INTRODUCCIÓN A LA PROGRAMACIÓN

ISBN: 978-607-9063-26-9

Luis Manuel Martínez Hernández
Paula Elvira Ceceñas Torrero
María Elizabeth Levva Arellano

Autores

Luis Manuel Martínez Hernández
Universidad Juárez del Estado de Durango (UJED)
Instituto de Investigaciones Históricas - UJED
Facultad de Ciencias Exactas – UJED
Facultad de Psicología – UJED
Universidad Pedagógica de Durango
Red Durango de Investigadores Educativos, A. C.

Paula Elvira Ceceñas Torrero
Universidad Pedagógica de Durango
Área Básica - UJED

María Elizabeth Leyva Arellano
Facultad de Ciencias Químicas – UJED

Revisión

Yareli Villalba Segovia
Yenifer Rivas García

Primera Edición: Septiembre de 2014
Editado en México
ISBN: 978-607-9063-26-9

Editor:
Red Durango de Investigadores Educativos, A. C.

Coeditores:
Universidad Juárez del Estado de Durango
Benemérita y Centenaria Escuela Normal del Edo. de Dgo.
Universidad Pedagógica de Durango
Centro de Actualización del Magisterio (Durango)
Instituto Universitario Anglo Español
Instituto de Investigaciones Históricas - UJED
Facultad de Ciencias Exactas – UJED
Facultad de Psicología - UJED
Facultad de Ciencias Químicas - Durango – UJED
Escuela de Lenguas - UJED
Área Básica – UJED

Pintura de la portada (al oleo):
Diana Elizabeth Martínez Leyva – Nombre de la pintura “Dolphy y bebé”

Diseño de portada
Diana Elizabeth Martínez Leyva

Corrección de estilo: Mtra. Paula Elvira Ceceñas Torrero

INTRODUCCIÓN

PARA CUALQUIERA QUE TENGA ESTE MANUAL EN SUS MANOS:

Para ti que tu pasión es la programación o para ti que la odias, para ti que eres un experto o, en especial, para ti que crees que la computación no tiene nada que ver contigo. Tal vez has oído hablar de los PARADIGMAS, si de aquellos que nos indican la forma tradicional de hacer las cosas, los que nos dicen como están establecidas las reglas del pensamiento, pero también de aquellos que nos impiden ver más allá de lo que se nos inculca.

¿Qué quiere decir todo esto? Bueno, cuántas veces dentro del medio donde vivimos se nos presentan cosas que en primera impresión nos parecen ilógicas, pero cuántas veces también nos damos cuenta de que realmente lo son en ese preciso momento en que las vemos; siempre se nos inculca una forma de cómo "deben" ser las cosas, que en ese preciso momento en que vemos algo ilógico de inmediato nuestro cerebro lo identifica, pero más pronto aún nuestro cerebro bloquea la información, pues es tan ilógico el hecho, que es imposible creer que exista. Más rápido que un segundo, el cerebro es capaz de cambiar la información a algo más lógico, esto es un paradigma.

Los paradigmas son de lo más comunes, cada quien en su área, ya sea trabajando en la mecánica, en la física, en la programación, en el mercado, etc., es víctima y beneficiario de los paradigmas, víctima por el hecho de ejercer una resistencia al cambio, beneficiarios porque un paradigma en nuestra área es más fácil de ser vencido por otro paradigma de otra área.

Los suizos, los inventores del reloj, dejaron ir la oportunidad de su vida cuando un inventor les presentó la innovadora forma del reloj, el reloj de cuarzo que vino a sustituir a el reloj mecánico que funcionaba a base de engranajes complejos, estos rechazaron la idea la cual fue presentada en una exposición internacional, los japoneses y los estadounidenses de "Texas Instrumens" no la dejaron escapar y hoy son ellos los que controlan el mercado del

reloj. Pero lo más sorprendente no es el hecho de que Suiza no sea la potencia del mercado del reloj, sino el hecho de que el inventor fue un suizo.

Muchos científicos han encontrado la solución a grandes problemas utilizando conceptos que parecen ilógicos, pero que son más lógicos que el problema que estudian, lo que hace ilógico a estos conceptos es el paradigma que corresponde al área de trabajo.

Los innovadores de un área son foráneos a ella, y he aquí la relación y el propósito de la redacción de esta introducción, con los paradigmas de personas que parecieran ser ajenas a la programación (foráneos), se podría romper los paradigmas que en la programación existen, y lograr con ello ampliar los horizontes de la programación.

Después de lo antes expuesto es el momento de dar el paso mayor, es nuestra elección dar ese importante paso, con la garantía que el cambio ofrece, pues recordemos que el cambio siempre es bueno solo hay que tener el valor para dar el paso, antes de enfermarnos de parálisis paradigmática, una enfermedad que nos hace ver los paradigmas como un reto y no como una oportunidad de cambiar a algo mejor.

ÍNDICE

CAPÍTULO I. Conjuntos y relaciones

1.1 Conjuntos y Elementos

Un conjunto puede ser considerado como una colección de objetos, los elementos o miembros del conjunto. Normalmente, usaremos mayúsculas A, B, X, Y,…para denotar conjuntos, y minúsculas a, b, x, y,…para denotar elementos de conjuntos. El enunciado “p es un elemento de A”, o equivalentemente, “p pertenece a A”, se escribe:

$$p \in A$$

La negación de $p \in A$ se describe $p \notin A$.

Principio de extensión: dos conjuntos A y B son iguales sí y solo si tienen los mismos elementos.

Ejemplo:

(a) El conjunto A de arriba también se puede escribir como:

A = {x | x es una letra del alfabeto español, x es una vocal }

Principio de abstracción: dado cualquier conjunto U y cualquier propiedad P, hay un conjunto A tal que los elementos de A son exactamente aquellos miembros de U que tienen la propiedad P.

1.2 Conjunto Universal y Conjunto Vacío

Para un conjunto U fijo y una propiedad P, puede no haber ningún elemento de U que tenga la propiedad P, por ejemplo, el conjunto:

$$S = \{x \mid x \text{ es un entero positivo}, x^2 = 3\}$$

No tiene elementos ya que ningún entero positivo tiene la propiedad requerida.
El conjunto sin elementos se llama *conjunto vacío* y se denota por: Ø

En otras palabras, si S y T con ambos vacíos, entonces S = T ya que tiene los mismos elementos, o sea, ninguno.

1.3 Subconjuntos

Si cada elemento de un conjunto A es también elemento de un conjunto B, entonces A se llama subconjunto de B. También decimos que A esta contenido en B o que B contiene a A.

Esta relación se escribe $A \subset B$

Si A no es un subconjunto de B, es decir, si por lo menos un elemento de A, no pertenece a B, escribimos $A \not\subset B$.

Ejemplo:

(a) Considere los conjuntos

$A = \{1, 3, 4, 5, 8, 9\}$

$B = \{1, 2, 3, 5, 7\}$

$C = \{1, 5\}$

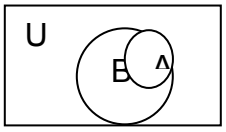
Entonces $C \subset A$ y $C \subset B$ ya que 1 y 5, los elementos de C también son elementos de A y de B. Pero $B \not\subset A$ ya que algunos de sus elementos, por ejemplo 2 y 7, no pertenecen a A. Además, como los elementos de A, B y C también tienen que pertenecer al conjunto universal U, tenemos que U por lo menos tiene que contener el conjunto {1, 2, 3, 4, 5,6, 7, 8, 9}.

Teorema 1

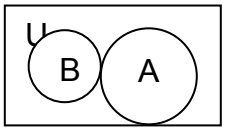
(1) Para todo conjunto, se tiene Ø ⊂ A ⊂ U.
(2) Para todo conjunto A, se tiene A ⊂ A.
(3) A ⊂ B si y B ⊂ C, entonces A ⊂ C.
(4) A = B si y solo si A ⊂ B y B ⊂ A.

1.4 Diagrama de Venn

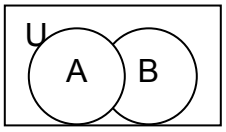
Un diagrama de Venn es una representación pictórica de conjuntos por conjuntos de puntos en el plano. El conjunto universal U se representa por el interior de un rectángulo, y los otros conjuntos se representan por discos contenidos en un rectángulo.



(a) A c B



(b) A y B son disyuntivas



(c) no están ni A ni en B

1.5 Unión e Intersección

La unión de dos conjuntos A y B, denotada por A U B, es el conjunto de todos los elementos que pertenecen a A o a B;

$$A \cup B = \{x \mid x \in A \text{ ó } x \in B\}$$

La intersección de dos conjuntos A y B, denotada A ∩ B, es el conjunto de elementos que pertenecen tanto a A como a B.

$$A \cap B = \{x \mid x \in A, x \in B\}$$

1.6 Complementos

El complemento absoluto o, sencillamente, complemento de un conjunto A, denotado por A`, es el conjunto de elementos que pertenecen a U, pero que no pertenecen a A:

$$A` = \{x \mid x \in U, x \notin A\}$$

El complemento relativo de un conjunto B con respecto a un conjunto A o, sencillamente la diferencia de A y B, denotada por A \ B, es el conjunto de elementos que pertenecen a A, pero que no pertenecen a B:

$$A \setminus B = \{x \mid x \in A, x \notin B\}$$

1.7 Algebra de Conjuntos: Dualidad

Los conjuntos, bajo las anteriores operaciones de unión, intersección y complemento, satisfacen las leyes enumeradas en la tabla 4-1, una similaridad basada en la analogía entre las operaciones entre conjuntos U, ∩, y complemento, y las conectivas lógicas ˇ, ˆ, y la negación.

Leyes de ídem potencias	
1ª. A U A = A	1b. A ∩ A = A
Leyes asociativas	
2ª. (A U B) U C= A U (B U C)	2b. (A ∩ B) ∩ C= A ∩(B ∩ C)
Leyes conmutativas	
3ª. A U B= B U A	3b. A ∩ B = B ∩ A
Leyes distributivas	
4ª. A U (B ∩ C) = (A U B) ∩ (A U C)	4b. A ∩ (B U C) = (A ∩ B) U (A ∩ C)
Leyes de identidad	
5ª. A U Ø = A	5b. A ∩ U = A
6ª. A U U = U	6b. A ∩ Ø = Ø
Leyes de involución	
7.(A) = A	
Leyes de complemento	
8ª. A U A = U	8b. A ∩ A = Ø
9ª. U = Ø	9b. Ø = U
Leyes de De Morgan	
10ª. (A U B)` = A` ∩ B`	10b. (A ∩ B)` = A` U B`

1.8 Conjuntos Finitos, Principio de Conteo

Se dice que un conjunto es finito si contiene exactamente m elementos diferentes en donde m denota algún entero no negativo. En caso contrario, se dice que el conjunto es infinito. Por ejemplo, el conjunto vacío y el conjunto de letras en el alfabeto español son finitos, mientras que el conjunto de los enteros positivos pares {2, 4, 6,…}, es infinito.

Si un conjunto A es finito, n(A) denotará el número de elementos A.

Lema: Si A y B son conjuntos finitos disyuntivos, entonces A U B es finito y n(A U B) = n(A)+n (B).

Teorema 2

Si A, B y C son conjuntos finitos, entonces A U B y A ∩ B son finitos y n(A U B) = n(A) + n (B) - (A ∩ B)

Podemos aplicar este resultado para obtener una fórmula semejante pasa cualquier número finito, K, de conjuntos finitos.

1.9 Clases de Conjuntos, Conjunto Potencia, Particiones

Dado un conjunto S, podemos hablar de algunos de sus subconjuntos. Así, estaríamos considerando un conjunto de conjuntos. Cada vez que se presenta esta situación, para evitar confusiones hablaremos de una clase de conjuntos o una colección de conjuntos en lugar de un conjunto de conjuntos. Si queremos considerar alguno de los conjuntos en una clase dada de conjuntos, hablaremos entonces de una subclase o de una subcolección. Los subconjuntos en una partición se llaman células.

1.10 Parejas Ordenadas, Conjuntos Productos

El orden de los elementos en un conjunto con dos elementos no interesa, por ejemplo: (3,5) = (5,3)

Por otra parte, una pareja ordenada consiste en dos elementos de los cuales uno designa el primer elemento, y el otro, el segundo. Tal pareja ordenada se escribe (a, b), en donde a es el primer elemento y b es el segundo. Dos parejas ordenadas (a, b) y (c, d) son iguales si y solamente si a = c y b = d.

1.11 Relaciones

Sean A y B conjuntos. Una relación binaria, R, de A en B asigna a cada pareja ordenada (a, b) en A x B exactamente uno de los siguientes enunciados:

1).- “a esta relacionado con b”, escrito a R b
2).- “a no está relacionado con b”, escrito a R b

Una relación de un conjunto A en el mismo conjunto A se llama relación en A.

Definición. Una relación R de A en B es un subconjunto de A x B.

El dominio de una relación R es el conjunto de todos los primeros elementos de las parejas ordenadas que pertenecen a R, y el recorrido de R es el conjunto de todos los segundos elementos.

1.12 Representaciones Gráficas de Relaciones

Consideremos una relación S en el conjunto R de los números reales, es decir, S es un subconjunto de $R^2 = R \times R$. Como R^2 se puede representar por un conjunto

de puntos en el plano, podemos visualizar S destacando aquellos puntos en el plano que pertenecen a S. La representación gráfica de esta relación a veces se llama gráfica de la relación.

Frecuentemente, la relación S consta de todas las parejas ordenadas de números reales que satisfacen alguna ecuación dada: E(x, y) = 0

1.13 Relaciones de Equivalencia

Una relación ~ en un conjunto S se llama relación de equivalencia si tiene las siguientes tres propiedades:

1. para cada a en S, tenemos a ~ a.
2. si a ~ b, entonces b ~ a.
3. si a ~ b y b ~ c, entonces a ~ c.

La primera propiedad se llama reflexiva, la segunda simétrica y la tercera transitiva. Así que una relación en un conjunto es una relación de equivalencia si es reflexiva, simétrica y transitiva.

La idea general detrás de una relación de equivalencia es que esta nos da una clasificación de objetos que son de alguna manera “similares”. De hecho, la relación de igualdad en cualquier conjunto S es una relación de equivalencia; es decir, a = a para cada a en S; si a = b, entonces b = a; y si a = b y b = c, entonces a = c.

Teorema 3

Sea ~ una relación de equivalencia en un conjunto S. Entonces el conjunto cociente S/~ es una partición de S. Específicamente:

1. para cada a en S tenemos a ∈ [a]

2. [a] = [b] si y solamente si a~b

3. si [a] ≠ [b], entonces [a] y [b] son disyuntos.

1.14 Funciones

Supongamos que a cada elemento de un conjunto A se le asigna un único elemento de un conjunto B; la colección de tales asignaciones se llama función de A en B.

El conjunto A se llama dominio de la función y al conjunto B se le llama condominio.

Generalmente se usa un símbolo para denotar una función.

Al conjunto de todos los valores de imágenes se le llama recorrido o imagen de f.
Una función se puede expresar por medio de una fórmula matemática.

Definición. Una función f: A→ B es una relación de A en B, tal que cada a ∈ A pertenece a una pareja ordenada única (a, b) en f.

Problemas Resueltos

Conjuntos. Operaciones en Conjuntos.

1. ¿Cuáles de los siguientes conjuntos son iguales: {r, t, s}, {s, t, r, s}, {t, s, t, r}, {s, r, s, t}?

   R= Todos son iguales. El orden y/o la repetición no cambian un conjunto.

2. Considere los siguientes conjuntos:

   Ø, A= {1}, B= {1, 3}, C= {1, 5, 9}, D= {1, 2, 3, 4, 5}, E= {1, 3, 5, 7, 9}, U= {1, 2,…, 8, 9}

   Inserte el símbolo correcto c o ⊄ entre cada pareja de conjuntos:

   a) Ø c A ya que Ø es un subconjunto de todo conjunto.
   b) A c B ya que 1 es el único elemento de A y pertenece a B.
   c) B ⊄ C ya que 3 c B pero 3 ∉ C.
   d) B c E ya que todos los elementos de B también pertenecen a E.
   e) C ⊄ D ya que 9 está en C pero 9 no está en D.
   f) C c E ya que los elementos de C también pertenecen a E.
   g) D ⊄ E ya que 2 ∈ D pero 2 ∉ E.
   h) D c U por que los elementos de D también pertenecen a U.

3. Muestre que podemos tener A ∩ B = A ∩ C sin que B = C.

   Sea A = {1, 2}, B = {2, 3} y C = {2, 4}. Entonces, A ∩ B = {2} y A ∩ C = {2}. Así A ∩ B = A ∩ C pero B es distinto de C.

Algebra de Conjuntos, Diagramas de Venn.

4. Demuestre la identidad (U ∩ A) U (B ∩ A) = A.

(U ∩ A) U (B ∩ A)= (A ∩ U) U (A ∩ B)	Ley conmutativa
= A ∩ (U U B)	Ley distributiva
= A ∩ (B U U)	Ley conmutativa
= A ∩ U	Ley de identidad
= A	Ley de identidad.

Clases de Conjuntos, Particiones

5. Determine el conjunto potencia P(S) de S = {a, b, c, d}.

   Los elementos de P(S) son los subconjuntos de S. Así que:

   P(S)=[S, {a, b, c},{a, b, d},{a, c, d},{ b, c, d},{a, b},{a, c},{a, d},{b, c}, {b, d}, {c, d}, {a}, {b}, {c}, {d},Ø]

   Observe que P(S) tiene 2^4 =16 elementos.

Relaciones

6. Dado (2x, x+y) = (6,2) encuentre x y y.

   2x = 6; x = 3 x+y = 2; x = -1

Funciones

7. Sea A el conjunto de estudiantes de un colegio. Determine cuáles de las siguientes asignaciones definen funciones en A. (a) A cada estudiante, su edad; (b) a cada estudiante, el profesor; (c) a cada estudiante, su sexo y (d) a cada estudiante, su cónyuge.

   Una colección de asignaciones es una función en A si y sólo si a cada elemento a en A se le asigna exactamente un miembro. Así que:

   a) Sí, porque cada estudiante tiene una y solamente una edad.
   b) Sí, si cada estudiante tiene solamente un profesor; no, si algún estudiante tiene más de un profesor.

c) Sí.
d) No, si algún estudiante no está casado.

Problemas Suplementarios

Conjuntos, Operaciones en Conjuntos

1.- ¿Cuáles de los siguientes conjuntos son iguales?

{1, 2}, {1, 3}, {2, 1}, {3, 1, 3}, {1, 2, 1}

$A = \{x \mid x^2-4x+3 = 0\}$ $C = \{x \mid x \in N, x < 3\}$

$B = \{x \mid x^2-3x+2 = 0\}$ $D = \{x \mid x \in N, x \text{ es impar}, x< 5\}$

RESPUESTA. {1, 2} = {2, 1} = {1, 2, 1} = B = C, {1, 3} = {3, 1, 3} = A = D

2.- Haga una lista de los siguientes conjuntos si el conjunto universal es U = {a, b, c,…, y, z}. ¿Cuáles de estos conjuntos, si hay alguno, son iguales?

A = {x | x es una vocal}
B = {x | x es una letra en la palabra orca}
C = {x | x precede a f en el alfabeto}
D = {x | x es una letra en la palabra arco}
RESPUESTA. A = {a, e, i, o, u}, B = D = {l, i, t, e}, C = {a, b, c, d, e}

3.- Sea A = {1, 2,..., 8, 9}, B = {2, 4, 6, 8}, C = {1, 3, 5, 7, 9}, D = {3, 4, 5} y E = {3, 5}. ¿Cuáles conjuntos pueden ser iguales a X si se nos da la siguiente información?

a) X y B son disyuntos. b) X c D pero $X \not\subset B$
c) X c A pero $X \not\subset C$ d) X c C pero $X \not\subset A$

RESPUESTA. a) C y E, b) E, c) B, d) ninguno

4.- La fórmula A \ B= A ∩ B` define la operación de diferencia en términos de las operaciones de intersección y complemento. Encuentre una fórmula que defina la unión de dos conjuntos, A U B, en términos de las operaciones de intersección y complemento.

RESPUESTA. A U B= (A` ∩ B`) `

5.- Determine cuáles de los siguientes conjuntos son finitos:

(a) el conjunto de las rectas paralelas al eje x.
(b) el conjunto de las letras del alfabeto español.
(c) El conjunto de números que son múltiplos de 5.
(d) El conjunto de animales que viven en la tierra.
(e) El conjunto de números que son soluciones de la ecuación x ^ 27+ 26x ^ 18 – 17x ^ 11+ 7x ^ 3 -10= 0
(f) El conjunto de círculos con centro en el origen (0, 0)

RESPUESTA. (a) Infinito, (b) finito, (c) infinito (d) finito, (e) finito, (f) infinito.

CAPÍTULO 2. ARITMÉTICA DEL COMPUTADOR

2.1 CONCEPTOS MATEMÁTICOS BÁSICOS

En este tema se van a ver conceptos matemáticos que son necesarios en el estudio de la aritmética del computador.

- **Números aproximados; dígitos significativos**

En un dispositivo, como en una calculadora, un micrómetro, un computador electrónico moderno solo puede manejar un número finito de dígitos en un momento dado, por ejemplo, la altura de un estudiante puede ser registrada como 187 cm., siendo que su verdadera estatura es 187.5 cm. También resultan números aproximados cuando se usan fracciones decimales que terminan para representar números irracionales, como:

$$\pi \approx 3.1416 \qquad \sqrt{2} = 1.414$$

La exactitud de un número aproximado A se mide con el número de dígitos significativos de A. Por ejemplo, una onza se puede aproximar en el sistema métrico como: 28 gr. o 0.028 kg.

Las reglas formales para los dígitos significativos son las siguientes:

1) Un dígito no nulo siempre es significativo.
2) El dígito 0 es significativo si se encuentra entre otros dígitos significativos.
3) El dígito 0 nunca es significativo cuando está precedido por dígitos no nulos.

Ejemplos:

-Considerar el número 3.14 y 880.077, todos son dígitos significativos, así que estos números contienen 3 y 6 dígitos significativamente.

-Considere el número 0.000345, éste número contiene solamente 3 dígitos significativos, el 3, 4 y 5.

✖ Redondeo de números

Para aproximar un valor numérico por medio de otro número se quita uno o más de los dígitos menos significativos y luego redondeando el número que queda.

Las reglas para redondeo, en donde “dígito de prueba” se refiere al primer dígito de los que se quitan, son:

-Aproximación por defecto. Si el dígito de prueba es menor que 5, los dígitos precedentes no se cambian.

-Aproximación por exceso. Si el dígito de prueba es mayor que 5 o es un 5 seguido por lo menos de un dígito no nulo, el dígito precedente se aumenta en 1(llevando un 1 si el dígito precedente es un 9).

-Regla sume si impar. Si el dígito de prueba es 5 seguido solamente de ceros, el dígito precedente no se cambia si es par, pero se incrementa en 1 si es impar.

Bajo estas reglas, el máximo error de redondeo será la mitad del valor de posición del último dígito que quede.

Ejemplos:

-Redondear a 2 cifras: 1.73482 = 1.73, 258.678 = 258.68

-Redondear a 1 cifra: 1.152 = 1.2, 3.35 = 3.4

-Redondear a 3 dígitos significativos: 0.77777 = 0.778, 888.5 = 888

- **Truncamiento**

Muchos cálculos aritméticos en el computador resultan con más dígitos de los que se pueden almacenar en las localizaciones de memoria. En lugar de redondear los números, la mayoría de los computadores se programan simplemente para suprimir los dígitos menos significativos. A ésta operación se le llama truncamiento.

Ejemplo:	Número	88.77	-7.8989	999.111	-0.012345
	Valor truncado	88.7	-7.89	999	-0.0123

- **Valor absoluto**

El valor absoluto de un número puede verse intuitivamente como la magnitud, sin tener en cuenta el signo. Se denota el valor absoluto de un número a por $|a|$, éste se define como el mayor de los números de a y –a. O, sea:

$$|a| = a \text{ si } (a>0)$$
$$= 0 \text{ si } (a=0)$$
$$= -(a) \text{ si } (a<0)$$

Ejemplo:

-Encontrar el valor absoluto de: 15= $|15| = 15$, 0= $|0| = 0$ y -8= $|-8| = 8$

-Evaluar: $|3-8| = |-5| = 5$, $|3| - |8| = 3-8 = -5$ y $-|-5| = -(5) = -5$

2.2 FORMA EXPONENCIAL

Un número muy grande o muy chico se puede abreviar, escribiendo un número multiplicado por una potencia de 10. Todo número se puede escribir en forma de un número veces una potencia de 10, llamada forma exponencial.

- Se le llama base 10.
- b=10.
- Alfabeto={0,1,2,3,4,5,6,7,8,9}
- El número an an-1... a0. a-1 ...a-(l-1) a-l equivale a la expresión:

an 10n +an-1 10n-1 +...+a0+a-1 10-1 +...+a-(l-1) 10-(l-1) +a-l 10-l

Ejemplo:

567= 5.67×10² 0.005= 5.00×10^-3 25= 0.25×10²

Tal forma no es única, por ejemplo:

567 = 0.0567×10^4 = 0.567×10³ = 56.7×10^1 = 0.25*10^-2

Considerar ahora cualquier número no nulo A. Podemos escribir A = M×10ⁿ, en donde el punto decimal aparece directamente enfrente del primer dígito no nulo en M. A esto se le llama forma exponencial normalizada de A. Al número M se le llama mantisa de A, y al exponente n se le llama exponente de A. Así mismo los números binarios, como los números decimales

Ejemplo:

NUMERO DECIMAL	FORMA EXPONENCIAL NORMALIZADA	MANTISA	EXPONENTE
222.2	0.2222×10³	0.2222	3
-0.00006	-0.6×10^-4	-0.6	-4

Así mismo los números binarios, como los números decimales, se pueden escribir en forma exponencial, usando potencias de dos en lugar de potencias de 10.

- El más utilizado es el binario o de base 2.
- b = 2
- alfabeto = {0,1}
- Cada cifra se denomina **bit**, corresponden dentro del computador a valores de tensión almacenados.
- Otros sistemas usados en Informática son el **octal** y el **hexadecimal** (base 8 y 16 respectivamente).
- Se usan por su transformación fácil y directa entre número expresados en binario y estas bases.
- Los alfabetos son {0,1,2,3,4,5,6,7} y {0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f}

Ejemplo:

NUMERO BINARIO	FORMA EXPONENCIAL NORMALIZADA	MANTISA	EXPONENTE
1010.1	0.10101×2^4	0.10101	4
0.001111	0.11110×2^-2	0.11110	-2

Un código binario es **continuo** si sus combinaciones de bits o palabras código correspondiente a números decimales consecutivos difieren en un solo bit, es decir, son adyacentes. Si además se cumple que la última combinación es adyacente a la primera, se denomina **cíclico.** El código **GRAY** o reflejado es binario continuo y cíclico. El código de Gray de n bits, se forma a partir del código de Gray de n-1 bits, reflejando éste a partir de una línea horizontal y rellenando por encima de la línea con ceros a la izquierda, y por debajo con unos. Es adecuado para realización de contadores con registros de desplazamiento.

2.3 ARITMÉTICA DEL COMPUTADOR

Los computadores normalmente ejecutan cálculos aritméticos con números en forma exponencial; podemos llamar a esto aritmética de punto flotante o aritmética real. Algunos lenguajes de programación, tales como el FORTRAN, hacen posible que el computador ejecute un tipo separado de aritmética para números almacenados como enteros en punto fijo.

- ✓ **Aritmética de enteros**

La principal propiedad de la aritmética de enteros es que el resultado de cualquier operación con enteros debe ser entero. Como:

12+15 = 17 12-5 = 7 12×5= 60

Pero la división de enteros, el resultado se obtiene truncando el cociente usual a un entero. Como:

12÷5= 2 7÷8= 0 -9÷2= -4

- ✓ **Aritmética de punto flotante (Real)**

Ahora todos los números se almacenan y procesan en forma exponencial. Sea P la precisión del computador. El principal hecho para recordar es que el resultado de cualquier operación es normalizado y que la mantisa se redondea o se trunca a P dígitos.

- En la adición real si dos números se van a sumar y tienen exponentes iguales, las mantisas se suman y se queda el mismo exponente, como:

0.2356×10^4 + 0.4123×10^4= 0.6479×10^4

- En cambio, si tienen exponentes diferentes, entonces uno tendrá que normalizarse de tal manera que tengan el mismo exponente, como:

$0.1166\times10^{2} + .8811\times10\text{^}4 = 0.001166\times10\text{^}4 + 0.8811\times10\text{^}4 = 0.882266\text{^}\times10\text{^ }4$

- La substracción real es muy similar a la adición real, solamente que se restan las mantisas e igualmente queda el mismo exponente.

$0.8844\times10\text{^}-2 - 0.3322\times10\text{^}-2 = 0.5522\times10\text{^}-2$

$0.6666\times10^{3} - 0.3333\times10^{2} = 0.6666\times10^{3} - 0.03333\times10^{3} = 0.63327\times10^{3}$

- En la multiplicación real se multiplican las mantisas y se suman los exponentes, como:

$(0.3355\times10^{2}) \times (0.4466\times10^{3}) = 0.1498\times10\text{^}5$

$(0.4444\times10\text{^}-5) \times (0.3579\times10^{2}) = 0.1590\times10\text{^}-3$

- En la división real se dividen las mantisas y se restan los exponentes, como:

$(0.4444\times10\text{^}7) \div (0.1357\times10\text{^}4) = 3.271\times10^{3}$

$(0.3333\times10\text{^}-4) \div (0.2121\times10^{2}) = 0.5089\times10\text{^}-6$

2.4 ERRORES

Como el computador retiene un número limitado de dígitos significativos, la mayoría de los valores numéricos almacenados y de cálculos son solo aproximaciones de los valores verdaderos. La diferencia entre el verdadero valor y el valor aproximado de una cantidad se llama error absoluto, esto es: $ę = A - \bar{A}$ y la razón entre el error absoluto y el valor verdadero es el error relativo y se expresa como porcentaje, esto es:

$$r = \frac{ę}{A} = \frac{A - \bar{A}}{A}$$

Ejemplo:

Si A = 1.427, encuentre el error absoluto y el error relativo cuando a) A es redondeado a 1.43 y b) A es truncado a 1.42.

a) El error absoluto e es la diferencia: e = 1.427 – 1.43 = –0.003

El error relativo r es la razón:

$$r = \frac{e}{A} = \frac{-0.003}{1.427} = -0.0021$$

Así, $|r| = 0.21$

e= 1.427 – 1.42= 0.007

b) $$r = \frac{e}{A} = \frac{0.007}{1.427} = 0.0049$$

Así, $|r| = 0.49\%$, que es más de 2 veces el error relativo en a).

PROBLEMAS

1.- Determinar el dígito más significativo, el menos significativo, y el número de dígitos significativos en:

a) 44.44, 30 303, 6.6707, 5.005

44.44, 4 y 4, cuatro; 30 303, 3, cinco; 6.6707, 7, cinco; 5.005, 5, cinco.

b) 0.222, 0.000 333 3, 00 011, 0.008 008

0.222, 2, tres; 0.000 333 3, 3, cuatro; 00 011, 11, dos; 0.008008, 8, cuatro.

c) 2.220 00, 3300.000, 0.004 440, 55 500 000

2.220 00, 0, seis; 3300.000, 0, siete; 0.004 44 0, 0, cinco; 55 500 000, 5, seis.

2.- Redondear a) a 2 decimales, b) a 3 decimales, c) a 5 dígitos significativos:

1) 22.4444 a) 22.44 b) 22.444 c) 22.44

2) 1.234 567 a) 1.23 b) 1.235 c) 1.235

3) 333.777 a) 333.78 b) 333.777 c) 333.8

4) 0.065 4321 a) 0.06 b) 0.065 c) 0.06543

5) 0.005 678 a) 0.00 b) 0.006 c) 0.005678

3.- Redondear a) a 2 decimales, b) a 2 dígitos significativos:

1) 0.445 00 a) 0.44 b) 0.44 2) 7.775 a) 7.78 b) 7.8

3) 66.665 000 a) 66.66 b) 67 4) 8.885020 a) 8.89 b) 8.9

5) 2.3350 a) 2.34 b) 2.3

4.- Truncar a) a un entero, b) a 4 dígitos significativos:

1) 44.44 a) 44 b) 44.44 2) 30.303 a) 30 b) 30.30

3) 6.6707 a) 7 b) 6.670 4) 5.005 a) 5 b) 5.005

5) –0.044 488 8 a) 0 b) -0.044 48

5.- Sea TRUN (Q) la parte entera del número Q. Resolver la ecuación: 2×TRUN(N/2) =N N/2 = entero, o N = entero par

6.- Evaluar: a) $|4-9| = |-5| = 5$ b) $|-4-9| = |-13| = 13$

c) $|-4+9| = |5| = 5$ d) $|-4|-|9| = 4-9 = -5$

e) $|3-5|-|6-2| = |-2| - 4 = 2-4 = -2$ f) $||-6|-|3-12|| = |6-9| = 3$

7.- Escribir sin exponente:

a) 44.44 × 10^4 = 444 400 b) 55.55 × 10^-5 = 0.000 555 5

c) -0.066 × 10^2 = -6.6 d) 0.00077 × 10^-3 = 0.000 000 77

e) 88.99 × 10^0 = 88.99 f) 1.234 × 10^-4 = 0.000123 4

8.- Escribir cada número en forma exponencial normalizada:

a) 333.444 = 3.33444×10^2 b) -1.2345 = -0.12345×10^1

c) 0.0006677 = -0.6677×10^-3 d) -.8899 = -0.8899×10^0

e) 222 000 = 0.222000×10^6 f) -0.03 = -0.3×10^-1

9.- Encontrar la mantisa y el exponente de cada uno de los números del problema 8.

a) 333.444 = 0.333 4444, n = 3
b) -1.2345 = -0.12345, n = 1
c) 0.0006677 = -0.6677, n = -3
d) -.8899 = -0.8899, n = 0
e) 222 000 = 0.222 000, n = 6
f) -0.03 = -0.3, n = -1

10.- Escribir en notación científica cada uno de los números en el problema 8.

a) 333.444 = 3.33444×10^{2}
b) -1.2345 = -1.2345
c) 0.0006677 = 66.77×10^-5
d) -.8899 = -8.899×10^1
e) 222 000 = $222.0\ 00\times10^{3}$
f) -0.03 = -3×10^-2

11.- Encontrar el valor de cada uno de los números escritos en forma E de computador:

a) 222.2E+2 = 22 220
b) -3.3E-03 = -0.0033
c) 0.4E4 = 4000
d) 0.5E-5 = 0.000 005

12.- Dar la forma exponencial normalizada binaria de cada uno de los números binarios:

a) 111.000111 = 0.111000111×2^{3}
b) 0.0011001100 = 0.11001100×2^-2
c) 1010.1010 = -0.10101010×2^4
d) 0.1111 = 0.1111×2^0

En cada uno de los problemas siguientes, suponga que el computador almacena cada número en un lugar de memoria de 32 bits.

13.-Encontrar la representación interna de:

a) 118

0	0	0	…	0	0	1	1	1	0	1	1	0

b) -118

1	1	1	…	1	1	0	0	0	1	0	1	0

14.- Encontrar la representación interna de:

a) 397

0	0	0	…	0	0	1	1	0	0	0	1	1	0	1

b) -397

1	1	1	…	1	1	0	0	1	1	1	0	0	1	1

15.- Encontrar la representación interna de A= -50.375

1	1	0	0	0	1	1	0	1	1	0	0	1	0	0	1	1	0	0	0	…	0	0	0

16.- Encontrar la representación interna de B= 0.09375

0	0	1	1	1	1	0	1	1	1	0	0	0	0	0	0	…	0	0	0

17.- Encontrar la representación interna de C = 0.2

0	0	1	1	1	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	…	1	1	0	0

18.- Supongamos que un exponente n se representa en un campo de 7 bits como sigue. El primer bit se reserva para el signo, 1 para+, 0 para –. En el campo de 6 bits restante, n es representado en su forma binaria si n es positiva, y como su complemento a 2s si n es negativa. Demuestre que ésta es exactamente la misma representación dada al almacenar la característica de 7 bits de n, C = n+64.

C = 7+64 = 71

19.- Si el computador ha sido programado para efectuar aritmética de enteros en punto fijo, ¿Qué valores se obtienen para:

a) 6 + 10, 2-7, 3 × (-5), -4-8, -4 - (6-3) = 16, -5, -15, -12, -7

b) 11/4, -15/3, 8/11, 123/4, -26/8, -5/8 = 2, -5, 0, 30, -3, 0

20.- En la aritmética de enteros, el cociente J/K de dos enteros J y K es ≤ que el cociente usual. ¿Verdadero o falso? Falso, porque para un cociente negativo, el truncamiento generalmente da un número mayor.

21.- Dar los resultados de las 3 adiciones en punto flotante:

a) 0.2233 × 10² + 0.6688 × 10^1 = 0.2901×10²

b) 5.666 + 44.55 = 0.5021×10²

c) 111.77 + 55.666 = 0.1673×10³

22.- Efectuar las siguientes substracciones en punto flotante:

a) 0.9922 × 10^-3 – 0.4477 × 10^-3 = 0.5005 × 10^-3

b) 33.666 – 2.7777 = 0.3088 × 10²

c) 0.8888 × 10² – 0.2222 × 10³ = -0.1333×10^3

23.- Dar los resultados de las siguientes multiplicaciones en punto flotante:

a) (0.5432×10³) × (0.3333×10^-5) = 0.1810×10^-2 b) 222.88 × 1.1177= 0.2488×10³

24.- Dar los resultados de las siguientes divisiones en punto flotante:

a) (0.2233×10^-2) ÷ (0.6611×10³) = 0.3377×10^-5 b) 111.99 ÷ 44.888 = 0.2493×10^1

25.- Dado A= 66.888, encontrar el error absoluto e y el error relativo r si a) A ha sido redondeado a 66.89, b) A ha sido truncado a 66.88.

e = 66.888 – 66.89 = –0.002

El error relativo r es la razón:

$$r = \frac{e}{A} = \frac{-0.002}{66.888} = -0.0000299$$

Así, $|r| = 0.0000299$

e = 66.888 – 66.88 = 0.008

-0.00299%

b) $$r = \frac{e}{A} = \frac{0.008}{66.888} = 0.000119602$$

Así, $|r| = 0.01196\%$

26.- Dado A = 66.888 y B = 66.111. Cuando el computador calcula la diferencia D = A–B (en donde las mantisas han sido truncadas a P = 4 dígitos), ¿Cuáles son los errores absolutos y relativo?

D = A– B = 66.888– 66.111 = 0.777 Đ = 0.77

e = 0.008 r = 0.01196%

27.- Encontrar una cuota al error absoluto cuando se suman dos números aproximados.

$|Ca+b| = |Ca+Cb| \leq |Ca| + |Cb|$

CAPÍTULO 3. Lógica: tablas de verdad

3.1 INTRODUCCIÓN

Un computador puede ser programado para tomar decisiones basadas en si ciertos enunciados – por ejemplo “El número que se ha computado es mayor que 100” – son verdaderos o falsos. A la verdad o falsedad de un enunciado se le llama *valor de verdad*; un enunciado es *verdadero o falso*, pero no ambas cosas. Algunos enunciados son *enunciados compuestos*, es decir, están integrados por subenunciados y varias conectivas.

Ejemplo:

(a) “Las rosas son rojas y las violetas azules” es un enunciado compuesto por los subenunciados “las rosas son rojas” y “las violetas son azules”.

(b) “Él es inteligente o estudia todas las noches” es, implícitamente, un enunciado compuesto por los subenunciados “Él es inteligente” y “estudia todas las noches”.

(c) “¿Para dónde va?” no es un enunciado ya que no es ni verdadero ni falso.

3.2 CONJUNCIÓN, p∧q

Dos enunciados cualesquiera se pueden combinar con la palabra “y” para formar un enunciado compuesto llamado la *conjunción* de los enunciados originales. Simbólicamente, p∧q denota la conjunción de los enunciados p y q, que se lee “p y q”.

La tabla de verdad del enunciado compuesto p∧q está dada por la siguiente tabla:

p	q	p∧q
V	V	V
V	F	F
F	V	F
F	F	F

En este caso, la primera línea es una manera abreviada de decir que si p es verdadero y q es verdadera, entonces p∧q es verdadero. Las otras líneas tienen significados análogos. Observe que p∧q es verdadero solamente en el caso en que ambos subenunciados sean verdaderos.

Ejemplo: Considere los cuatro enunciados siguientes:

(i) París está en Francia y 2+2=4.
(ii) París está en Francia y 2+2=5.
(iii) París está en Inglaterra y 2+2=4.
(iv) París está en Inglaterra y 2+2=5.

Solamente el primer enunciado es verdadero.

3.3 DISYUNCIÓN, p∨q

Dos enunciados pueden combinarse con la palabra “o” para formar un nuevo enunciado que se llama *disyunción* de los dos enunciados originales: Simbólicamente, p∨q denota la disyunción de los enunciados p y q y se lee “p o q”. El valor de verdad de p∨q está dado por la siguiente tabla de verdad, que consideramos define a p∨q:

p	q	pvq
V	V	V
V	F	V
F	V	V
F	F	F

Observe que pvq es falso solamente cuando ambos enunciados son falsos.

Ejemplo: Considere los cuatro enunciados:

(i) París está en Francia o 2+2=4.
(ii) París está en Francia o 2+2=5.
(iii) París está en Inglaterra o 2+2=4.
(iv) París está en Inglaterra o 2+2=5.

Solamente (iv) es falso.

3.4 NEGACIÓN, ∼P

Dado cualquier enunciado p, se puede formar otro enunciado, llamando la *negación* de p, escribiendo “Es falso que…” antes de p o, si es posible insertando en p la palabra “no”. Simbólicamente

∼P denota la negación de p (se lee “no p”).

La tabla de verdad de ∼P está dada por la siguiente tabla:

p	∼P
V	F
F	V

En otras palabras, si p es verdadero entonces ∼P es falso, y si p es falso entonces ∼P es verdadero.

Ejemplo: Considere los siguientes enunciados:

(a) París está en Francia.	(d) 2+2=5.
(b) Es falso que París está en Francia.	(e) Es falso que 2+2=5.
(c) París no está en Francia.	(f) 2+2≠5.

Entonces (b) y (c) son cada una la negación de (a); y (e) y (f) son cada uno la negación de (d). Ya que (a) es verdadero, los enunciados (b) y (c) son falsos; y como (d) es falso, los enunciados (e) y (f) son verdaderos.

EJERCICIOS RESUELTOS DEL 3.2 al 3.4

1.- Sea p “Hace frío” y sea q “Está lloviendo”. De una frase verbal sencilla que describa cada uno de los siguientes enunciados:

(1) ∼P, (2) p∧q, (3) p ∨q, (4) q∨∼P

RESPUESTA:

(1) No hace frío.
(2) Está haciendo frío y está lloviendo.
(3) Está haciendo frío o está lloviendo.
(4) Está lloviendo o no está haciendo frío.

2.- Sea p “Él es alto” y sea q “Él es buen mozo”. Escriba cada uno de los siguientes enunciados en forma simbólica usando p y q.

(1) p∧q (2) p∧∼q (3) ∼ (∼p ∨q) (4) ∼p∧∼q

RESPUESTA:

(1) Él es alto y buen mozo.
(2) Él es alto pero no buen mozo.
(3) Es falso que él sea bajo o buen mozo.
(4) Él no es ni alto ni buen mozo.

3.5 PROPOSICIONES Y TABLAS DE VERDAD

Con un uso repetido de las conectivas lógicas (∨, ∧, ~ y otras que se discutirán adelante), podemos construir enunciados compuestos que son más elaborados. En el caso en el que los subenunciados p, q,… de un enunciado compuesto P (p, q,…) sean variables, llamamos al enunciado compuesto un *proposición*.

La tabla de verdad de la proposición ~ (p∧~q), por ejemplo, se construye como sigue:

p	q	~q	p ∧~q	~ (p∧~q)
V	V	F	F	V
V	F	V	V	F
V	V	F	F	V
V	F	V	F	V

Observe que las primeras columnas de la tabla son para las variables p, q,… y que hay suficientes líneas en la tabla para permitir todas las posibles combinaciones de V y F para estas variables. Hay pues una columna para cada etapa “elemental” de la construcción del enunciado el valor de verdad de cada paso es determinado por las etapas anteriores con las definiciones de las conectivas ∨, ∧, ~. Finalmente, obtenemos la tabla de verdad de la proposición, que aparece en la última columna.

Observación: La tabla de verdad de la proposición anterior consiste precisamente en las columnas bajo las variables y la columna bajo la proposición:

p	Q	~ (p∧~q)
V	V	V
V	F	F
F	V	V
F	F	V

Las otras columnas se usaron solamente en la construcción de la tabla de verdad.

EJERCICIOS RESUELTOS DEL 3.5.

1.- Encuentre la tabla de verdad de ~p∧q.

P	q	~p	~p∧q
V	V	F	F
V	F	F	F
F	V	V	V
F	F	V	F

2.- Encuentre la tabla de verdad de ~ (p ∨q).

p	q	p ∨q	~ (p ∨q)
V	V	V	F
V	F	V	F
F	V	V	F
F	F	F	V

3.6 TAUTOLOGIAS Y CONTRADICCIONES

Algunas proposiciones P (p, q,…) contienen solamente V en la última columna de sus tablas de verdad, es decir, son verdaderas para cualquier valor de verdad de sus variables. A tales proposiciones se les llama *tautologías*. Análogamente, una proposición p (p, q,…) se llama *contradicción* si contiene solamente F en la última columna de su tabla de verdad. Esto se verifica construyendo sus tablas de verdad.

P	~p	p ∨~p
V	F	V
F	V	V

p	~p	P∧~p
V	F	F
F	V	F

Observamos que la negación de una tautología es una contradicción ya que siempre es falsa, y la negación de una contradicción es una tautología ya que siempre es verdadera.

Principio de sustitución: Si P (p, q,…) es una tautología, entonces P (P_1, P_2,...) es una tautología para proposiciones cualesquiera P_1, P_2,...

Ejemplo: Por la anterior tabla de verdad, p ∨~p es una tautología. Reemplazando p por q∧r obtenemos la proposición (q∧r) ∨~ (q∧r) que por el principio de sustitución, también debería de ser una tautología. Esto se verifica con la siguiente tabla de verdad:

q	r	q∧r	~ (q∧r)	(q∧r) ∨~ (q∧r)
V	V	V	F	V
V	F	F	V	V
F	V	F	V	V
F	F	F	V	V

EJERCICIOS RESUELTOS 3.6.

1.- Verifique que la proposición p∨~ (p∧q) es una tautología.

p	q	(p∧q)	~ (p∧q)	P∨~ (p∧q)
V	V	V	F	V
V	F	F	V	V
F	V	F	V	V
F	F	F	V	V

Como la tabla de verdad de p∨~ (p∧q) es V para todos los valores de verdad de p y q, entonces es una tautología.

2.- Verifique que la proposición (p∧q) ∧~ (p∨q) es una contradicción.

p	q	p∧q	P∨q	~ (p∨q)	(p∧q) ∧~ (p∨q)
V	V	V	V	F	F
V	F	F	V	F	F
F	V	F	V	F	F
F	F	F	F	V	F

Como la tabla de verdad de (p∧q) ∧~ (p∨q) es una F para todos los valores de verdad de p y q, entonces es una contradicción.

3.7 EQUIVALENCIA LÓGICA: ALGEBRA DE PROPOSICIONES

Se dice que dos proposiciones P (p, q,…) y Q (p, q,…) son *lógicamente equivalentes*, o sencillamente *equivalentes* o *iguales*, denotado por

P (p, q,…) ≡ Q (p, q,…) si tienen idénticas tablas de verdad. Por ejemplo, considérese las tablas de verdad de ~ (p∧q) y ~ p∨~ q:

p	q	(p∧q)	~ (p∧q)
V	V	V	F
V	F	F	V
F	V	F	V
F	F	F	V

p	q	~ p	~ q	~ p∨~ q
V	V	F	F	F
V	F	F	V	V
F	V	V	F	V
F	F	V	V	V

Como las tablas de verdad son las mismas, mejor dicho, ambas proposiciones son falsas en el primer caso y verdaderas en los otros tres casos, las proposiciones ~ (p∧q) y ~ p∨~ q son lógicamente equivalentes y podemos escribir:

$$\sim (p \wedge q) \equiv \sim p \vee \sim q$$

Ejemplo: Considere el enunciado

"Es falso que las rosas son rojas y las violetas son azules".

Este enunciado se puede escribir en la forma ~ (p∧q) en donde p es "las rosas son rojas" y q es "las violetas son azules". Sin embargo, por las tablas de verdad anteriores, ~ (p∧q) es lógicamente equivalente con ~ p∨~ q. Así, el enunciado dado tiene el mismo significado que el enunciado:

"Las rosas no son rojas, o las violetas no son azules."

3.8 ENUNCIADOS CONDICIONAL Y BICONDICIONAL

Muchos enunciados, particularmente en la matemática, son de la forma “Si p entonces q”. Tales enunciados se llaman enunciados *condicionales* y se denotan por:

Tabla 4-1 Leyes del álgebra de proposiciones.

Leyes de idempotencia	
1a. $p \vee p \equiv p$	1b. $p \wedge p \equiv p$
Leyes asociativas	
2a. $(p \vee q) \vee r \equiv p \vee (q \vee r)$	2b. $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Leyes conmutativas	
3a. $p \vee q \equiv q \vee p$	3b. $p \wedge q \equiv q \wedge p$
Leyes distributivas	
4a. $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	4b. $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Leyes de identidad	
5a. $p \vee f \equiv p$	5b. $p \wedge t \equiv p$
6a. $p \vee t \equiv t$	6b. $p \wedge f \equiv f$
Leyes de complementos	
7a. $p \vee \sim p \equiv t$	7b. $p \wedge \sim p \equiv f$
8a. $\sim t \equiv f$	8b. $\sim f \equiv t$
Ley de involución	
9. $\sim\sim p \equiv p$	
Leyes de DeMorgan	
10a. $\sim (p \vee q) \equiv \sim p \wedge \sim q$	10b. $\sim (p \wedge q) \equiv \sim p \vee \sim q$

El condicional $p \rightarrow q$ frecuentemente se lee “p implica q” o “p sólo si q”.

Otro enunciado común es la forma “p si y sólo si q”. Tales enunciados denotados por:

$p \leftrightarrow q$ se llaman enunciados *bicondicionales*.

Los valores de verdad de $p \rightarrow q$ y $p \leftrightarrow q$ se dan en las siguientes tablas

p	q	$p \rightarrow q$	$p \leftrightarrow q$
V	V	V	V
V	F	F	F
F	V	V	F
F	F	V	V

Ahora considérese la tabla de verdad de la proposición $\sim pvq$:

P	q	$\sim p$	$\sim pvq$
V	V	F	V
V	F	F	F
F	V	V	V
F	F	V	V

Obsérvese que la anterior tabla de verdad es idéntica a la tabla de verdad de $p \rightarrow q$. Así que $p \rightarrow q$ es lógicamente equivalente a la proposición $\sim pvq$:

$$p \rightarrow q \equiv \sim pvq$$

Considere la proposición condicional $p \rightarrow q$ y las otras proposiciones condicionales simples que contienen p y q:

$$q \rightarrow p, \quad \sim p \rightarrow \sim q \quad y \quad \sim q \rightarrow \sim p$$

Estas proposiciones se llaman respectivamente *recíproca*, *inversa*, y *contra recíproca* de la proposición $p \rightarrow q$. En seguida presentamos las tablas de verdad de las cuatro proposiciones.

p	q	Condicional $p \rightarrow q$	Recíproca $q \rightarrow p$	Inversa $\sim p \rightarrow \sim q$	Contra recíproca $\sim q \rightarrow \sim p$
V	V	V	V	V	V
V	F	F	V	V	F
F	V	V	F	F	V
F	F	V	V	V	V

Observe que un enunciado condicional y su recíproco o inverso no son lógicamente equivalentes.

Teorema: Un enunciado condicional $p \rightarrow q$ y su contra recíproca $\sim q \rightarrow \sim p$ son lógicamente equivalentes.

Ejemplo:

(a) Considere los siguientes enunciados sobre un triángulo A:

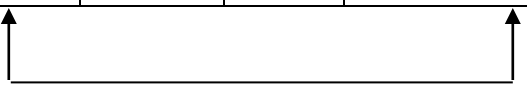
$p \rightarrow q$: Si A es equilátero, entonces A es isósceles.
$q \rightarrow p$: Si A es isósceles, entonces A es equilátero.

En este caso $p \rightarrow q$ es verdadero, pero su recíproco $q \rightarrow p$ es falso.

EJERCICIOS RESUELTOS DEL 3.7 al 3.8.

1.- Demuestre que la disyunción distribuye sobre la conjunción, o sea, demuestre la ley distributiva $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

P	q	r	$q \wedge r$	$p \vee (q \wedge r)$	$p \vee q$	$p \vee r$	$(p \vee q) \wedge (p \vee r)$
V	V	V	V	V	V	V	V
V	V	F	F	V	V	V	V
V	F	V	F	V	V	V	V
V	F	F	F	V	V	V	V
F	V	V	V	V	V	V	V
F	V	F	F	F	V	F	F
F	F	V	F	F	F	V	F
F	F	F	F	F	F	F	F



Como las tablas de verdad son idénticas, las proposiciones son equivalentes.

2.- Simplifique la proposición usando la tabla 4-1:

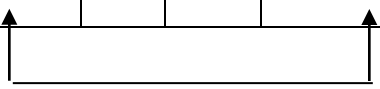
(a) p∨ (p∧q).

Equivalencia	Razón
(1) p∨ (p∧q) ≡ (p∧t) ∨ (p∧q)	(1) Ley de identidad.
(2) ≡ p∧ (t∨q)	(2) Ley distributiva.
(3) ≡ p∧t	(3) Ley de identidad.
(4) ≡ p	(4) Ley de identidad.

3.- Demuestre las leyes de Morgan:

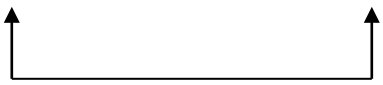
(a) ~ (p∧q) ≡ ~p∨~q

p	q	p∧q	~ (p∧q)	~p	~q	~p∨~q
V	V	V	F	F	F	F
V	F	F	V	F	V	V
F	V	F	V	V	F	V
F	F	F	V	V	V	V



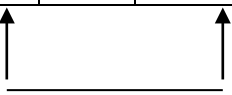
(b) ~ (p∨q) ≡ ~p∧~q

p	Q	p∨q	~ (p∨q)	~p	~q	~p∧~q
V	V	V	F	F	F	F
V	F	V	F	F	V	F
F	V	V	F	V	F	F
F	F	F	V	V	V	V



4.- Verifique: ∼∼p ≡ p.

p	∼p	∼∼p
V	F	V
F	V	F



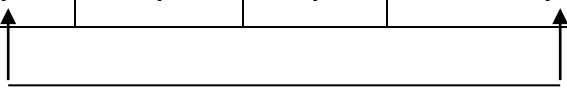
5.- Escriba de nuevo el siguiente enunciado sin usar la condicional.

(a) Si hace frío, él se pone sombrero.

R: No está haciendo frío o él se pone el sombrero.

6.- Demuestre que “p implica q y q implica p” es lógicamente equivalente con la bicondicional “p si y sólo si q”; o sea, (p → q) ∧ (q → p) ≡ p ↔ q.

p	q	p ↔ q	p → q	q → p	(p → q) ∧ (q → p)
V	V	V	V	V	V
V	F	F	F	V	F
F	V	F	V	F	F
F	F	V	V	V	V



7.- Demuestre que p ↔ q ≡ (p∨q) → (p∧q) por álgebra de proposiciones

Equivalencias	Razón
(1) p ↔ q ≡ (∼p∨q) ∧ (∼q∨p)	
(2) ≡ [(∼p∨q) ∧∼q] ∨ [(∼p∨q) ∧p]	(2) Ley distributiva
(3) ≡ [(∼q∧q) ∨ (∼q∧∼p)] ∨ [(p∧q) ∨ (p∧∼p)]	(3) Leyes conmutativa y distributiva.
(4) ≡ [f∨ (∼q∧∼p)] ∨ [(p∧q) ∨f]	(4) Ley de complemento.
(5) ≡ (∼q∧∼p) ∨ (p∧q)	(5) Ley de identidad.
(6) ≡ [∼ (p∨q)] ∨ (p∧q)	(6) Ley De Morgan.
(7) ≡ (p∨q) → (p∧q)	(7) Sección 4.8.

8.- Determina la contra recíproca de cada enunciado.

(a) Si Juan es poeta, entonces es pobre.
R: La contra recíproca de $p \rightarrow q$ es $\sim q \rightarrow \sim p$. Así que la contra recíproca es:
Si Juan no es pobre, entonces no es poeta.

(b) Solamente si Marcos estudia pasará el examen.
R: El enunciado dado es equivalente a "Si Marcos pasa el examen, entonces estudió". Así que la contra recíproca es:
Si Marcos no estudia, entonces no pasará el examen.

3.9 ARGUMENTOS

Un *argumento* es una relación entre un conjunto de proposiciones P_1, P_2,..., Pn, llamadas *premisas*, y otra proposición Q, llamada la conclusión; denotamos un argumento por:

$P_1, P_2, ..., Pn \vdash Q$

Definición: Un argumento $P_1, P_2, ..., Pn \vdash Q$ es válido si Q es verdadero cada vez que las premisas P_1, P_2,..., Pn sean verdaderas.

Un argumento que no es válido se llama *falacia*.

Ejemplo:

(a) El siguiente argumento es válido:

$p, p \rightarrow q \vdash q$ (Ley de independencia, *modus ponendo ponens.*)

Teorema: El argumento P_1, P_2,..., Pn $\vdash$ Q es válido si y sólo si ($P_1 \wedge P_2 \wedge ... \wedge$ Pn) $\rightarrow$Q es una tautología.

En el siguiente ejemplo aplicamos este teorema.

Ejemplo 4. Un principio fundamental del racionamiento lógico dice:

“Si p implica q y q implica r, entonces p implica r”.

O sea, el siguiente argumento es válido:

$p \rightarrow q, \quad q \rightarrow r \vdash p \rightarrow r$ (Ley del Silogismo).

Este hecho se verifica con la siguiente tabla de verdad que muestra que la proposición

$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$.

p	q	r	[(p	→	q)	∧	(q	→	r)]	→	(p	→	r)
V	V	V	V	V	V	V	V	V	V	V	V	V	V
V	V	F	V	V	V	F	V	F	F	V	V	F	F
V	F	V	V	F	F	F	F	V	V	V	V	V	V
V	F	F	V	F	F	F	F	V	F	V	V	F	F
F	V	V	F	V	V	V	V	V	V	V	V	V	V
F	V	F	F	V	V	F	V	F	F	V	V	V	F
F	F	V	F	V	V	V	F	V	V	V	V	V	V
F	F	F	F	V	V	V	F	V	F	V	V	V	F
Paso			1	2	1	3	1	2	1	4	1	2	1

Es necesario destacar que la validez del argumento no depende de los valores de verdad o del contenido de los enunciados que aparecen en el argumento, si no

solamente de la estructura formal del argumento. Esto se ilustra en el siguiente ejemplo.

Ejemplo: Considere el siguiente argumento:

S_1: Si un hombre es soltero, es infeliz.
S_2: Si un hombre es infeliz, muere joven.
S: Los solteros mueren jóvenes.

En este caso el enunciado S debajo de la línea denota la conclusión del argumento, y los enunciados S_1 y S_2 por encima de la línea denotan las premisas. Afirmamos que el argumento S_1, $S_2 \vdash S$ es válido. Y que el argumento es de la forma $p \rightarrow q, \quad q \rightarrow r \vdash p \rightarrow r$ en donde p es “Él es soltero”, q es “Él es infeliz” y r es “El muere joven”, y por el ejemplo anterior este argumento (Ley de Silogismo) es válido.

3.10 IMPLICACIÓN LÓGICA

Se dice que un proposición P (p. q,…) *implica lógicamente* un proposición Q (p, q,…), escrito P (p, q,…) ⇒ Q (p, q,…)

Si Q (p, q,…) es verdadera cada vez que P (p, q,…) sea verdadera.

Teorema: Para proposiciones cualesquiera P (p, q,…) y Q (p, q,…) los tres enunciados siguientes son equivalentes:

(i) P (p, q,…) implica lógicamente a Q (p, q,…).
(ii) El argumento P (p, q,…) ⊢ Q (p, q,…) es válido.
(iii) La proposición p (p, q,…) → Q (p, q,…) es una tautología.

Si P ⇒ Q y Q ⇒ P, entonces P y Q deben de tener la misma tabla de verdad, y por lo tanto P ≡ Q. El recíproco también es cierto. Así, la noción de implicación lógica está íntimamente ligada a la equivalencia lógica.

EJERCICIOS RESUELTOS DEL 3.9 AL 3.10.

1.- Verifique la validez del siguiente argumento:

Si estudio, no perderé matemáticas.
Si no juego básquetbol, entonces estudio.
Pero perdí matemáticas.
Por lo tanto, jugué básquetbol.

Primero traduzca a su forma simbólica. Sean p “Yo estudio”, q “Pierdo matemáticas” y r “Juego básquetbol”. Entonces el argumento dado es como sigue:
$p \rightarrow \sim q,\ \ \sim r \rightarrow p,\ q \vdash r$

Para verificar la validez del argumento, construya las tablas de verdad de las proposiciones dadas $p \rightarrow \sim q,\ \ \sim r \rightarrow p,\ q \vdash r$:

p	q	r	~q	p →~q	~r	~r → p
V	V	V	F	F	F	V
V	V	F	F	F	V	V
V	F	V	V	V	F	V
V	F	F	V	V	V	V
F	V	V	F	V	F	V
F	V	F	F	V	V	F
F	F	V	V	V	F	V
F	F	F	V	V	V	F

2.- Demuestre que $p \leftrightarrow \sim q$ no implica lógicamente $p \rightarrow q$.

Construya las tablas de verdad de $p \leftrightarrow \sim q$ y $p \rightarrow q$

p	q	~q	$p \leftrightarrow \sim q$	$p \rightarrow q$
V	V	F	F	V
V	F	V	V	F
F	V	F	V	V
F	F	V	F	V

CAPÍTULO 4. Simplificación de circuitos lógicos

4.1 Expresiones Booleanas Minimales

Considere una expresión *E* en un álgebra de Boole. Como *E* puede representar un circuito lógico, es posible que queramos una representación de *E* que en algún sentido sea minimal. Ahora definimos e investigamos las formas minimales de sumas de productos para *E*.

Si *E* es una expresión de Boole de suma de productos, *EL* denotará el número de literales en *E*, y *Es* denotará el número de sumandos en *E*. Por ejemplo, si

$$E = abc' + a'b'd' + ab'c'd + a'bcd$$

Entonces *EL* = 14 y *Es* = 4. Sea ahora *F* una expresión de Boole de suma de productos equivalente a *E*. Decimos que *E* es más simple que *F* si

$$EL \leq FL \qquad Es \leq Fs$$

y por lo menos una de las relaciones es una desigualdad estricta.

Definición: Una expresión de Boole está en forma minimal de suma de productos (o sencillamente, es una suma minimal) si está en forma de suma de productos y no hay ninguna otra expresión equivalente en forma de suma de productos que sea más simple que *E*.

Un producto fundamental *P* se llama *implicante primo* de una expresión de Boole *E* si

$P + E = E$

Teorema: Si una expresión de Boole *E* está en forma minimal de suma de productos, entonces cada sumando de *E* es un implicante primo de *E*.

El así llamado *método de consenso,* discutido en los problemas 8.3 y 8.4, se puede usar para representar cualquier expresión de Boole como la suma de todos sus implicantes primos. Una manera de encontrar una suma minimal para *E* es expresar cada implicante primo en forma completa de suma de productos, y quitar un por uno aquellos implicantes primos cuyos sumandos aparecen entre los sumandos de los implicantes primos que quedan. Por ejemplo, demostramos en el problema 8.4 que

E = x'z' + xy + x'y' + yz'

está expresado como la suma de todos sus implicantes primos. Tenemos

x'z' = x'z' (y + y') = x'yz' + x'y'z'
xy = xy (z + z') = xyz + xyz'
x'y' = x'y' (z + z') = x'y'z + x'y'z'
yz' = yz' (x + x') = xyz' + x'yz'

Ahora se puede quitar x'z', puesto que sus sumandos x'yz' y x'y'z', aparecen entre los otros. Así que

E = xy + x'y' + yz'

y esto está en forma de suma minimal para *E*, ya que ninguno de los implicantes primos es superfluo, es decir, se puede quitar sin cambiar *E*.

EJERCICIOS RESUELTOS 4.1.

1.- Encuentre *EL*, el número de literales y *Es*, el número de sumandos, para cada expresión de Boole *E*:

(a) *E* = xy'z + x'z' +yz' + x (b) *E* = x'y'z + xyz + y + yz' + x'z

R: (a) *EL* = 3 + 2 + 2 + 1 = 8 *Es* = 4 (b) *EL* = 3 + 3 + 1 + 2 +2 = 11 *Es* = 5

2.- Sean F_1 y F_2 productos fundamentales, tales que exactamente una variable, por ejemplo Xk, aparezca complementada en sólo uno de P_1 y P_2 y no complementada en el otro. El *consenso* de P_1 y P_2 es, entonces, el producto (sin repetición) de los literales de P_1 y los literales de P_2 después de que Xk y X'k sean suprimidas. (No definimos un consenso de P_1= X y P_2 = X'.)

(a) Encuentre el consenso de:

(1) xyz's y xy't (2) xy' y

R: (1) xz'st (2) x

3.- Use el método de consenso para encontrar los implicantes primos y una suma minimal para *E* = xy + y't + x'yz' + xy'zt'

E = xy + y't + x'yz' + xy'zt' + xzt' (Consenso de xy y xy'zt')

= xy + y't + x'yz' + xzt' (xy'zt' incluye a xzt')
= xy + y't + x'yz' + xzt' + yz' (Consenso de xy y x'yz')
= xy + y't + xzt' + yz' (x'yz' incluye a yz')
= xy + y't + xzt' + yz' + xt (Consenso de xy y y't)
= xy + y't + xzt' + yz' + xt + xz (Consenso de xzt' y xt)
= xy + y't + yz' + xt +xz (xzt' incluye a xz)
= xy + y't + yz' + xt +xz (Consenso de y't y yz')

Ningún paso en el método de consenso se puede aplicar ahora. Así, los implicantes primos de *E* son xy, y't, yz', xt, xz y z't. Escribiendo estos implicantes

en forma completa de suma de productos y suprimiendo uno por uno aquellos que son superfluos, finalmente obtenemos E = y't + xz + yz' como una suma minimal para E.

4.2 MAPAS DE KARNAUGH

Los *mapas de Karnaugh* son maneras pictóricas de encontrar implicantes primos y formas minimales para las expresiones de Boole que involucran máximo seis variables.

En nuestros mapas de Karnaugh, se representarán por cuadrados los productos fundamentales en las mismas variables. Decimos que dos de tales productos fundamentales P_1 y P_2 son *adyacentes* si P_1 y P_2 difieren en exactamente un literal, lo cual tiene que ser una variable completada en un producto y no completada en otro. Así que la suma de dos productos adyacentes será un producto fundamental con un literal menos. Por ejemplo,

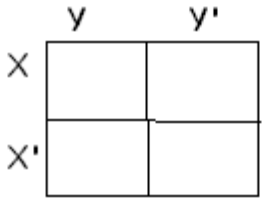
xyz' + xy'z' = xz' (y + y') = xz' (1) = xz'
x'yzt + x'yz't = x'y (z + z') = x'yt (1) = x'yt

Caso de dos variables

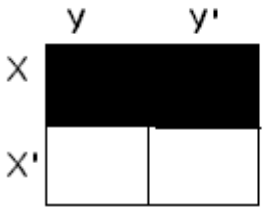
El mapa de Karnaugh que corresponde a las expresiones de Boole E(x, y) se visualiza en la Fig. (1) (a). Podemos ver el mapa de Karnaugh como un diagrama de Venn en el que x está representado por los puntos en la mitad superior del mapa, sombreada de la Fig. (1) (b), y *y* está representado por los puntos en la mitad izquierda del mapa, sombreada de la Fig. (1) (c). Así que x' está representado por los puntos de la mitad derecha del mapa. De esta manera, los cuatro posibles productos fundamentales con dos literales,

xy xy' x'y x'y'

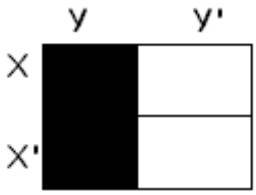
están representados por los cuatro cuadrados por el mapa, tal como se ha rotulado en la Fig. (1) (d). Observe que dos de tales cuadrados son adyacentes en el sentido definido anteriormente si y sólo si están geométricamente adyacentes (tienen un lado en común).



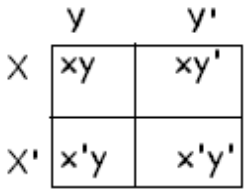
(a)



(b) x sombreado



(c) y sombreado



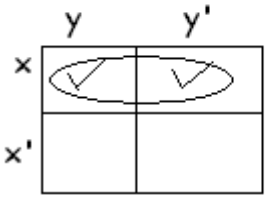
(d)

Figura (1)

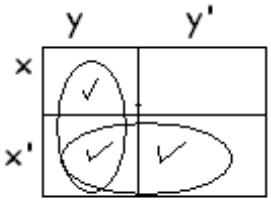
Cualquier expresión de Boole completa de suma de productos *E* (x, y) está representada en un mapa de Karnaugh marcando los cuadrados apropiados. Por ejemplo,

$E_1 = xy + xy'$ $E_2 = xy + x'y + x'y'$ $E_3 = xy + x'y'$

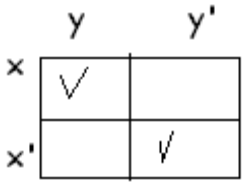
están representados respectivamente en la Fig. 8-2 (a), (b), y (c). (Los óvalos se explicaran más adelante),



(a) E_1



(b) E_2



(c) E_3 Figura (2)

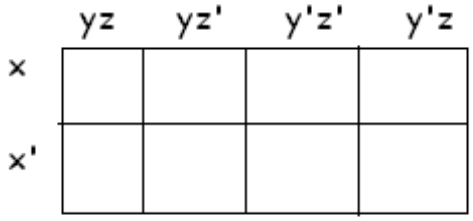
- El E_1 consta de dos cuadrados adyacentes designados por el óvalo en la Fig. (2) (a); E_1 = x, es su suma minimal.
- El E_2 contiene dos parejas de cuadrados adyacentes que incluyen todos los cuadrados de E_2; E_2 = x' + x'y', es su suma minimal.
- El E_3 está formado por dos cuadrados aislados que representan xy y x'y'; así que xy y x'y' son implicantes primos de E_3 y; E_3 = xy + x'y', es una suma minimal.

Caso de tres variables

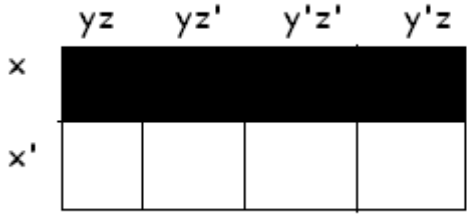
El mapa de Karnaugh que corresponde a las expresiones de Boole *E* (x, y, z) se representa en la figura 8-3. Observe que hay exactamente ocho productos fundamentales con tres literales,

xyz xyz' xy'z xy'z' x'yz x'yz' x'y'z x'y'z'

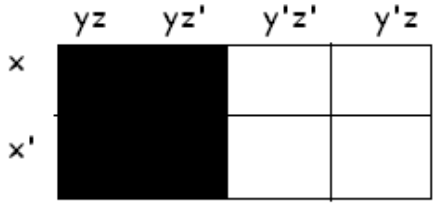
Por un rectángulo básico en el mapa de Karnaugh de tres variables, figura (3) o Fig. (4), queremos decir un cuadrado, dos cuadrados adyacentes, o cuatro cuadrados que forman un



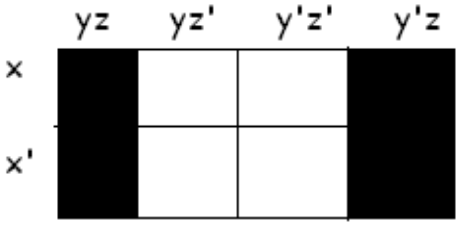
(a)



(b) x sombreado



(c) y sombreado



(d) z sombreado

Figura (3)

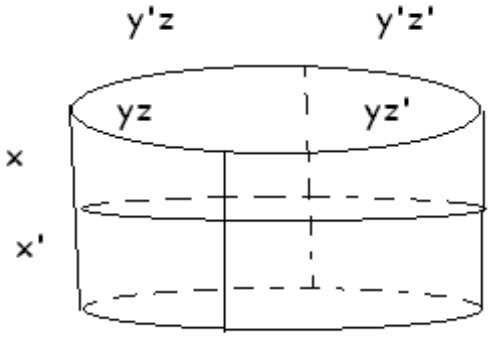


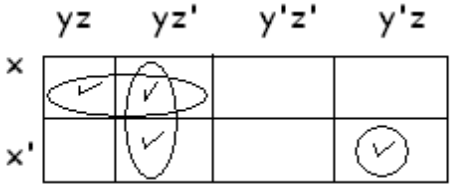
Figura (4)

rectángulo de uno por cuatro o dos por dos. Estos rectángulos básicos corresponden a los productos fundamentales de tres, dos, y un literal, respectivamente. Además, el producto de exactamente aquellos literales que aparecen en cada cuadrado del rectángulo.

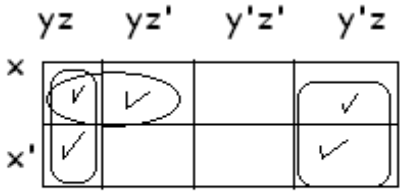
EJEMPLO: Considere las tres expresiones de Boole completas siguientes de suma de productos en las variables x, y, z:

E_1 = xyz + xyz' + x'yz' + x'y'z
E_2 = xyz + xyz' + xy'z + x'yz + x'y'z
E_3 = xyz + xyz' + x'yz' + x'y'z' + x'y'z

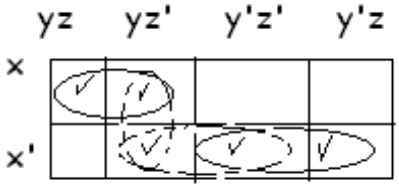
E_1, E_2, y E_3 están representados en la Fig. (5) marcando los cuadrados apropiados en los mapas de Karnaugh. Mostramos cómo usar estos mapas para encontrar las sumas minimales para las expresiones; la suma minimal para E_1, E_2, E_3 son: E_1 = xy + yz' + x'y'z, E_2 = xy + z, E_3 = xy + yz' + x'y' = xy + x'z' + x'y'



(a) E_1



(b) E_2



(c) E_3 Figura (5).

Caso de cuatro variables

El mapa de Karnaugh que corresponde a las expresiones de Boole *E* (x, y, z, t) está representado en la Fig. (6). Cada uno de los dieciséis cuadros del mapa corresponde a uno de los dieciséis productos fundamentales.

xyzt xyzt' xyz't' xyz't xy'zt … x'yzt'

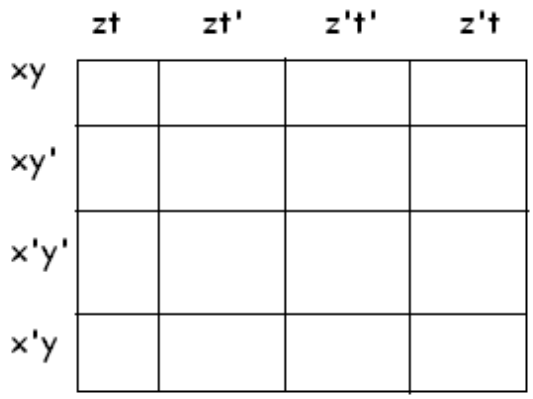
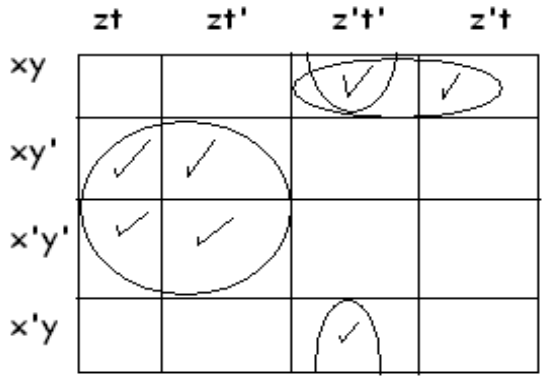


Figura (6)

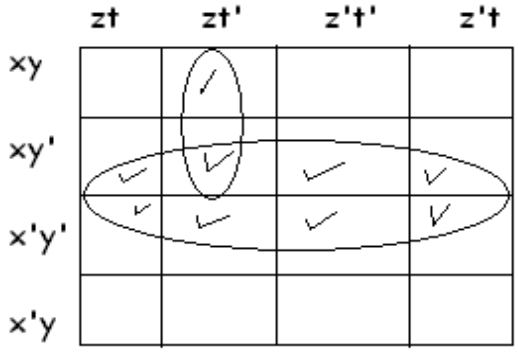
EJEMPLO: Considere las tres expresiones de Boole E_1, E_2, E_3 en las variables x, y, z, t que están dadas en los mapas de Karnaugh en la Fig. 8-7, por ejemplo,

E_1 = xyz't' + xyz't + xy'zt + xy'zt' + x'y'zt + x'y'zt' + x'yz't

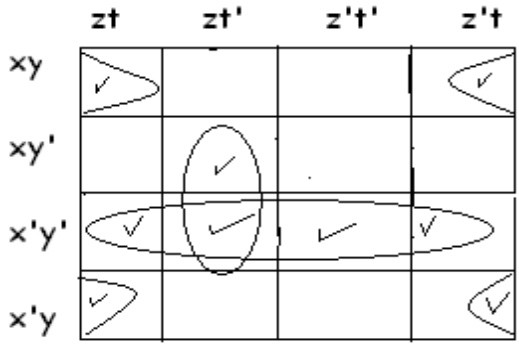
Usamos estos mapas para encontrar las formas de suma minimal.



(a) E_1



(b) E_2



(c) E_3

E_1 = y'z + xyz' + yz't' es la suma minimal para E_1; E_2 = y' + xzt' es la suma minimal para E_2; E_3 = yt + x'y' + y'zt' es la suma minimal para E_3.

Un recubrimiento minimal del mapa consiste en los tres rectángulos básicos maximales designados. Así que E = xz + y'z' + yzt' es una suma minimal para E.

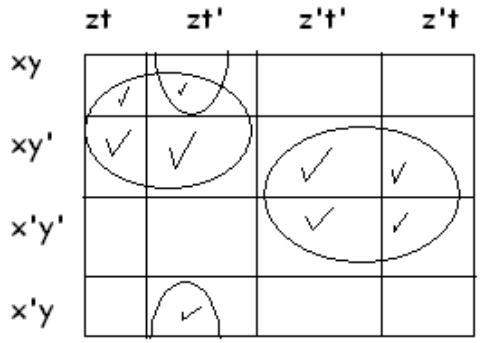
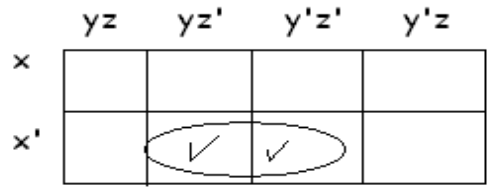


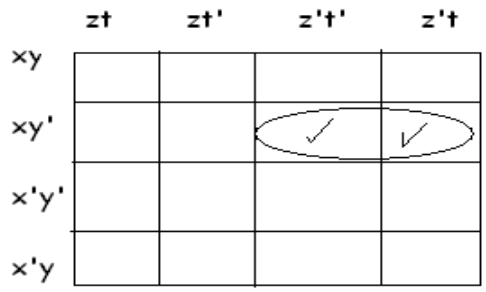
Figura (8)

EJERCICIOS 4.2

1.- Encuentre el producto fundamental P representado por cada rectángulo básico en los mapas de Karnaugh que se muestran en la Fig. (11).



(a)



(b)

R: (a) x' y z' aparecen en ambos cuadrados; así F = x'z'-

(b) x, y', z' aparecen en ambos cuadrados; así P = xy'z'.

2.- Use un mapa de Karnaugh para encontrar una suma minimal para

E = y't' + y'z' + yzt' + x'y'zt.

Marque los cuatro cuadrados correspondientes al producto fundamental y't', los cuatro cuadrados correspondientes a y'zt', y el cuadrado correspondiente a x'y'zt. Esto da el mapa de Karnaugh en la Fig. 8-14. Un recubrimiento minimal consta de los tres rectángulos básicos maximales marcados.

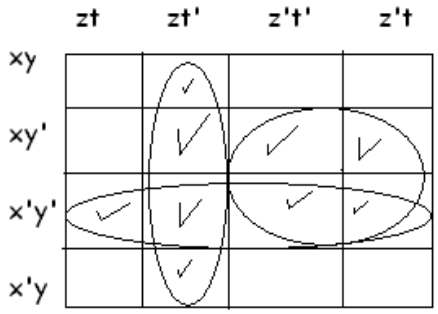


Figura (14)

Así E = zt' + y'z' + x'y' e suna suma minimal para E.

4.3 CIRCUITOS MINIMALES AND-OR

La construcción de un circuito minimal AND-OR que tendrá una tabla de verdad prescrita.

EJEMPLO: Diseñe un circuito minimal AND-OR L de tres entradas que tenga la tabla de verdad.

A	00001111
B	00110011
C	01010101
L	11001101

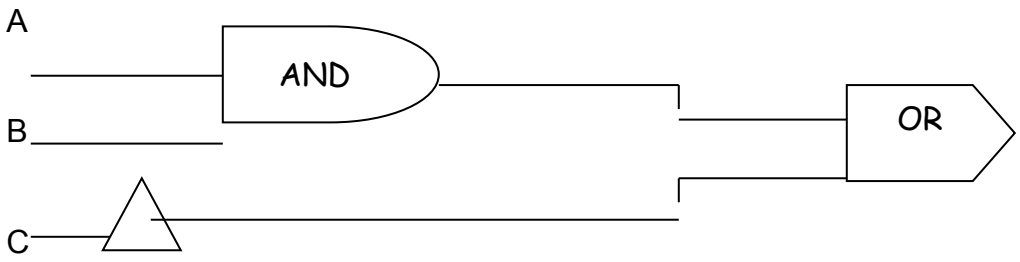
De la tabla de verdad podemos obtener la fórmula completa de suma de productos para L:

$$L = A' * B' * C + A' * B * C' + A * B' *C' + A * B *C' + A* B *C$$

Observe que L tiene dos implicaciones primos B y AC, en su recubrimiento minimal; así que

$$L = B' + AC$$

es una suma minimal para L. La Fig. 8-9 da el correspondiente circuito minimal AND-OR para L.



EJERCICIO DEL 4.3.

1.- Dibuje un circuito AND-OR minimal que dé la siguiente tabla de verdad.

A	00001111
B	00110011
C	01010101
L	10101001

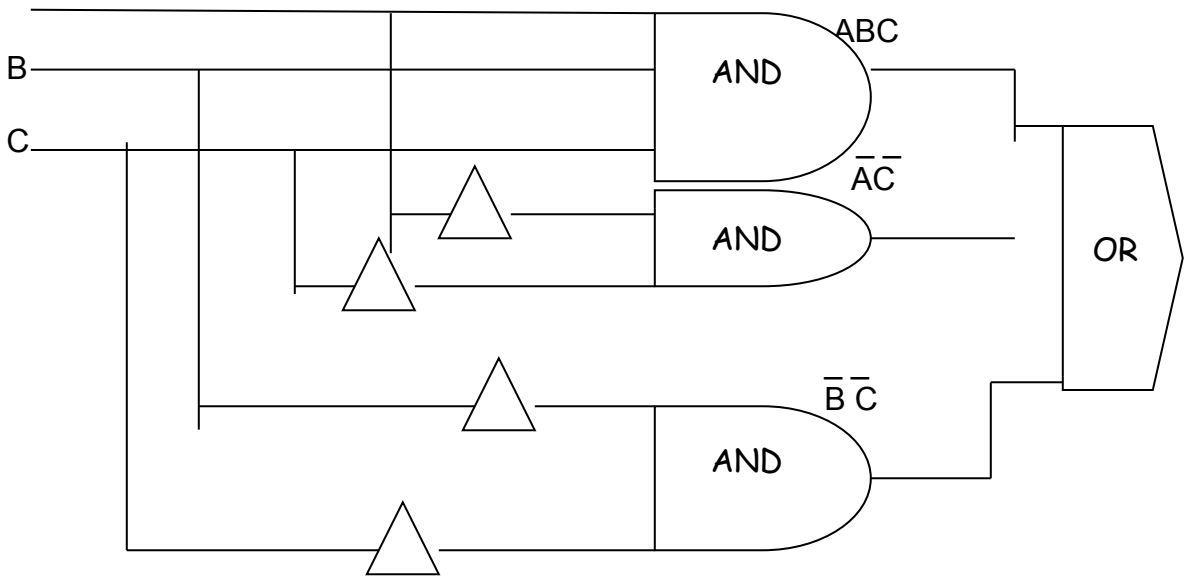
De la tabla de verdad obtenemos la representación como suma de productos.

$$L = \bar{A} * B * \bar{C} + A * \bar{B} * \bar{C} + A * \bar{B} * C + A * B * C$$

El mapa de Karnaugh de L. Hay tres implicantes primos, como se indica con los tres óvalos. Así

$$L = \bar{A}BC + A\bar{C} + B\bar{C}$$

es una suma minimal para L; el correspondiente circuito AND-OR minimal aparece en la Fig. 8-15.



CAPÍTULO 5. ÁLGEBRA DE BOOLE, COMPUERTAS LÓGICAS

5.1 INTRODUCCIÓN

Tanto los conjuntos como las proposiciones tienen propiedades similares. Estas propiedades se usan para definir una estructura matemática llamada álgebra de Boole, en honor a George Boole, quien la desarrolló a mediados del siglo XIX. El álgebra de Boole denominada también álgebra de la lógica, permite prescindir de la intuición y simplificar deductivamente afirmaciones lógicas que son todavía más complejos. El objetivo principal de este tema es llegar a manejar los postulados y teoremas del álgebra de Boole como herramienta básica en el análisis y síntesis de circuitos digitales.

Este capítulo examina el álgebra de Boole primero en abstracto, y luego en dos casos concretos, los circuitos de interruptores y las compuertas lógicas.

5.2 ÁLGEBRA DE BOOLE

El álgebra booleana es un sistema matemático deductivo centrado en los valores cero y uno (falso y verdadero). Un operador binario " +" definido en éste juego de valores acepta un par de entradas y produce un solo valor booleano, por ejemplo, el operador booleano AND acepta dos entradas booleanas y produce una sola salida booleana.

Sea B un conjunto en el cual se han definido 2 operaciones binarias, + y *, y una operación unitaria, denotada ´; sean 0 y 1 dos elementos diferentes de B. Entonces a la séxtupla {B, +, *, ´, 0, 1} se le llama álgebra de Boole si cumplen los siguientes axiomas para elementos a, b, c cualesquiera en el conjunto B:

Para suma y multiplicación:

- Conmutativo. Se dice que un operador binario es conmutativo si A + B = B + A para todos los posibles valores de A y B.
- Asociativo. Se dice que un operador binario es asociativo si (A + B) + C = A + (B + C) para todos los valores booleanos A, B, y C.
- Distributivo. Dos operadores binarios " + " y " * " son distributivos si A + (B * C) = (A + B) * (A + C) para todos los valores booleanos A, B, y C.
- Complemento: a+ a´= 1

Sea B el conjunto de dos elementos {0, 1}, los complementos se definen por 1´= 0 Y 0´=1.

Por ejemplo:

-Sean Bn el conjunto de sucesiones de n bits. Definir suma, producto y complementos de éstas sucesiones bit por bit. a= 1101010 b= 1011011
a + b = 1111011 a*b = 1001010 a´ = 0010101

5.3 DUALIDAD

El dual de cualquier enunciado en un álgebra de Boole B es el enunciado obtenido al intercambiar las operaciones + y *, e intercambiar los correspondientes elementos identidad 0 y 1, en el enunciado original.

Ejemplo:

(1 + A) × (B + 0) = B es (0 × A) + (B × 1) = B

Si cualquier enunciado es una consecuencia de los axiomas de un álgebra de Boole es también el dual una consecuencia de estos axiomas, ya que el enunciado dual se puede probar usando el dual de cada paso en la demostración del enunciado original, a esto se le llama principio de dualidad.

TEOREMAS BÁSICOS

Usando los axiomas [B1] a [B4], se demuestra el siguiente teorema:

- P3 Los operadores · y + son conmutativos.
- P4 · y + son distributivos uno con respecto al otro, esto es, A· (B+C) = (A·B)+(A·C) y A+ (B·C) = (A+B) ·(A+C).
- P5 Para cada valor A existe un valor A' tal que A·A' = 0 y A+A' = 1. Éste valor es el complemento lógico de A.
- P6 · y + son ambos asociativos, esto es, (AB) C = A (BC) y (A+B)+C = A+ (B+C).

Es posible probar todos los teoremas del álgebra booleana utilizando éstos postulados, además es buena idea familiarizarse con algunos de los teoremas más importantes de los cuales podemos mencionar los siguientes:

- Teorema 1: A + A = A
- Teorema 2: A · A = A
- Teorema 3: A + 0 = A
- Teorema 4: A · 1 = A
- Teorema 5: A · 0 = 0
- Teorema 6: A + 1 = 1
- Teorema 7: (A + B)' = A' · B'
- Teorema 8: (A · B)' = A' + B'
- Teorema 9: A + A · B = A
- Teorema 10: A · (A + B) = A
- Teorema 11: A + A'B = A + B
- Teorema 12: A' · (A + B') = A'B'
- Teorema 13: AB + AB' = A
- Teorema 14: (A' + B') · (A' + B) = A'
- Teorema 15: A + A' = 1
- Teorema 16: A · A' = 0

Los teoremas siete y ocho son conocidos como Teoremas de De Morgan en honor al matemático que los descubrió.

5.4 ORDEN Y ÁLGEBRAS DE BOOLE

Una relación ≤ en un conjunto S se llama un orden parcial en S si cumple las 3 propiedades siguientes:

1) a ≤ a para todo a en S
2) Si a≤ b y b≤ a, entonces a= b
3) Si a≤ b y b≤ c, entonces a≤ c

En tal caso a≤b se lee “a precede a b”.

El término “parcial” se usa al definir un conjunto parcialmente ordenado S, porque puede haber elementos a y b de S que no son comparables, o sea, tales que ni a≤ b ni b ≤ a. Si, por otra parte, todo par de elementos de S es comparable, entonces se dice que S es totalmente ordenado, o linealmente ordenado, y S se denomina cadena.

5.5 EXPOSICIONES DE BOOLE: FORMA SUMA DE PRODUCTOS

Una expresión Booleana E en las variables X1, X2,…, Xn, algunas veces escrito E(X1,..., Xn), es una variable o una expresión construida con éstas variables que usan las operaciones Booleanas +, * y ´.

Características:

Un álgebra de Boole es un conjunto en el que destacan las siguientes características:

1- Se han definido dos funciones binarias (que necesitan dos parámetros) que llamaremos aditiva (que representaremos por x + y) y multiplicativa (que

representaremos por xy) y una función monaria (de un solo parámetro) que representaremos por x'.

2- Se han definido dos elementos (que designaremos por 0 y 1)

3- Tiene las siguientes propiedades:

Conmutativa respecto a la primera función: x + y = y + x
Conmutativa respecto a la segunda función: xy = yx
Asociativa respecto a la primera función: (x + y) + z = x + (y +z)
Asociativa respecto a la segunda función: (xy)z = x(yz)
Distributiva respecto a la primera función: (x +y) z = xz + yz
Distributiva respecto a la segunda función: (xy) + z = (x + z)(y + z)
Identidad respecto a la primera función: x + 0 = x
Identidad respecto a la segunda función: x1 = x
Complemento respecto a la primera función: x + x' = 1
Complemento respecto a la segunda función: xx' = 0

Propiedades del Álgebra de Boole:

Idempotente respecto a la primera función: x + x = x
Idempotente respecto a la segunda función: xx = x
Maximalidad del 1: x + 1 = 1
Minimalidad del 0: x0 = 0
Involución: x'' = x
Inmersión respecto a la primera función: x + (xy) = x
Inmersión respecto a la segunda función: x(x + y) = x
Ley de De Morgan respecto a la primera función: (x + y)' = x'y'
Ley de De Morgan respecto a la segunda función: (xy)' = x' + y'

Función Booleana

Una función booleana es una aplicación de A x A x A x....A en A, siendo A un conjunto cuyos elementos son 0 y 1 y tiene estructura de álgebra de Boole. Supongamos que cuatro amigos deciden ir al cine si lo quiere la mayoría. Cada uno puede votar sí o no. Representemos el voto de cada uno por x_i. La función devolverá sí (1) cuando el número de votos afirmativos sea 3 y en caso contrario devolverá 0.

Si x1 vota 1, x2 vota 0, x3 vota 0 y x4 vota 1 la función booleana devolverá 0.

Producto mínimo (es el número posible de casos) es un producto en el que aparecen todas las variables o sus negaciones.

El número posible de casos es 2^n.

Siguiendo con el ejemplo anterior. Asignamos las letras A, B, C y D a los amigos.

Los posibles casos son:

Votos ABCD	Resultado
1111	1
1110	1
1101	1
1100	0
1011	1
1010	0
1001	0
1000	0
0111	1

0110	0
0101	0
0100	0
0011	0
0010	0
0001	0
0000	0

Las funciones booleanas se pueden representar como la suma de productos mínimos (minterms) iguales a 1.

En nuestro ejemplo la función booleana será:

f (A,B,C,D) = ABCD + ABCD' + ABC'D + AB'CD + A'BCD

5.6 COMPUERTAS LÓGICAS

Las compuertas son bloques del hardware que producen señales del binario 1 o 0 cuando se satisfacen los requisitos de entrada lógica. Las diversas compuertas lógicas se encuentran comúnmente en sistemas de computadores digitales. Cada compuerta tiene un símbolo gráfico diferente y su operación puede describirse por medio de una función algebraica. Las relaciones entrada - salida de las variables binarias para cada compuerta pueden representarse en forma tabular en una tabla de verdad.

* