Tema 2: Programas

2.1. Concepto de programa.-

Un programa puede definirse como un conjunto de ordenes que transforman unos datos de entrada para obtener una información de salida comprensible, formalizada y adecuada a las necesidades del usuario, usando para tal cometido un determinado algoritmo que describa las ordenes a realizar.

En base a esta definición de programa, se puede deducir que estos están constituidos principalmente por tres bloques:

- Entrada de datos.
- Algoritmo de resolución.
- Salida de resultados.

Por tanto conceptualmente un programa se puede representar como una caja negra. La caja negra o algoritmo de resolución será el encargado de transformar los datos de entrada en los de salida según unas especificaciones previas que deben ser conocidas antes de su implementación.

Las entradas se deben establecer de donde provienen y en que momento del programa se requieren. En algunos casos el proceso será interactivo, es decir provienen del usuario mientras el programa se está ejecutando, en otros casos pueden proceder de ficheros o de la memoria principal, en definitiva en la mayoría de los casos provendrán de los periféricos de entrada. Por tanto el programador deberá realizar una lista de las entradas así como la fuente de cada entrada y su formato. De esta forma se preverán todas las posibles situaciones y se evitará que los datos sean solicitados sin que se hayan introducido previamente en la memoria principal.

Por otra parte se deben listar todas las salidas del programa, así como el formato requerido por el usuario, es decir cual será el dispositivo de salida (periférico) y la forma de presentar dicha información. La respuesta a estas preguntas permitirá codificar correctamente las ordenes de salida y la elección adecuada del dispositivo periférico de salida: pantalla, impresora, disco, etc.

Una vez que las entradas y salidas han sido determinadas, se debe decidir como obtener las salidas deseadas a partir de las entradas dadas. Para ello el algoritmo de resolución constará una etapa de diseño del modelo de resolución del problema, donde se establece el modelo preciso para la resolución del mismo, teniéndose en cuenta el análisis hecho anteriormente. Una segunda etapa que comprenderá el estudio y desarrollo del algoritmo para producir el código correspondiente en el lenguaje de programación elegido. Aunque previamente a esta codificación se suele expresar previamente el algoritmo en alguna de la representación siguiente:

Pseudocódigo: Es una técnica para expresar en lenguaje natural la lógica de un programa, es decir, su flujo de control. En definitiva el pseudocódigo no es un lenguaje de programación, sino un modo de plantear un proceso de forma que su traducción a un lenguaje de programación sea sencilla. Durante todo este capítulo se irán introduciendo las palabras que se usarán en la realización de los pseudocódigos para la resolución de los problemas.

2.2. Elementos de un programa.-

Los programas están constituidos además de por los elementos descritos en el capítulo anterior, tales como constantes, variables, operadores y expresiones por una serie de ordenes que recogen dichos elementos. Estas son conocidas con el nombre de sentencias o instrucciones. Se pueden clasificar básicamente en 5 grupos:

- **Sentencias de entrada/salida**: Permiten establecer la comunicación desde el exterior (periféricos) a la memoria central del ordenador. El proceso es semejante al realizado por el ser humano cuando lee o escribe. En la lectura se capta a través de los ojos el contenido del libro y se grava en la memoria, mientras que en la escritura se pasan los conocimientos almacenados en la memoria a un soporte como puede ser una hoja de papel. Ejemplos de estas instrucciones podría ser:

LEER A	Extraer de los periféricos de entrada (teclado, disco, etc.) un valor y almacenarlo en la posición de memoria identificada por la variable A.
ESCRIBIR A	Poner en el dispositivo de salida (pantalla, impresora, etc.) el valor de la posición de memoria referenciada por la variable A.

- **Sentencias de asignación:** Permiten asignar valores a las variables. Así por ejemplo sentencias como:

```
A = 34
B = 12 + 45
```

  asignan el valor que se encuentra a la derecha del igual a la variable de la izquierda. El lado derecho de la sentencia puede ser cualquier expresión válida, mientras que el lado derecho debe ser el nombre de una variable y no una expresión. Por tanto no debe confundirse estas sentencias de asignación con ecuaciones matemáticas.

- **Sentencias de control**: Permiten romper la secuencia de ordenes del programa. Estas se basan en un test o comparación de una expresión lógica. Dichas bifurcaciones producidas por este tipo de sentencias consisten bien en un salto hacia adelante de la secuencia normal del

programa o hacia atrás. En cuanto a las condiciones a cumplir pueden ser incondicionales, es decir se realizan siempre que se ejecute la sentencia o condicionales, se realizan sólo al cumplirse un determinado proceso.

