulos, su propias instrucciones de E/S, de asignación y de control, etc. Un programa principal deberá contener pocas instrucciones, y en él se debe ver claramente los diferentes pasos del proceso que se ha de seguir para la obtención de la solución buscada.

2.- Módulos secundarios

Su estructura coincide básicamente con la de un programa principal, con alguna diferencia en su encabezamiento y finalización. En consecuencia, un módulo secundario puede tener sus propios módulos, denominados submódulos, asociados a un refinamiento del mismo. La función principal de un módulo es resolver de modo **independiente** uno de los subproblemas en los que se divide el problema inicial. Un módulo no se ejecuta por si mismo sino cuando es invocado por el módulo principal o por otro módulo.

TIPOS DE MÓDULOS

En función de su situación física con respecto al módulo que los invoca

Módulos Internos
Son aquellos que están definidos en el mismo fichero fuente que el módulo que los invoca.

Módulos Externos
Son aquellos que figuran físicamente separados del módulo que los invoca, es decir, en otros ficheros fuentes. Estos módulos pueden ser compilados separadamente, e incluso pueden haber sido codificados en un lenguaje de programación distinto al del programa principal. Generalmente se enlazan con el programa principal en la fase de montaje o linkado, cuando ya son módulos objeto, es decir, cuando ya están traducidos al lenguaje máquina.

En función del momento en el que han sido desarrollados

Módulos de usuario o de programa
Son aquellos que son desarrollados en el programa actual para resolver un los distintos subproblemas en los que se divide el problema inicial.

Módulos de librería
Son módulos que han sido desarrollados previamente (normalmente por terceras personas) y que están almacenados en unos ficheros **externos** (no son ejecutables por si mismos, ya están compilados y se enlazan en la fase de montaje o linkado).

Las librerías tienen una doble utilidad: de una parte facilitar la reutilización del código, y de otra liberar al programador de tareas de codificación tediosas de bajo nivel. Además facilitan la portabilidad del software cuando diferentes sistemas utilizan el mismo estándar de codificación. Las librerías en función de cómo han sido creadas se utilizarán de modo diferente. Teniendo en cuenta este criterio, las podemos clasificar en dos tipos fundamentales:

Librerías de enlace estático:
Son módulos objeto generados a partir del código fuente de la declaración y definición de las funciones. Estas son utilizadas cuando el programa es linkado. El linkador extraerá de estas librerías solo el código correspondiente a las funciones utilizadas, añadiéndolo al programa ejecutable. Una vez obtenido el programa ejecutable no será necesaria la presencia de las librerías para que el programa funcione. El inconveniente es que se generan programas de gran tamaño.

Librerías de enlace dinámico / compartidas

Son ficheros con código ejecutable que mantienen su independencia física del programa principal. La extracción del código correspondiente a una función llamada por el programa principal se realiza en tiempo de ejecución. Es imprescindible la presencia de estas librerías para que el programa funcione adecuadamente. La utilidad principal es evitar programas ejecutables de tamaño excesivo. Por otra parte este tipo de librerías pueden ser compartidas por varios programas simultáneamente sin necesidad de cargar su código varias veces en memoria.

En la mayoría de los lenguajes modernos que permiten la programación modular, en un módulo se pueden incluir varios subprogramas que realizan funciones relacionas, Ej.- Entrada/ Salida, Calculo matemático, presentación gráfica de datos, comunicaciones, acceso a Bases de Datos. El mismo concepto de módulo se denomina algunos lenguajes como Clase, Paquete o Librería.

ESTRUCTURA DE SUBPROGRAMA

Un subprograma es un fragmento un programa que realiza una tarea concreta y que recibe un nombre por el que puede ser llamada o activada desde otra parte del programa. Son herramientas básicas que hacen posible la programación modular y siendo prácticamente indispensables en la programación profesional. Al concepto de subprograma se le denomina también subrutina, función, procedimiento o método.

Ventajas del uso de subprogramas:

- Evita la duplicación de grupos de instrucciones en diferentes partes del algoritmo.
- Facilita la construcción y compresión de los algoritmos, ya que dividimos el algoritmo en varias partes lo que permite la programación modular
- Cada subprograma es más fácil de describir que el algoritmo completo
- Facilita la reutilización de código, la portabilidad y el trabajo en grupo.

