FUNDAMENTOS DE PROGRAMACIÓN

Tema 2

Conceptos básicos de Metodología de la Programación

1º Administración de Sistemas Informáticos
I.E.S. Francisco Romero Vargas
Departamento de Informática

1. OBJETOS DE UN PROGRAMA.

Un objeto es todo aquello que puede ser manipulado por una instrucción y que va a contener los datos que intervienen en la solución de un problema

Un objeto es cualquier dato o código que ocupa memoria direccionable. En C, los objetos incluyen variables, constantes, funciones, punteros, arrays, cadenas, estructuras y uniones.

Mediante ellos, en un programa podremos realizar el almacenamiento de los datos y de los resultados de las distintas operaciones que intervienen en la solución del problema.

Todo objeto tiene tres atributos:

- **Nombre o identificador:** Palabra creada por el programador, generalmente una cadena de letras, dígitos y guiones.
- **Tipo:** Conjunto de valores que puede tomar y operaciones que se permite realizar con ellos, así como la cantidad de memoria que utilizan.
- **Valor:** Elemento del tipo que se le asigna.

- **Constantes.**

Objeto cuyo valor permanece invariable a lo largo de la ejecución de un programa.

Una **constante literal** es aquella cuyo valor se escribe explícitamente en el programa fuente (por ejemplo: 1000, 3.14, "Introduzca datos: "). Una **constante con nombre** es la denominación de un valor concreto, de tal forma que en el programa fuente se utiliza su nombre cada vez que se necesita referenciarlo. Todas las constantes tienen asociado un tipo implícitamente.

- **Variables.**

Son objetos cuyo valor puede ser modificado a lo largo de la ejecución de un programa.

La memoria principal o central podemos entenderla como un conjunto de celdas elementales identificadas cada una de ellas por una dirección. Cada celda contiene un valor que puede ser leído y/o modificado o escrito, la escritura o modificación consiste en escribir un nuevo valor reemplazando al anterior. El hardware nos permite acceder a cada una de estas celdas. **El concepto de variable dentro de los lenguajes de programación es una abstracción del concepto de celda de memoria**, de tal forma que en vez de hacer referencia a las direcciones de las celdas donde están los datos,

haremos referencia al nombre de la variable asociado a esas celdas. Una variable se caracteriza por los siguientes atributos:

Nombre: hace referencia a la dirección de memoria donde se va a almacenar el valor de la variable, o dicho de otra forma al área de memoria ligada a la variable. Los nombres de las variables se llaman **identificadores**. Cada lenguaje de programación tiene sus propias reglas para formar identificadores. Lo normal es que los nombres que se elijan para las variables tengan relación con el objeto al que representan.

Tipo: la clase o conjunto de valores que se puede asociar a la variable junto con las operaciones que se pueden realizar sobre ella.

Ámbito o Alcance: conjunto de sentencias del programa en el que aparece la variable.

Tiempo de vida: intervalo de tiempo en el que el área de memoria está ligada a la variable.

Valor: la información almacenada en las posiciones de memoria asociadas o ligadas a la variable.

Ejemplo 1:

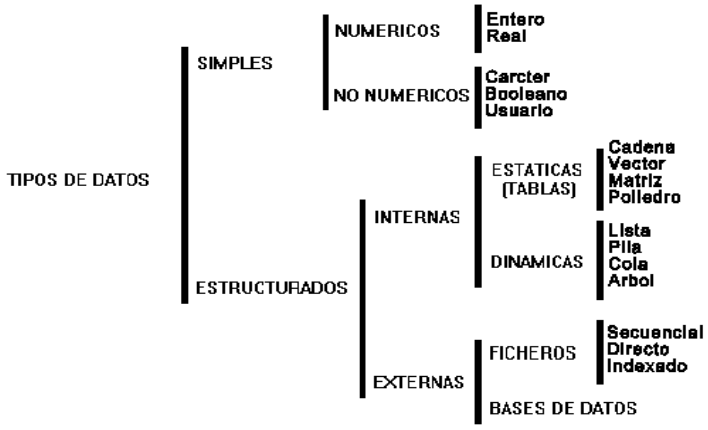
Escribir un algoritmo que calcule el área de un círculo (en metros cuadrados) y la longitud de la circunferencia (en metros) conociendo el radio expresado en milímetros.

```
Programa:    Circulo
Entorno:     Constantes
                    PI = 3.141592
             Variables
                    radio, longitud, area : real
Algoritmo:
             Escribir (“Introducir radio (en milímetros): “)
             Leer (radio)
             radio = radio / 1000
             longitud = 2 * PI * radio
             area = PI * radio * radio
             Escribir (“Área círculo (en metros cuadrados): “ , area)
             Escribir (“Longitud circunferencia (en metros): “, longitud)
```

2. TIPOS DE DATOS.

Dato es una expresión general que describe los objetos con los que opera un ordenador. Cada objeto lleva asociado un tipo de dato que determina el conjunto de valores que puede tomar.

El tipo de un objeto especifica la clase de valores o caracteres que pueden asignarse al objeto.



Existen diferentes clases o tipos de datos según las magnitudes o conceptos que se representan, y cada uno de ellos tienen distinta representación interna en la computadora. Por ejemplo, se representan internamente de forma diferente una cadena, un número entero o un número real.

El **rango de un tipo de datos numérico** es el conjunto formado por todas sus cantidades representables, y viene determinado por el formato de representación en memoria.

A continuación se estudiarán algunos de los tipos de datos simples.