Estas sentencias constituyen la estructura básica o esqueleto de los programas, por tanto serán desarrolladas con mayor profundidad en el siguiente apartado.

- **Sentencias de declaración de variables:** Permiten reservar zonas de memoria para alojar los datos al mismo tiempo que definen el tipo de información a almacenar, quedando por tanto la variable preparada para su uso en el programa. Debe tenerse presente que no puede usar una variable sin su declaración previa. Por ejemplo la siguiente sentencia:

  ENTERO A

  Reserva un espacio de memoria para almacenar la variable B. Quedando esta definida como tipo entero. Por tanto si se intentara hacer una asignación como B = 1.2; ésta tomaría el valor 1, es decir la parte entera del valor 1.2.

- **Procedimientos**: Conjunto de sentencias que constituyen una unidad propia por si mismas, definidas para realizar una parte del programa. Están pueden ser definidas por el programador y bien pueden formar parte del sistema de programación. En ambos casos pueden admitir parámetros de entrada y devolver un parámetro de salida. Algunos ejemplos de funciones o procedimientos pueden ser:

  FUNCION SUMA (A, B)
  FUNCION AREA (BASE, ALTURA)

2.3. Sentencias de control.-

Los datos que se van a manejar, ya se ha visto que se pueden combinar para formar estructuras de datos, que son definiciones formales formando parte del programa. Ahora bien, para controlar las operaciones o acciones del programa se necesitan unas determinadas estructuras de control.

La estructuración de un programa, desde su punto de vista algorítmico, consiste en la combinación de unas operaciones fundamentales para crear otras más complejas. La programación estructurada utiliza únicamente tres sentencias básicas o estructuras de control:

- **Secuencia**: consiste en abstraer una sentencia en dos o más que se ejecutan una tras otra.
-
- **Selección o alternativa**: puede realizar una determinada sentencia entre varias posibles según el valor de una condición.

-
- **Iteración**: consiste en la repetición de otra sentencia o secuencia de sentencias un determinado número de veces.

Otro tipo de sentencias de control, que no deben usar en programación estructurada por razones que se verán en el siguiente capítulo son las sentencias **Incondicionales** (GOTO), las cuales efectúan una transferencia incondicional del control a otra parte del programa. La estructura de estas sentencias es:

GOTO <etiqueta>

2.3.1. Secuencia de sentencias.

La secuencia lógica es el elemento fundamental de todo programa. Una estructura secuencial está constituida por un conjunto ordenado de secuencias ejecutadas todas ellas el mismo número de veces en el mismo lugar del programa. La ejecución de éstas es de forma lineal.

Ejemplos de secuencias podrían ser acciones tales como:

```
AREA =  (BASE*ALTURA)/2
ESCRIBIR_RESULTADO SUMA
S  =  PI*SQR(RADIO)
LEER DATO_A
```

Cualquiera de estas operaciones es considerada como una sentencia; nótese que algunas de ellas pueden subdividirse en acciones más sencillas, esto es, necesitan varias sentencias para su realización. Del mismo modo, podemos agrupar varias sentencias en una única secuencia de sentencias.

Supongamos las dos sentencias simples: LEER DATO_A , LEER DATO_B. Deben ejecutarse una detrás de la otra, como se representa en la ilustración 2.a. Puesto que ambas son secuenciales y se ejecutan en sucesión, pueden agruparse en una única secuencia lógica, por ejemplo LEER DATOS, que internamente reúne a dos en el orden adecuado.

2.3.2 Sentencia selectiva.-

Este tipo de elemento determina una bifurcación en el programa. Dependiendo de cierta condición, puede ejecutarse una sentencia o secuencia entre varias posibles. Se denominará sentencia **SI...ENTONCES...SINO...** y la notación pseudocódigo que se va a utilizar es la siguiente:

SI <condición>
    ENTONCES <sentencia A>
SINO <sentencia B>
FINSI

Expresa que en caso de que se cumpla la condición se ejecutará la sentencia A y

si no se ejecutará B.

A veces no tiene que ejecutarse nada si no se cumple la condición con lo que la rama *SINO* sobraría y se tendría una modificación denominada instrucción **SI...ENTONCES**.

SI <condición>
    ENTONCES <sentencia A>
FINSI

Las sentencias selectivas pueden anidarse para formar instrucciones *SI..ENTONCES..SINO* con varias condiciones:

SI <condición A>
    ENTONCES <sentencia A>
SINO
    SI <condición B>
        ENTONCES <sentencia B>
    SINO
        SI <condición C>
            ENTONCES <sentencia C>
        ...
...