Una subprograma no suele utilizar las misma variables que el programa principal sino que define las sus suyas propias. Para comunicarse con el programa principal se utilizan una serie de variables de comunicación denominadas argumentos o parámetros, que permiten el paso de información entre el programa principal y el subprograma.

```
SUBPROGRAMA Nombre ( TipoVariable Variable1 (Uso), Tipo Variable Variable2 (Uso) )
CONSTANTES
   Tipo Constante Nombre = valor
   ......
VARIABLES:
   Tipo Variable Nombre,
   ......

INICIO
  ....
  ...
FIN-SUBPROGRAMA
```

Entorno de un subprograma

- Parámetros o argumentos del subprograma
  Tipos:
  - Entrada: Cuando el valor de la variable no se modifica durante la ejecución del subprograma
  - Salida: Cuando el valor de la variable se modifica sin utilizar su valor inicial.
  - Entrada/ Salida: Cuando se utiliza el valor inicial de la variable y asignando un nuevo valor.
- Variables y constantes propias del subprograma : A parte de los parámetros un procedimiento puede tener su propias variables (**variables locales**) distintas de las del programa principal.

*En algunos casos, aunque generalmente no es un buen método de programación, un subprograma puede acceder a las variables del programa principal (**variables globales**) para realizar algún tipo de tarea, sin embargo casi siempre es preferible acceder a dichos valores mediante el paso de parámetros.*

Invocación o llamada de un subprograma.

Para que un subprograma se ejecute debe ser llamado por el programa principal o por otro subprograma, esto se realiza como otra instrucción cualquiera, indicando que valores o variables queremos incluir en su parámetros.

```
INICIO
...
 NombreSubprograma ( Parámetros )
..
FIN
```

Ejemplo:

```
SUBPROGRAMA  CalcularSuma (Entero ValorUno  (E) ,Entero ValorDos (E) ,Entero Resultado  (S) )
INICIO
  Resultado = ValorUno + ValorDos
FIN-SUBPROGRAMA

PROGRAMA  HacerAlgo
Variables
   Entero Dato, Valorfinal
  ....
INICIO
...
CalcularSuma ( Dato, 123, Valorfinal);
....
FIN
```

PARÁMETROS DE UN SUBPROGRAMA

Parámetros formales: Son los parámetros tal como se declaran en la definición del subprograma:
→*(Entero ValorUno (E) ,Entero ValorDos ,Entero Resultado (S))*

Parámetros reales: Son los parámetros tal como se rellenan en la llamada al subprograma:
→*(Dato, 123, Valorfinal);*

Para que una llamada a un subprograma sea correcta, deben coincidir el tipo y el número de los parámetros reales con el tipo y número de los parámetros formales. *Ej.- Tres parámetros de tipo entero.*

Cuando un parámetro formal es de entrada, el parámetro real puede ser una variable, un valor o expresión del mismo tipo que el definido en el parámetro formal. Cuando una parámetro formal es de salida o entrada-salida, el parámetro real forzosamente tiene que ser una variable del mismo tipo que el parámetros formal, donde el subprograma pueda dejar los resultados. *Ej.- El primer y segundo parámetros pueden ser variables (Dato) o valores(123) pero el tercer parámetro al ser de salida, debe ser forzosamente una variable definida en el programa principal(Valorfinal).*

La realización de llamada a un subprograma se realiza gracias una estructura de control del procesador denominada **pila de ejecución**, donde se guarda la dirección de retorno al programa principal y los parámetros reales de la función. El paso de los parámetros entre el programa principal y el subprograma también se realiza mediante esta pila. Cuando pasamos el valor de una variable o expresión se dice que es **un parámetro por valor o copia.** Cuando pasamos la dirección de una variable se dice que es un **parámetro por referencia**. Cuando trabajamos con parámetros de salida o entrada y salida el paso siempre debe ser por referencia. Para poder modificar el valor de la variable del programa principal.

Tipos de subprogramas:

- Un **procedimiento** es un subprograma que realiza operaciones sobre los parámetros que pueden ser de entrada, de salida o de entrada-salida.