- **Enteros.**

El tipo entero es una representación del conjunto de los números enteros y por tanto son datos numéricos. Se representan externamente en decimal, por dígitos (0, 1, 2, .. , 9) formando cantidades enteras (sin parte fraccionaria), positivas o negativas. Por ejemplo: 43, -114, 0, -99887, 1234567, etc.

La transformación realizada consiste en representar el número en binario y almacenarlo en un número fijo de bits **n** de modo que distintos valores de **n** dan origen a distintos tipos de enteros. Su rango viene determinado por el formato de su representación interna.

			RANGO Desde	Hasta
Enteros cortos	1 byte	sin signo (binario puro)	0	$2^8 - 1 = 255$
		con signo (complemento a 2)	$-2^7 = -128$	$2^7 - 1 = 127$
Enteros	2 bytes	sin signo (binario puro)	0	$2^{16} - 1 = 65535$
		con signo (complemento a 2)	$-2^{15} = -32768$	$2^{15} - 1 = 32767$
Enteros largos	4 bytes	sin signo (binario puro)	0	$2^{32} - 1 = 4*10^9$
		con signo (complemento a 2)	$-2^{31} = -2147*10^6$	$2^{31} - 1 = 2147*10^6 - 1$

Fuera del rango determinado por el sistema, los números enteros carecen de representación; es decir, el tipo entero representa un subrango o subconjunto de los números enteros (conjunto Z) de modo que no todos los enteros se pueden representar.

Ciertas operaciones no pueden realizarse sobre datos de tipo entero: por ejemplo, 0/0 y n/0 y también sumar dos cantidades que sobrepasen el límite impuesto por el rango.

Si se pretende utilizar, en un rango determinado, cualquier número que sobrepase su límite máximo, se obtiene un resultado erróneo por desbordamiento de memoria (overflow). Es decir, el resultado codificado es mayor que el tamaño de la memoria que se le ha asignado, según el tipo, y no puede ser almacenado en ella.

Siempre que sea posible, se deben utilizar números enteros mejor que números reales pues el procesador los manipula más rápidamente, sobre todo, si no tiene coprocesador matemático.

Cualquier operación con datos de tipo entero es exacta, salvo que se produzcan desbordamientos.

- **Reales.**

Es una representación del conjunto de los números reales. Esencialmente, la transformación realizada consiste en expresar el número en la forma:

$$N = m * B^{e}$$

donde **N** es el número real a representar, **B** es la base utilizada para el exponente (que está prefijada para una computadora dada), **e** es el exponente del número y **m** es la mantisa.

El número se almacena en la computadora yuxtaponiendo el signo, el exponente y la mantisa, cada uno con un número de bits prefijado.

Por tanto, el tipo real representa datos numéricos, llamados también de **coma o punto flotante**. Se representan externamente por dígitos formando cantidades positivas o negativas que pueden tener cifras fraccionarias. Ejemplo: 3.0, 0.000333, 11.234, 8897687.34, etc.

Como en el caso de los enteros, el rango de los mismos depende de su formato de almacenamiento en memoria. Así, en el lenguaje C, el rango de los números reales en simple precisión (32 bits) va desde 3.4E-38 hasta 3.4E+38, y en doble precisión (64 bits) varía desde 1.7E-308 y 1.7E+308. Si el número es mayor que el límite superior del rango, se produce un error por desbordamiento (overflow). Cuando el número es menor que el límite inferior del rango, de modo que la mantisa sólo contiene ceros, se produce un error por desbordamiento inferior (underflow). Es decir, se necesitaría mayor número de bits en la mantisa para que pudiera aparecer en ella algún dígito significativo.

La limitación del número de bits usados para representar la mantisa provoca una falta de precisión en la representación.

		RANGO	
		Desde	Hasta
Reales simple precisión	32 bits	3.4E-38	3.4E+38
Reales doble precisión	64 bits	1.7E-308	1.7E+308
Reales doble precisión "largos"	80 bits	3.4E-4932	1.1E+4932

Para los datos de tipo real están definidas las operaciones aritméticas. La suma y la multiplicación de datos de tipo real cumplen la propiedad conmutativa, pero no siempre la asociativa ni la distributiva. Esto sucede porque el orden en que se realizan las operaciones influye en el resultado. En cada operación se producen errores por falta de precisión en la representación (errores de redondeo), que se acumulan durante todo proceso de cálculo.

Ejemplo 2:

Supongamos un formato de representación con base B=10 y mantisa de 4 dígitos decimales. Hay que tener en cuenta que en los cálculos hay que ir igualando exponentes y despreciando a partir de la quinta cifra menos significativa.

$x = 5000 = 0.5000 \cdot 10^{4}$ $y = -4999 = -0.4999 \cdot 10^{4}$ $z = 1.500 = 0.1500 \cdot 10^{1}$

La operación **(z+x) + y** $= (0.5000 \cdot 10^{4} + 0.0001 \cdot 10^{4}) - 0.4999 \cdot 10^{4} =$
$0.5001 \cdot 10^{4} - 0.4999 \cdot 10^{4} = 0.0002 \cdot 10^{4} =$ **2.0000**

por la propiedad asociativa de la suma tendría que ser igual a

z + (x+y) $= 0.1500 \cdot 10^{1} + (0.5000 \cdot 10^{4} - 0.4999 \cdot 10^{4}) =$
$0.1500 \cdot 10^{1} + 0.1000 \cdot 10^{1} = 0.2500 \cdot 10^{1} =$ **2.500**

También se produce error cuando en una expresión se suma una cantidad grande, en comparación con el valor de la expresión, lo que hace que se pierdan los dígitos menos significativos, y posteriormente se resta dicha cantidad u otra del mismo orden de magnitud.

Ejemplo 3:

$(x+z) - x = (0.5000 \cdot 104 + 0.0001 \cdot 10^{4}) - 0.5000 \cdot 10^{4} = 0.5001 \cdot 10^{4} - 0.5000 \cdot 10^{4} = 0.0001 \cdot 10^{4} = 1$

que no equivale al valor de z.

- **Alfanuméricos.**

Son datos compuestos por una secuencia de caracteres (llamada cadena), que es interpretada por el sistema como un dato único, y que se almacenan en posiciones consecutivas de la memoria.

El conjunto de caracteres representado depende del código que utilice la computadora: Uno de los códigos más usuales es el ASCII (código estándar americano para intercambio de información) de 7 bits, extendido actualmente a 8 bits.

Hay que tener en cuenta que cuando en una expresión se comparan caracteres se hace con relación a su código, y en el ASCII presentan este orden:

```
... 0   1    ...  9 .... A    ....  Z  .... a   ..... z
  48  49      57     65      90        97      122
```

El espacio tiene el código 32.

Así, por ejemplo: ‘A’ < ‘a’ es cierto.

Cuando se comparan cadenas, se hace carácter a carácter comenzando por la izquierda hasta encontrar una desigualdad:

"JUANITO" > "JUAN DE DIOS"

Los valores alfanuméricos suelen escribirse entre comillas (cadenas en C) o apóstrofos (un solo carácter en C). Por ejemplo: “JUAN” , “3224” (que no es lo mismo que 3224 pues con el tipo de dato numérico se puede operar aritméticamente y con el alfanumérico no).

Cada carácter simple de tipo alfanumérico suele almacenarse en un byte u octeto.

Para delimitar internamente la cadena se utilizan diversos métodos...

- el lenguaje C termina las cadenas con un carácter fin de cadena: el carácter nulo ('\0').
- Turbo Pascal, almacena como primer carácter de la cadena la longitud de ésta.

- **Lógicos.**

Es un tipo de datos que sólo puede tener dos valores o estados: verdadero o falso.

Se almacena en memoria mediante un octeto de ceros (0000 0000) para el valor FALSO, y cualquier otro valor (por ejemplo, 0000 0001) para el valor CIERTO.

Se utilizan para elegir entre dos alternativas diversas o como resultado de una comparación entre objetos. Tienen distintas representaciones externas, según los lenguajes empleados: true y false, 1 y 0, etc.

- **Punteros.**

Es un tipo de datos que se utiliza para indicar la dirección de memoria de un objeto.

Al objeto al que se la ha asignado ese tipo de datos se le llama con el mismo nombre del tipo de datos al que pertenece, es decir, se le denomina también puntero.

3. EXPRESIONES.

Una expresión es la representación de un cálculo necesario para la obtención de un resultado, es decir, devuelve un valor como resultado de evaluarse.

Se define una expresión de la siguiente forma:

- Un valor es una expresión; dado que por sí misma devuelve un valor:
  1.25 "JUAN"

- Una constante o variable:
  a Edad_de_Juan IVA

- Una función:
  seno(Angulo) coseno(Alfa)

- Una combinación de valores, constantes, variables, funciones, operadores, paréntesis, cumpliendo unas determinadas reglas de formación:

  a+b-3+IVA cos(x)+a*6

Una expresión consta de operadores (símbolos que denotan las operaciones) y operandos (objetos sobre los que actúan los operadores) y toma un valor que será el resultado de la ejecución de las operaciones indicadas. Según el resultado que producen, las expresiones se clasifican en:

- **Aritméticas**.- Si el resultado es de tipo numérico.

- **Lógicas o booleanas**.- Si el resultado es de tipo lógico (**Verdadero** o **Falso**).

- **Carácter o alfanuméricas**.- Si el resultado es de tipo carácter.

• Operadores.

Para la construcción de expresiones se pueden utilizar los siguientes operadores:

Aritméticos

Realizan operaciones tal y como las conocemos de la aritmética tradicional:

+	suma
-	resta y menos unario
*	producto
/	división real
DIV	división entera
MOD	resto de una división entera
**	potencia

Relacionales

Permiten realizar comparaciones: < (menor que), > (mayor que), = (igual que), <= (menor o igual que), >= (mayor o igual que) y <> (distinto a).

Se pueden utilizar con cualquiera de los tipos estándar: enteros, reales, carácter y lógico.

Para realizar comparaciones de datos de tipo carácter o de cadena de caracteres hay que tener en cuenta el código normalizado que utilice el ordenador (ASCII). En estos códigos existe una secuencia de ordenación de todos los caracteres:

´0´<´1´<´2´....´9´...<´A´<´B´.....<´Z´..<´a´<´b´....<´z´...

Lógicos

Los operadores lógicos básicos son: **NOT, AND, OR.** El resultado de las operaciones lógicas viene determinado por las tablas de verdad correspondientes a cada una de ellas según la lógica de proposiciones:

A	B	A AND B	A OR B	NOT A
F	F	F	F	V
F	V	F	V	
V	F	F	V	F
V	V	V	V	

Alfanumérico o de carácter (No permitido en C)

Concatenación: +, realiza la unión de cadenas de caracteres.