En los *SI...ENTONCES...* anidados debe tenerse en cuenta que la sentencia correspondiente a la rama de *SINO* es otra sentencia *SI...ENTONCES...* y así sucesivamente tantas veces sea necesario.

Existe otra situación en la que la condición no adquiere un único valor de verdadero (V) o falso (F) sino que puede adquirir varios. Por ejemplo, la condición de que una variable *COLOR* sea igual a un determinado valor que puede ser:

¿COLOR == 'R'? (Rojo) ó bien
¿COLOR == 'V'? (Verde) ó bien
¿COLOR == 'A'? (Azul) ó bien ....

En esta situación se ejecutará una sentencia determinada para cada posibilidad de la condición. A este tipo de sentencia se denomina **selección múltiple** o **alternativa múltiple**, se denominará **CASO** y se expresa como sigue:

CASO
    SI <condición> == <valor1>
        ENTONCES <sentencia 1>
    SI <condición> == <valor2>
        ENTONCES <sentencia 2>
    ....

```
    SI <condición> == <valorN>:
        ENTONCES <sentencia N>
    SINO
        <sentencia N+1>
FINCASO
```

Puede observarse como se consideran varias ramas de acción, una por cada valor posible de la condición, pero además se añade una rama más al final para el caso de que la condición no adquiera ninguno de los valores especificados. En la figura 5 se representa gráficamente la sentencia **CASO**. Cualquier sentencia de este tipo tiene una *SI...SINO* anidada equivalente.

Un ejemplo de sentencia alternativa simple podría ser el siguiente programa que determina el máximo de dos números visualizando el resultado por pantalla. La estructura del programa sería:

```
PROGRAMA_MAX2N
    LEER NUM1
    LEER NUM2
    SI (NUM1 > NUM2)
     ENTONCES  MAXIMO = NUM1
    SINO  MAXIMO = NUM2
    FINSI
    VISUALIZAR MAXIMO
FIN_MAX2N
```

Este programa en un principio lee los datos, (se podrían haber unido las dos sentencias en una sola 'LEER DATOS') después se pregunta por la condición, si el primer número leído es mayor que el segundo es evidente que el máximo es *NUM1*, con lo que se asigna su valor a una variable temporal *MAXIMO*, si no se cumple la condición, se le asigna el otro dato leído. Finalmente escribe el número que contiene la variable *MAXIMO* que será el resultado. En la representación gráfica o diagrama de flujo de control, tan sólo se han utilizado las dos estructuras hasta ahora estudiadas.

Un ejemplo de estructura alternativa múltiple (CASO) podría ser el siguiente un programa que visualice un saludo según la hora del día que se le introduzca. Así, a partir de las 6 horas y hasta las 12h. responderá '*Buenos días*', a partir de las 12h. y hasta las 20h. responderá '*Buenas tardes*' y entre 20h y 6h '*Buenas noches*'.

```
PROGRAMA_SALUDO
    LEER HORA
    CASO
        SI (HORA > 6) Y (HORA <= 12)
         ENTONCES VISUALIZAR 'Buenos días'
```

```
            SI (HORA > 12) Y (HORA <= 20)
             ENTONCES VISUALIZAR 'Buenas tardes'
            SI ((HORA > 20) Y (HORA < 24)) O ((HORA >= 0) Y (HORA <=6))
             ENTONCES VISUALIZAR 'Buenas noches'
            SINO VISUALIZAR 'La hora introducida no es correcta'
        FINCASO
FIN_SALUDO
```

Este programa prevé la posibilidad de que el usuario introduzca una hora equivocada (menor que 0 ó mayor que 23). Nótese como en la tercera rama, la del saludo nocturno, se expresa una condición lógica formada por la unión de dos para tener en cuenta los dos intervalos de la noche, entre 20h y 23h y entre 0h. y 6h.

2.3.3. Setencia iterativa.-

Esta estructura de control expresa la repetición de una misma sentencia o secuencia de sentencias un número de veces que viene determinado por el cumplimiento o no de una condición.

La acción que se ejecuta en cada iteración será una secuencia de sentencias que, como se ha visto, puede contener varias sentencias más simples. Se trata, por tanto, de ejecutar una misma secuencia de sentencias varias veces hasta que la condición adquiera un determinado valor en cuyo caso finalizará del bucle.

Esta sentencia se va a expresar como **MIENTRAS...HACER** y su lógica funcional es
:

MIENTRAS <condición>
    HACER <sentencia>
FINMIENTRAS

El proceso expresa que mientras se cumpla la condición se ejecute la sentencia A hasta que deje de cumplirse. Naturalmente la sentencia a ejecutar debe manipular de algún modo el valor de la condición, ya que si no, éste siempre sería el mismo y se obtendría un bucle sin salida o infinito (hay veces en la programación que se utilizan bucles infinitos para ciertos algoritmos que se rompen mediante una acción externa).