- Una **función** es un subprograma que devuelve un único valor del resultado a partir de operaciones con los parámetros que en principio serán exclusivamente de entrada. Una función incluye una instrucción especial denominada RETURN (Retorno) donde se devuelve al programa principal el valor calculado por la función.

```
 FUNCION  CalcularSuma ( Entero ValorUno(E), Entero ValorDos(E) ) RESULTADO  Entero
Variables
   Entero Resultadosuma
Inicio
  Resultadosuma = ValorUno + ValorDos
  RETURN  Resultadosuma
Fin

...
Valorfinal = CalcularSuma ( Dato, 123 )

....


FUNCION  CalcularProducto  ( Entero N1 (E), Entero N2 (E) )  RESULTADO Entero
Variables
  Entero Contador, Acumulador
Inicio
  Contador    = 0
  Acumulador = 0
  PARA Contador = 1  HASTA  N2  INCREMENTO 1
     Acumulador =  Acumulador + N1
  FINPARA
  RETURN  Acumulador
Fin
```

Ejemplo de definición de funciones:

```
PROCEDIMIENTO  VerTablaDeMultiplicar  ( Entero Numero(E) )
FUNCION  CalcularProducto ( Entero N1 (E), Entero N2 (E) )  RESULTADO Entero
FUNCION  PrimerDivisor  ( Entero N1 (E) ) RESULTADO  Entero
PROCEDIMIENTO  ConvertirAMayusculas  ( Carácter Letra (E/S) )
FUNCION  HayMasDatos  ( TipoAlmacen  AlmacenPrincipal ) RESULTADO Lógico
```

En el lenguaje C, no hay grandes diferencias entre funciones y procedimientos, es el programador quien determina como va a comportarse el subprograma.

EJEMPLO DE PROGRAMACIÓN MODULAR:

Elaborar un programa que calcule el total de una factura según las siguientes reglas

- Primero se leerán el precio de tres artículos : Precio A, Precio B, Precio C, Si alguno precio es menor o igual a cero se solicitará de nuevo al usuario.
- Se leerán las unidades compradas de cada uno de los artículos: unidades A, unidades B, unidades C, las unidades deben ser mayor o igual a cero.
- Se Muestra el importe de unidades x precio salvo sin las unidades son igual a 0
- Se Muestra el importe Bruto
- Se aplica un descuento máximo del 10% si importe bruto supera las 10.000 o si adquiere como mínimo un artículo de cada tipo .
- Se Muestra el importe del descuento si existe
- Se Muestra el importe de la factura
- Se Muestra el importe final de la factura aplicando un 14% de IVA

Ejemplo de Ejecución:

```
Precio A: 1200
Precio B: 0
Precio B: 500
Precio C: 1000
Unidades de A: 2
Unidades de B: 0
Unidades de C:10

FACTURA.
Unidades de A     2  x  1200    ..... 2.400
Unidades de  C   10 x  1000    .....10.000
  Importe Bruto                  12.400
  Descuento 10%                   1.240
  Importe  Total                 11.160
  IVA  14 %                       1.562
  IMPORTE TOTAL  FACTURA          12.722 Pts
```

RESOLUCIÓN:

-------- PRIMERA APROXIMACIÓN ----------

Con recursos abstractos

```
PROGRAMA Calcular factura
ENTORNO
  Lista de Precios
  Lista de Artículos
  Importes
INICIO
  LeerPrecios
  LeerUnidades
  Mostrar Precios x Unidades()
  Mostrar Importes Descuentos y Iva
FIN
```

-------- PROGRAMA PRINCIPAL ----------

```
PROGRAMA Mostrar la factura
ENTORNO
  Entero UnidadesA, UnidadesB, UnidadesC
  Entero PrecioA, PrecioB, PrecioC
  Entero Importe, Descuento, IVA

INICIO
  LeerPrecio(PrecioA,'A');
  LeerPrecio(PrecioB,'B');
  LeerPrecio(PrecioC,'C');
  LeerUnidades(UnidadesA,'A');
  LeerUnidades(UnidadesB,'B');
  LeerUnidades(UnidadesC,'C');
  MostrarPxU(PrecioA,UnidadesA,'A');
  MostrarPxU(PrecioB,UnidadesB,'B');
  MostrarPxU(PrecioC,UnidadesC,'C');
  CalcularMostrarImportes()
FIN-PROGRAMA
```

```
-------- SUBPROGRAMAS ----------------------

PROCEDIMIENTO LeerPrecio (Entero Precio (S) , carácter Tipo (E))
INICIO
  Mostrar "Introduzca el precio para el articulo ",Tipo
  Leer Precio
  MIENTRAS (Precio <= 0 )
     Mostrar "Precio debe ser mayor que cero "
     Leer Precio
  FIN-MIENTRAS
FIN-LeerPrecio

PROCEDIMIENTO LeerUnidades ( Entero Unidades (S), carácter Tipo (E))
INICIO
  Mostrar "Introduzca el unidades para el articulo ",Tipo
  Leer Unidades
  MIENTRAS Unidades < 0 )
     Mostrar "La unidades no pueden ser negativas"
     Leer Unidades
  FIN-MIENTRAS
FIN-LeerUnidades

PROCEDIMIENTO  MostrarUxP ( Entero Precio(E), Entero Unidades(E), carácter Tipo(E) )
INICIO
   SI ( Unidades > 0 )
      ENTONCES
        Mostrar "Unidades de", Tipo,Unidades,'X',Precio, '....', Unidades * Precio
   FINSI
FIN-MostarUxP

PROCEDIMIENTO CalcularMostrarImportes ( )
INICIO
  Importe = (PrecioA*UnidadesA)+ (PrecioB*UnidadesB) + (PrecioC*UnidadesC)
  Mostrar  "Importe Bruto ", Importe
  SI ( Importe >= 10000) OR
        ( (UnidadesA >0 ) AND (UnidadesB > 0) AND (UnidadesC > 0) )
     ENTONCES
       Descuento = Importe * 0.10
       Mostar " Descuento 10 %  ", Descuento
       Importe = Importe - Descuento
       Mostar " Importe Total  ", Importe
  FIN-SI
  IVA = Importe * 0.14
  Mostrar " IVA  14 % ", IVA
  Importe = Importe + IVA
  Mostrar " IMPORTE TOTAL FACTURA ", Importe
FIN-CalcularMostrarImportes
```

Ejercicios de modularidad:

1.- Escribir un programa con diseño modular que utilice un procedimiento que intercambia los valores en memoria de dos variables numéricas reales.

2.- Escribir un programa con diseño modular que introduce por teclado dos números y presenta una serie de opciones correspondientes a operaciones aritméticas a realizar con los mismos (suma, resta, multiplicación, división y división). En función a la opción elegida se llama a la función correspondiente que retorna el resultado de dicha operación para que la visualice el programa principal. Una vez realizada la primera operación se le pregunta al usuario si quiere seguir realizando mas operaciones. En caso afirmativo se le volverá a visualizar el menú con las operaciones y en caso negativo se finalizará el programa.

3.- Escribir una función booleana llamada Digito que determine si un carácter es uno de los dígitos del 0 al 9.

4.-Escribir una función que calcule y retorne el número de días que faltan hasta final de año, a partir de una determinada fecha (dia mes año) y teniendo en cuenta si el año es bisiesto.
(NOTA: un año es bisiesto, si es múltiplo de 4, excepto aquellos que siéndolo, sean múltiplos de 100 pero no de 400. Ej. 2000 es bisiesto, 1900 no lo es, ya que aunque es múltiplo de 4, lo es de 100 pero no de 400)

5.- Diseñar un programa modular que determine los números menores que N, que sean primos y se puedan expresar como suma de otros dos números primos inferiores.

6.- Dos números son amigos si cada uno de ellos es igual a la suma de los divisores del otro (no consideramos divisor el propio número). Como por ejemplo el 220 y el 284 (que son amigos):
284 => divisores 1 + 2 + 4 + 71 + 142 = 220
220 => divisores 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
Se introduce un nº por teclado. Obtener los números amigos inferiores a él.

Recursividad

Cuando un procedimiento o función se llama o se invoca a si mismo para la realización de alguna tarea se dice que es recursivo. La recursividad es muy útil para tratar con estructura recursivas (Ej-árboles) o con determinados problemas como aquellos que necesitan aplicar técnicas de backtracking (vuelta a tras).

En todo subprograma recursivo debe existir una condición que provoque la terminación de la misma para que no exista una recursividad infinita.

Ejemplos:

1.- Cálculo del factorial mediante una función recursiva

N! = N * N-1 * N-2 …. 1 En forma recursiva: N! = N * (N-1)! Si N = 0 el factorial es 1	FUNCION Factorial (Entero Numero (E)) RESULTADO Entero INICIO SI (Numero = 0) RETURN 1 SI NO RETURN Numero * Factorial (Numero – 1) FINSI FIN

2.- Cálculo del producto mediante sumas utilizando la recursividad

```
FUNCION Producto ( Entero N1 (E),  Entero N2 ) RESULTADO  Entero
INICIO
  SI  ( N2 = 0 )
     RETURN  0
  SINO
     RETURN  N1 +   Producto ( N1, N2 – 1 )
  FINSI
FIN
```

Las funciones recursivas consumen memoria del ordenador (Pila) cada vez que se llaman a si mismas, siendo generalmente más lentas de ejecución que una versión iterativa, sin embargo, algunas veces ofrecen la solución más sencilla y rápida a un problema.
Existen lenguajes que sólo trabajan con recursividad para realizar los ciclos: Lenguajes de Inteligencia artificial como Lisp o PROLOG.

Ejercicios:

1.- Elaborar una función recursiva que datos números N y M, ambos enteros, me calcule la potencia de N elevado a M.

```
FUNCION Potencia  ( Entero N, Entero N ) Devuelve Entero
INICIO
  SI ( M = 0)
     RETURN 1
  SINO
     RETURN  N * Potencia  (N, M-1)
  FIN-SI
FIN-FUNCION
```

2.- Programa una función recursiva que permita calcular el elemento N de la serie de Fibonacci:
Valores de la serie:

1, 1 ,2 ,3 ,5 ,8 ,13 ,21 ,34 ,…

Fib(n) = Fib(n-1) + Fib(n-2). El elemento n es la suma de los dos anteriores:
Fib(1) = 0 y Fib(0) = 1 Los dos primeros elementos de la serie tienen el valor 1.

```
FUNCION Fibonacci  ( Entero N) Devuelve Entero
INICIO
  SI (( N = 0) OR ( N = 1 ))
     RETURN 1
  SINO
     RETURN  Fibonacci(N-1) + Fibonacci(N-2)
  FIN-SI
FIN-FUNCION
```

Ejemplo de algoritmo de vuelta a tras.

Teseo, el héroe griego, acaba de vencer al minotauro. Para salir del laberinto tiene que aplicar algún algoritmo que le permita buscar la salida. Gracias a Ariadna, hija del rey Minos y enamorada de nuestro héroe, Teseo dispone de un hilo que le sirve para marcar los lugares que ha visitado. Esto le evitará perderse.

El laberinto es una tabla de N x N posiciones con los siguientes valores:

- Muro: Lugar donde no podemos avanzar.
- Libre: Lugar donde podemos movernos
- Salida: La salida del laberinto
- Visitado: Lugar libre que ya hemos visitado

```
FUNCION  HaySalida ( Entero x, Entero y ) Devuelve Lógico
INICIO
  SI ( Laberinto [x , y ] = Salida )
        Return CIERTO
  SINO
     SI  ( Laberinto [ x, y ] = Muro )  OR ( Laberinto[ x, y ] = Visitado )
          Return Falso
     SINO
          Laberinto [ x, y ] = Visitado
          Return  Haysalida ( x +1, y )  OR  Haysalida ( x -1, y )
                  Haysalida ( x, y +1 )  OR  Haysalida (x, y -1 )
     FIN-SI
  FIN-SI
FIN-FUNCION
```

Explicación:

Si estoy en la salida devuelvo verdadero, si me encuentro con un muro o una posición que ya he visitado devuelvo falso, sino, estoy en una posición libre, la marco como visitada y busco si hay salida al norte, sur, este y oeste de la posición actual.

TABLAS DE DECISIÓN

Las tablas de decisión son una herramienta de diseño que permite expresar de forma detallada como a partir de un conjunto complejo de condiciones se toman las distintas acciones. Aunque es una herramienta relativamente antigua, ya que surge a inicios de la década de los sesenta, mantiene su utilidad siendo un método muy eficaz a la hora de solucionar determinados problemas.

Estructura

Matriz de condiciones, Entrada de condiciones
Matriz de acciones, Entrada de acciones

Condición 1	S	S	S	S	N	N	N	N
Condición 2	S	S	N	N	S	S	N	N
Condición 3	S	N	S	N	S	N	S	N
Acción 1	x							
Acción 2		x						
Acción 3		x	x	x	x	x		
Acción N							x	x

Una vez construida la tabla de verdad completa, existen una serie de métodos para simplificarla que consisten en eliminar aquellas condiciones que son indiferentes a la hora de realizar una acción. Los más utilizados son la llamada regla del paraguas y los mapas de Karnaugh.

Una vez simplificada, podemos construir el programa en base a una serie de sentencias condicionales encadenadas. Para obtener el algoritmo más eficaz, se comenzará a evaluar por la condición que tiene menos indifencias, es decir que es más determinante a la hora de elegir las acciones a realizar.

Consultar ejemplos y ejercicios.