“casa”+ “hola” = “casahola”

Si **matricula** es una variable que contiene “2342-H” entonces:

“MA”+'-'+ matrícula = “MA-2342-H”

- **Precedencia de operaciones**

Cuando en una expresión hay más de un operador, hay que tener en cuenta las reglas de prioridad o precedencia que permiten determinar el orden en que el ordenador va a realizar las operaciones.

1.º Paréntesis (comenzando por los más internos).
2.º Signo.
3.º Potencias (si existe este operador).
4.º Productos y divisiones.
5.º Sumas y restas.
6.º Concatenación (de cadenas de caracteres).
7.º Relacionales.
8.º Negación (NOT).
9.º Conjunción (AND).
10.º Disyunción (OR).

Para alterar este orden se usan los paréntesis que, como se puede ver, están en primer lugar en el orden de precedencias, de forma que toda expresión encerrada entre paréntesis se ejecuta primero. En caso de coincidir la prioridad de dos o más operadores, éstos se ejecutan de izquierda a derecha.

NOTA: Este orden de prioridad de ejecución de las operaciones varía dependiendo del lenguaje de programación utilizado. Cuando estudiemos el lenguaje de programación C se establecerá el orden exacto de prioridades atendiendo a los múltiples operadores que este lenguaje utiliza.

- **Conversión de fórmulas en expresiones aritméticas**

A la hora de diseñar algoritmos habrá que convertir las expresiones algebraicas en expresiones aritméticas algorítmicas:

$$\frac{A+B}{C} \quad \text{..... } (A+B)/C$$

$$\frac{1}{7} + \frac{XY(3+C)}{5} \quad \text{..... } 1/7+X*Y*(3+C)/5$$

4. PARTES PRINCIPALES DE UN PROGRAMA.

Un programa puede considerarse como una secuencia lógica de acciones (instrucciones) que manipulan un conjunto de objetos (datos) para obtener unos resultados que serán la solución al problema que resuelve dicho programa.

Todo programa, en general, contiene dos bloques bien diferenciados para la descripción de los dos aspectos anteriormente citados:

- **Bloque de declaraciones.** En él se especifican todos los objetos que utiliza un programa (constantes, variables, tablas, registros, ...) indicando sus características. Este bloque se encuentra localizado siempre por delante del comienzo de las acciones.

- **Bloque de instrucciones.** Constituido por el conjunto de operaciones que se han de realizar para la obtención de los resultados deseados.

El entorno de un programa u objetos del mismo es el conjunto de elementos capaces de almacenar unidades de información, necesarios para contener tanto los datos de entrada como los datos resultantes de todos los procesos que se lleven a cabo.

La ejecución de un programa consiste en la realización secuencial del conjunto de instrucciones de que se compone, desde la primera a la última, de una en una. Este orden de realización únicamente será alterado mediante sentencias de control.

Por lo general, los programas (sobre todo, los que usan proceso interactivo) contiene entrelazadas las operaciones básicas para el tratamiento de la información: entrada de datos, proceso o manipulación y salida de resultados. Si bien es cierto que, debido a la propia naturaleza de las instrucciones, las de entrada de datos se suelen encontrar al comienzo del programa, las de proceso se encuentran en medio y las de salida de resultados se encuentran hacia el final.

5. PSEUDOCÓDIGO.

La solución de un problema, para su ejecución por parte de una computadora, requiere el uso de una notación que sea entendida por ella, es decir, un lenguaje de programación. No obstante, durante la fase de diseño del programa, la utilización de un lenguaje así no es aconsejable debido a su rigidez.

Además de la utilización de las representaciones gráficas (ordinogramas), un programa puede describirse mediante un **pseudocódigo** o notación pseudocodificada que **es un lenguaje de especificación de algoritmos y que podemos considerar como un lenguaje intermedio entre el lenguaje natural y el lenguaje de programación de alto nivel**, de tal manera que permita flexibilidad para expresar las acciones que se han de realizar y, sin embargo, imponga algunas limitaciones, importantes desde el punto de vista de su posterior codificación en un lenguaje de programación determinado.

Al igual que las otras técnicas, la utilización de una notación intermedia permite el diseño del programa sin depender de ningún lenguaje de programación, y es, después del diseño, cuando se codificará el algoritmo obtenido en aquel lenguaje que nos interese.

Al contrario que en otro tipo de técnicas, con respecto al pseudocódigo no hay estandarización o normalización, así que pueden utilizarse diversas notaciones.

El pseudocódigo se ha de considerar más bien una herramienta para el diseño de programas que una notación para la descripción de los mismos. Debido a su flexibilidad permite obtener la solución a un problema mediante aproximaciones sucesivas, es decir, mediante lo que se denomina diseño descendente.

En general, diremos que una notación es un pseudocódigo si mediante ella podemos describir la solución de un problema en forma de algoritmo dirigido a la computadora, utilizando palabras y frases del lenguaje natural sujetas a unas determinadas reglas. Todo pseudocódigo debe posibilitar la descripción de tipos de datos, constantes, variables, expresiones, archivos y otros objetos, y además:

- **Instrucciones de entrada/salida.**
- **Instrucciones de proceso.**
- **Sentencias de control del flujo de ejecución.**
- **Acciones compuestas (un conjunto de las anteriores) que hay que refinar posteriormente.**

6. ESTRUCTURA DE UN ALGORITMO REPRESENTADO EN PSEUDOCÓDIGO.

La estructura general que van a presentar nuestros programas escritos en pseudocódigo es la siguiente:

```
Programa  <nombre del programa>

Entorno
          <descripción de los objetos>

Algoritmo
          <instrucciones>
Fin-algoritmo
```

A su vez, el entorno (definido en el bloque de declaraciones) tendrá el siguiente aspecto:

```
Entorno
Constantes
     <identificador de constante> = <valor>
     ...
Tipos
     <tipo definido por el usuario> = <tipo (simple | estructurado |
definido por el usuario)>
     ...
Variables
     <identificador de variable> : <tipo (simple | estructurado |
definido por el usuario)>
     ...
```

Será siempre necesario darle un valor inicial a las variables en el bloque de instrucciones; nunca lo haremos en el bloque de declaraciones. Precisamente es en este bloque y sólo en él donde se le pueden asignar valor a las constantes.

Para la construcción de los **<identificadores>** seguiremos las siguientes reglas:

- Un identificador debe comenzar siempre por un carácter de subrayado o por un carácter alfabético en mayúsculas o en minúscula (excepto la 'ñ', la 'Ñ' y las vocales acentuadas o con diéresis).

- El resto de los caracteres podrán ser letras, números o símbolos de subrayado. Nunca se admitirán caracteres especiales (incluido el espacio) como '[' ']' '*' '=' ',' '.' '-' ';' ':' 'ñ' 'á' 'é' 'í' 'ó' 'ú' 'ü' etc.

- La longitud podrá variar entre 1 y varios caracteres. No obstante, solamente los primeros 31 caracteres se consideran significativos para el lenguaje C. Y todos los caracteres son significativos para el lenguaje C++ .

- En C, las mayúsculas y las minúsculas tienen un tratamiento diferente (por ejemplo, total, Total y TOTAL se consideran tres identificadores distintos). Además, un identificador no puede coincidir con una palabra reservada y no debe tener el mismo nombre que una función que se haya creado o que forme parte de una biblioteca de C.

Por su parte, el bloque de instrucciones se especificará de la siguiente forma:

```
Algoritmo
     <instrucción-1>
     <instrucción-2>
     . . .
     <instrucción-N>
Fin-algoritmo.
```

7. IDENTIFICADORES ESTÁNDAR DE TIPOS DE DATOS.

Como ya vimos, los tipos de datos pueden ser simples y estructurados. Los tipos de datos estructurados están formados bien por otros tipos de datos estructurados o por tipos de datos simples.

Los tipos de datos simples que hemos visto hasta ahora son: numérico entero, numérico real, carácter y lógico; y entre los estructurados, el tipo cadena.

Los identificadores, para cada uno de los tipos detallados anteriormente, son los siguientes:

Tipo de Dato Simple	Identificador de Tipo
numérico entero	**entero**
numérico real	**real**

lógico	**booleano**
carácter	**carácter**

Estos 4 tipos de datos son los que utilizaremos para la creación de programas sencillos en este tema. Además, y aunque no sea un tipo simple de datos, consideraremos también el tipo de dato **cadena de caracteres**, imprescindible para la realización de algunos programas. Cuando profundicemos en el lenguaje C, las cadenas se estudiarán de forma más precisa. Ahora, sólo nos basta saber que existe un tipo de dato denominado

cadena[<tamaño>]

donde <tamaño> indica el número máximo de caracteres que puede contener la cadena; desde 0 al valor que indique tamaño.

Todos aquellos tipos de datos que no dispongan de un identificador estándar definido deberán ser declarados dentro del apartado **Tipos** del entorno del programa. En temas posteriores se verán las particularidades de cada caso.

8. CLASIFICACIÓN DE LAS INSTRUCCIONES.

Una instrucción puede modificar el entorno (darle valores a las variables), limitarse a la observación de éste (leer datos o escribirlos) o a controlar el orden de ejecución de otras instrucciones (instrucciones de control del flujo del programa).

Según la función que desempeñan dentro de un programa las instrucciones se clasifican de la siguiente manera:

- **Instrucciones primitivas.**
  - De asignación
  - De entrada
  - De salida

- **Instrucciones de control.**
  - Alternativas
  - Repetitivas

9. INSTRUCCCIONES PRIMITIVAS.

Son aquellas que ejecuta el ordenador de forma inmediata. Pueden ser:

- **De asignación.**

Es la instrucción que nos permite realizar cálculos evaluando una expresión y depositando su valor final en un objeto o realizar movimientos de datos de un objeto a otro.

En nuestro pseudocódigo utilizaremos la notación:

```
<variable> = <expresión>
```

aunque también podemos encontrar la siguiente:

```
<variable> ← <expresión>
```

El funcionamiento de esta instrucción es la siguiente:

1º. Se evalúa la <expresión> convirtiéndose en su valor final.
2º. El valor final se asigna a la variable, borrándose cualquier otro valor previo que éste pudiera tener (en este caso se dice que el valor se machaca).

La <variable> y la <expresión> deben coincidir en tipo y se admite que el propio objeto que recibe el valor final pueda intervenir en la misma, pero entendiéndose que lo hace con su valor anterior.

Ejemplo 4:

```
a = 3
b = b + 1
d = b * b - 4 * a * c
nombre = “Dpto. Informática”
```

- **De entrada.**

Su misión consiste en tomar un dato desde un dispositivo externo de entrada y almacenarlo en el espacio de memoria principal reservado para la variable cuyo identificador aparece en la propia instrucción. Si esta variable tiene ya un valor, éste se pierde. En nuestro pseudocódigo lo haremos del siguiente modo:

```
Leer (<variable>)
```

donde <variable> es el objeto donde se va a depositar el dato leído.

También se consideran dentro de esta categoría las instrucciones de depuración de los datos de entrada, es decir, las que se encargan de comprobar la corrección de los mismos.

El dispositivo de entrada de datos es el teclado (dispositivo estándar) mientras no se diga lo contrario.

Ejemplo 5:

```
Leer (nombre)
Leer (radio)
```

- **De salida**

Su misión consiste en enviar datos los datos finales (resultados) desde la memoria central a un dispositivo externo de salida, bien tomándolos de objetos del entorno (constantes o variables), bien definidos de alguna forma en la propia instrucción

o como una combinación de ellos, formando una o varias expresiones (que serán evaluadas y su valor final resultará exteriorizado) separadas todas ellas por comas. Nuestro formato en pseudocódigo será el siguiente:

```
Escribir (<lista de expresiones>)
```

donde <lista de expresiones> son los datos que queremos exteriorizar.

Se incluyen también todas las órdenes e instrucciones para dar formato a los resultados y controlar el dispositivo (saltos de página, borrar pantalla, etc.)

El dispositivo de salida será la pantalla (dispositivo estándar) siempre y cuando no se diga lo contrario.

Ejemplo 6:

```
Escribir ("Buenas tardes")
Escribir ("La suma de a más b es: ", a+b)
Escribir (a, b)
Escribir (a,'*',b," resulta ",a*b)
```

10. INSTRUCCIONES DE CONTROL ALTERNATIVAS.

Como ya sabemos, las instrucciones de control (tanto las alternativas como las repetitivas) son instrucciones que no realizan trabajo efectivo alguno salvo la evaluación de expresiones, generalmente lógicas, con el objetivo de controlar la ejecución de otras instrucciones o alterar el orden de ejecución normal (secuencial) de las instrucciones de un programa.

Las instrucciones de control alternativas son aquellas que controlan la ejecución de uno o varios bloques de instrucciones, dependiendo del cumplimiento o no de alguna condición o del valor final de una expresión. Existen tres modelos típicos de instrucciones alternativas:

- Alternativa simple
- Alternativa doble
- Alternativa múltiple

• Instrucción alternativa simple.

Controla la ejecución de un conjunto de instrucciones por el cumplimiento o no de una condición (resultado de evaluar una expresión lógica), de tal forma que, si la expresión es VERDAD, se ejecutan; si es FALSA, no se ejecutan y se sigue con la siguiente instrucción. En pseudocódigo la representaremos del siguiente modo:

```
Si <condición> entonces
     <lista de instrucciones>
Fin_si
```

Ejemplo 7:

```
Si nota >= 5 entonces
     Escribir ('Asignatura aprobada con un  ', nota)
Fin_si
```

- **Instrucción alternativa doble.**

Controlan la ejecución de dos conjuntos de instrucciones por el cumplimiento o no de una condición (resultado de evaluar una expresión lógica), de tal forma que, si la expresión es VERDAD se ejecutan las instrucciones del primer bloque; si es FALSA, se ejecutan las instrucciones del segundo. En pseudocódigo la representaremos del siguiente modo:

```
Si <condición> entonces
     <lista de instrucciones 1>
   sino
     <lista de instrucciones 2>
Fin_si
```

Ejemplo 8:

```
Si nota >= 5 entonces
   Escribir ('Asignatura aprobada con un  ', nota)
  sino
   Escribir ('Asignatura suspensa con un  ', nota)
Fin_si
```

- **Instrucción alternativa múltiple.**

Cuando existe la posibilidad de elegir entre más de dos caminos emplearemos una instrucción alternativa múltiple; ésta controla la ejecución de varios conjuntos de instrucciones según el valor final de una expresión, de tal forma que cada conjunto de instrucciones está ligado a un posible valor de la expresión, existiendo un bloque final que engloba otros posibles valores no definidos.

- Las distintas opciones tienen que ser disjuntas, es decir, sólo puede cumplirse una de ellas.
- Por otra parte, la <expresión> ha de ser de tipo simple ordinal, es decir, debe pertenecer a un tipo de datos donde todos sus elementos se puedan disponer en una fila ordenada: nunca puede utilizarse un número real. Tipos válidos pueden ser: caracteres, enteros y lógicos.

Este tipo de estructura se puede sustituir por otra alternativa anidada (un conjunto de sentencias Si unas dentro de otras), pero la alternativa múltiple es más sencilla de utilizar y en muchos lenguajes tiene una instrucción específica (instrucción Segun).

En resumen, se evalúa una <expresión> que podrá tomar n valores distintos y, según cual de ellos tome, se realizarán una serie de acciones. En pseudocódigo se representará de la siguiente manera:

```
Segun <expresión> entre
```

```
     <valor 1> : <lista de instrucciones 1>
     <valor 2> : <lista de instrucciones 2>
     ......................
     <valor n> : <lista de instrucciones n>
     sino      <lista de instrucciones n+1>
Fin_segun
```

NOTA:

<valor x> puede ser también un rango de valores expresado de la forma:

```
<valor i> .. <valor j>
```

siendo <valor i> menor que <valor j> .

Ejemplo 9:

La siguiente instrucción alternativa simple anidada...

```
Si peso_vehiculo > 300 AND peso_vehiculo < 1200 entonces
      Escribir (“Turismo”)
      importe_peaje = 500
   sino
      Si peso_vehiculo >= 1200 AND peso_vehiculo < 3000 entonces
            Escribir (“Furgoneta”)
            importe_peaje = 750
         sino
            Si peso_vehiculo >= 3000 Y peso_vehiculo < 10000 entonces
                  Escribir (“Camión”)
                  importe_peaje = 1000
               sino
                  Escribir (“Gran camión”)
                  importe_peaje = 1500
            Fin_si
      Fin_si
Fin_si
```

se podría sustituir por la alternativa múltiple:

```
Segun  peso_vehiculo entre
      300..1200:  Escribir (“Turismo”)
                  importe_peaje = 500
      1200..3000: Escribir (“Furgoneta”)
                  importe_peaje = 750
      3000.10000: Escribir (“Camión”)
                  importe_peaje = 1000
      sino        Escribir (“Gran camión”)
                  importe_peaje = 1500
Fin_segun
```

Ejemplo 10:

```
Segun Tecla entre
      'A' .. 'Z' :      Escribir(“Mayúsculas”)
      'a' .. 'z' :      Escribir(“Minúsculas”)
      '0' .. '9' :      Escribir(“Dígitos”)
      sino              Escribir(“Carácter especial”)
```

```
Fin_segun
```

Ejemplo 11:

```
Segun Tecla entre
      '1' : Escribir(“Seleccionó la opción 1 de Altas de Clientes”)
      '2' : Escribir(“Seleccionó la opción 2 de Bajas de Clientes”)
      '3' : Escribir(“Seleccionó la opción 3 de Altas de Artículos”)
      '4' : Escribir(“Seleccionó la opción 4 de Altas de Artículos”)
      '5' : Escribir(“Seleccionó la opción de Salir del Programa”)
      sino  Escribir(“Ha pulsado una tecla inválida”)
Fin_segun
```

Ejemplo 12 (de alternativa anidada):

```
Si varon entonces
      Si altura > 180 entonces
            Escribir(“Jugará en equipo de baloncesto masculino.”)
         sino
            Escribir(“No es suficientemente alto.”)
      Fin_si
   sino                              ** no es varón
      Si altura > 170 entonces
            Escribir (“Jugará en equipo de baloncesto femenino.”)
         sino
            Escribir(“No es suficientemente alta.”)
      Fin_si
Fin_si
```

11. INSTRUCCIONES DE CONTROL REPETITIVAS.

Las instrucciones repetitivas son aquellas que permiten la ejecución de una secuencia de instrucciones un número -determinado o indeterminado- de veces dependiendo del resultado de evaluar una condición. A esta estructura se le denomina bucle o lazo. Por tanto, un bucle es una parte de un algoritmo que contiene instrucciones que se repiten.

- A las instrucciones que se repiten se les llama cuerpo del bucle y se llama iteración al hecho de ejecutar el cuerpo una vez.
- De algún modo se debe llegar a una situación que provoque la salida del bucle.
- La condición se comprueba a cada iteración o repetición.

Existen tres tipos de instrucciones repetitivas:

- **Instrucción Mientras (testeo antes de la ejecución del bucle).**
- **Instrucción Hacer..Mientras (testeo después de la ejecución del bucle).**
- **Instrucción Para.**

- **Instrucción Mientras.**

La estructura repetitiva **Mientras** es aquella en el que el número de iteraciones que tiene que realizar no se conoce por anticipado y el cuerpo del bucle se repite *mientras* se cumple una determinada condición.

Su formato es el siguiente:

```
Mientras <expresión lógica>
     <cuerpo del bucle>
Fin_mientras
```

... y funciona del siguiente modo:

- La <expresión lógica> se evalúa **antes** de cada ejecución del bucle. Si la condición es VERDADERA, se ejecuta el bucle, y si es FALSA, el control pasa a la sentencia siguiente al bucle.

- Si el resultado de la evaluación es FALSO cuando se ejecuta el bucle por primera vez, el cuerpo del bucle **no se ejecutará nunca**. En este caso, se dice que el bucle se ha ejecutado cero veces.

- Mientras la condición sea VERDADERA, el bucle se ejecutará. Esto significa que el bucle se ejecutará indefinidamente a menos que **"algo" en el cuerpo del bucle modifique la condición** haciendo que su valor pase a FALSO. Si la expresión nunca cambia de valor de la <expresión lógica> a FALSO, entonces el bucle no termina nunca y se denomina **bucle infinito** o sin fin.

En el diseño de cualquier bucle Mientras debe tenerse en cuenta lo siguiente:

- La expresión lógica debe tener un valor la primera vez que se evalúa (es decir, todas aquellas variables que estén involucradas en la condición deben tener un valor); en caso contrario, el programa funcionará de forma incontrolada al ejecutarse el bucle Mientras.
- La expresión lógica debe ser modificada por una sentencia en el cuerpo del bucle; en caso contrario, el bucle es infinito.

- Es posible que el cuerpo del bucle no se ejecute nunca. Esto sucederá si la expresión lógica es FALSA la primera vez que se evalúa.

Ejemplo 13:

```
Leer(numero)
Mientras numero <> 0
      Escribir("El doble de ", numero, " es ", numero * 2)
      Leer(numero)
Fin_mientras
```

- **Instrucción Hacer..Mientras.**

Controla la ejecución del conjunto de instrucciones que configuran el cuerpo del bucle, de tal forma que éstas se ejecutan mientras se cumpla la condición, que será evaluada siempre **después** de cada repetición.

Su formato es el siguiente:

```
Hacer
     <cuerpo del bucle>
Mientras <expresión lógica>
```

... y funciona del siguiente modo:

- La <expresión lógica> se evalúa al final del bucle, después de ejecutarse todo el <cuerpo del bucle>.
- Si <expresión lógica> es VERDADERA, se vuelve a repetir el bucle y se ejecuta el <cuerpo del bucle>.
- Si la <expresión lógica> es FALSA, se sale del bucle y se ejecuta la siguiente sentencia al Hacer..Mientras.
- La sintaxis no requiere Fin_Mientras.

Téngase en cuenta que en una sentencia Hacer..Mientras **siempre se ejecutará el <cuerpo del bucle> al menos una vez**, mientras que en el bucle Mientras puede que no se ejecute nunca si de principio no se cumple la <expresión lógica>.

Ejemplo 14:

```
Hacer
      Leer(numero)
      Escribir("El doble de ", numero, " es ", numero * 2)
Mientras numero <> 0
```

- **Instrucción Para.**

La instrucción Para requiere que conozcamos por anticipado el número de veces que se ejecuta el cuerpo del bucle. Si se desea que las sentencias controladas se ejecuten hasta que ocurra una determinada condición y no se conoce de antemano el número de repeticiones, entonces se debe utilizar la sentencia Mientras o Repetir en lugar de Para.

El formato de la instrucción Para es el siguiente:

```
Para <vble 1>=<vble 2> hasta <vble 3> con incremento <In> hacer
     <cuerpo del bucle>
Fin_Para
```

Las variables <vble 1>, <vble 2> y <vble 3> deben ser todas del mismo tipo de datos.

<vble 2> y <vble3> pueden ser tanto expresiones como constantes.

Su funcionamiento es como sigue:

1º. Antes de la primera iteración del bucle, a <vble 1> se le asigna el valor que contenga <vble 2>.
2º. <vble 1> se incrementa en cada iteración en In.
3º. La última iteración del bucle ocurre cuando el valor de <vble 1> es igual al valor de <vble 3>.

Es "ilegal" intentar modificar el valor de <vble 1>, <vble 2> o <vble 3> dentro del bucle.

Ejemplo 15:

```
Escribir (“Tabla de multiplicar del número: “, dato);
Para indice = 1 hasta 10 con incremento 1 hacer
      Escribir (dato, ‘x’, indice ,” = “ , dato * indice)
Fin_Para
```

12. ELEMENTOS AUXILIARES DE UN PROGRAMA.

Son objetos que realizan funciones específicas y, por su utilidad, frecuencia de uso y peculiaridades, vamos a hacer un estudio separado de las mismas al tiempo que haremos algunas consideraciones a la hora de escribir programas.

- **Contador.**

Es una variable que se utiliza para contar cualquier evento que pueda ocurrir dentro de un programa.

Un contador debe tener un valor inicial, normalmente 0, y se incrementa con una cantidad constante positiva o negativa.

- **Acumulador.**

Es una variable cuyo contenido se incrementa sucesivamente en cantidades variables. Se suele utilizar en aquellos casos en que se desea obtener un total acumulado de un conjunto de cantidades. En general se utilizan para calcular sumas y productos.

Es necesario inicializarlo con valor 0 (elemento neutro de la suma). Si, por el contrario, lo que se quiere acumular es una serie de productos, el acumulador deberá inicializarse con valor 1 (elemento neutro del producto).

El incremento del acumulador puede ser positivo o negativo.

- **Interruptor.**

También se le denomina conmutador, switch o bandera(flag). Son variables que sólo pueden tomar dos valores: VERDADERO o falso, 0 ó 1, 1 ó -1, etc.

Se suelen utilizar para:

- Recordar en un determinado punto del programa la ocurrencia o no de un suceso anterior.

- Para hacer que dos acciones diferentes se ejecuten de forma alternativa.

Se inicializan a un valor y en el punto del programa que corresponda se cambia al valor contrario.

- **Sangrías.**

Se trata del conjunto de espacios en blanco que se dejan a la izquierda de una línea para resaltar una parte del texto. En programación se utilizan asiduamente para marcar las dependencias de unas instrucciones con otras.

- **Comentarios.**

Son líneas explicativas cuyo objetivo es facilitar la comprensión del programa a quien lo lea. Estas líneas serán ignoradas por el procesador cuando ejecute el programa.

Un programa, en cualquier lenguaje, debe estar ampliamente documentado mediante comentarios intercalados a lo largo de todo su listado. Esto facilitará las posibles y necesarias modificaciones del mismo al simplificar su comprensión al programador que lo diseñó o a otro diferente encargado de su mantenimiento.

Los comentarios se utilizan para aclarar:

- El significado o cometido de un objeto del programa.
- El objetivo de un bloque de instrucciones.
- La utilización de una determinada instrucción.
- Cualquier aspecto del programa que sea necesario aclarar.

En la fase de diseño del programa no es necesario excederse en los comentarios pero sí se han de incluir aquellos que consideremos necesarios.

Se escribirán en cualquier línea entre los símbolos **/*** y ***/**.

- **Acciones compuestas.**

Una acción compuesta es aquella que ha de ser realizada dentro del algoritmo, pero que aún no está resuelta en términos de accione simples y sentencias de control.

En el diseño del programa se incluirán los nombres de las acciones compuestas en el algoritmo y posteriormente habrá que refinarlas, sustituyendo cada nombre por las instrucciones correspondientes o colocándolas aparte, mediante lo que se denomina subprograma, de la siguiente manera.

```
Programa <nombre del programa>
```

```
Entorno
     <descripción de los objetos globales al programa>
Algoritmo
     <instrucciones>
Fin_algoritmo

Subprograma <nombre del subprograma>
Entorno
     <descripción de los objetos locales al subprograma>
Subalgoritmo
     <instrucciones subprograma>
Fin_subalgoritmo

     ...
```