Un Ejemplo de sentencia iterativa podría ser el programa que calcula la media aritmética de N números enteros. El primero de los valores leídos corresponde con el número N de valores sobre los que se calculará la media. Recuérdese que la media de N números num1, num2, ..., numN es el resultado viene dada por:

(num1 + num2 +...+ numN)/N.

```
PROGRAMA_MEDIA
        LEER N
        SUMA = 0
        CONTADOR = 1
        MIENTRAS (CONTADOR <= N)
        HACER
                LEER NUM
                SUMA = SUMA + NUM
                CONTADOR = CONTADOR + 1
        FINMIENTRAS
        VISUALIZAR (SUMA / N)
FIN_MEDIA
```

Se han utilizado una variable *SUMA* para acumular la suma de todos los números leídos, conocido como acumulador, y una variable *CONTADOR* para contar el número de elementos que se van a leer (desde 1 hasta N), conocida como contador. Para ver más claramente este ejemplo supóngase que los datos introducidos al programa son:

```
4
3
10
1
22 <fin_de_datos>
```

El programa leerá el 4 y ejecutará un bucle 4 veces hasta que (contador>4). En la tabla 1 se muestra el progreso del programa, para el caso de los valores anteriores. En cada iteración se debe incrementarse la variable *CONTADOR* para que la condición adquiera el valor de salida, al mismo tiempo que se va incrementado el acumulador *SUMA*.

2.4. Ejercicios resueltos.-

1º Realizar el algoritmo, expresando en pseudocódigo para resolver dado un número N, contestar si es positivo o negativo.

En primer lugar se describe el algoritmo que resuelve el problema en lenguaje natural, describiendo los pasos a realizar.

1º Leer el número N del teclado.
2º Comprobar si el número N es mayor que cero.
3º En afirmativo, sacar por pantalla el mensaje “El número N es positivo”.
4º En caso contrario, sacar por pantalla el mensaje “El numero N es negativo”.

El desarrollo del algoritmo en pseudocódigo sería:

```
PROGRAMA VALOR
LEER N
SI (N > 0)
        ENTONCES ESCRIBIR "N es positivo"
SINO
        ESCRIBIR "N es negativo"
FINSI
FIN VALOR
```

2º Realizar el algoritmo, expresando en pseudocódigo para resolver dados N números, contestar si son positivos o negativos.

En primer lugar se describe el algoritmo que resuelve el problema en lenguaje natural, describiendo los pasos a realizar.

1º Inicializar contador de números leídos a investigar (I) a cero.
2º Leer la cantidad de números a investigar (N)
3º Comprobar si el contador de números leídos a investigar es menor o igual a la cantidad de números a leer.
4º En caso afirmativo finalizar.
5º En caso contrario continuar con los pasos 6º y siguientes.
6º Leer el número a investigar (V).
7º Incrementar el contador de números leídos en uno.
8º Comprobar si el número leído V es mayor que cero.
9º En caso afirmativo escribir por pantalla el mensaje “El número V es positivo”.
10º En caso negativo escribir por pantalla el mensaje “El número V es negativo”.
11º Volver al paso 3º.

El desarrollo del algoritmo en pseudocódigo sería:

```
PROGRAMA VALOR_BUCLE
I = 0
LEER N
MIENTRAS (I <= N)
HACER
        LEER V
        I = I + 1
        SI (V > 0)
                ENTONCES ESCRIBIR "V es positivo"
        SINO
                ESCRIBIR "V es negativo"
        FINSI
FINMIENTRAS
FIN VALOR_BUCLE
```

3º Realizar el algoritmo, expresando en pseudocódigo para calcular la suma de los diez primeros números pares.

En primer lugar se describe el algoritmo que resuelve el problema en lenguaje natural, describiendo los pasos a realizar.

1º Inicializar a cero el generador de números pares (I).
2º Inicializar a cero el acumulador de suma (S).
3º Comprobar si el generador de números pares es menor o igual a 20.
4º En caso afirmativo saltar al paso 8º.
5º En caso negativo incrementar en dos el generador de números pares.
6º Sumar al acumulador S el valor del número par correspondiente (I).
7º Saltar al paso 3º.
8º Escribir el valor del acumulador S.

El desarrollo del algoritmo en pseudocódigo sería:

```
PROGRAMA SUMA_PARES
I = 0
S = 0
MIENTRAS ( I <= 20)
HACER
        I = I + 2
        S = S + I
FINMIENTRAS
ESCRIBIR S
FIN SUMA_PARES
```

4º Realizar el algoritmo, expresando en pseudocódigo para calcular el factorial de un número N.

La fórmula para calcular el factorial de un número N es:

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

En primer lugar se describe el algoritmo que resuelve el problema en lenguaje natural, describiendo los pasos a realizar.

1º Inicializar contador del número de multiplicaciones (CONT) a cero.
2º Inicializar el acumulador del factorial (FACT) a uno.
3º Leer el número N del cual se quiere calcular el factorial.
4º Comprobar si el contador CONT es menor que N.
5º En caso afirmativo saltar al paso 9º.

6º En caso contrario incrementar el contador en uno.
7º Multiplicar factorial (FACT) por el contador CONT acumulando el valor de nuevo en FACT.
8º Volver al paso 4º.
9º Escribir el valor de FACT.

El desarrollo del algoritmo en pseudocódigo sería:

```
PROGRAMA FACTORIAL
CONT = 0
FACT = 1
LEER N
MIENTRAS (N < CONT)
HACER
        CONT = CONT + 1
        FACT = FACT * CONT
FINMIENTRAS
ESCRIBIR FACT
FIN FACTORIAL
```

2.5. Ejercicios propuestos.-

1º Realizar el algoritmo, expresando en pseudocódigo para que dado un número N y un tanto por ciento, contestar:

- El número y el tanto por ciento.
- La cantidad obtenida con el tanto por ciento de descuento y la cantidad a pagar.

2º Realizar el algoritmo, expresando en pseudocódigo para que dado un número N, contestar si es múltiplo de 3, 6 y 9 a la vez. Esta operación se desea realizarla un número indefinido de veces.

3º Realizar el algoritmo para resolver una ecuación de segundo grado, teniendo en cuenta las soluciones imaginarias. Identificar las variables antes de usarlas.

4º Hacer un algoritmo que realice el horóscopo, es decir dada una fecha de nacimiento con formato de mes y día responda el signo del horóscopo correspondiente.

5º Se va realizar un sondeo para las votaciones, de forma que salga un mensaje de vota o no vota, dependiendo de que se tenga dieciocho años o no, contando los que votan. Pidiendo la fecha de nacimiento y la fecha de votación realizar el algoritmo que resuelva dicha situación.

6º A través del ordenador se desea realizar la siguiente encuesta:
- Personas que tienen o no televisión.
- Personas que tienen televisión con WIDI.
– Personas que piensan comprarse televisor.

Realizar un algoritmo que calcule el tanto por ciento de los tres grupos.

7º Diseñar el algoritmo (en pseudocódigo) de un programa que:

1º) Pida por teclado un número (dato entero).
2º) Muestre por pantalla los mensajes:

Ha introducido <cantidad_de_números> número(s)
La suma es <suma>

3º) Pregunte al usuario si desea introducir otro o no.
4º) Repita los pasos 1º, 2º y 3º, mientras que, el usuario no responda 'n' de (no).
5º) Muestre por pantalla la media aritmética (dato real) de los números

introducidos.

8º Diseñar el algoritmo (en pseudocódigo) de un programa que:

1º) Pida por teclado un número (dato entero).
2º) Repita el paso 1º, mientras que, el número introducido sea distinto de cero.
3º) Muestre cuántos números mayores que cero han sido introducidos por el usuario, así como, la suma de todos ellos.

9º Diseñar el algoritmo (en pseudocódigo) de un programa que:

1º) Pida por teclado un número (dato entero).
2º) Repita el paso 1º de manera indefinida hasta que el usuario decida parar.
3º) Muestre cuántos números primos han sido introducidos por el usuario.

10º Diseñar el algoritmo (en pseudocódigo) de un programa que:

1º) Pida por teclado un número (dato entero).
2º) Muestre el número inverso del introducido. Así por ejemplo si el número introducido es 456, debe mostrar 654.

11º Diseñar el algoritmo (en pseudocódigo) de un programa que:

1º) Pida por teclado dos valores de tiempo (horas, minutos y segundos).
2º) Muestre el tiempo transcurrido entre ambos valores.

12º Diseñar el algoritmo (en pseudocódigo) de un programa que:

1º) Pida por teclado una cantidad de dinero (dato entero).
2º) Muestre dicha cantidad desglosada en monedas. Por ejemplo, si el usuario introduce 57 euros, el algoritmo debe mostrar 1 de 50 €, 1 de 5€ y 2 de 1€.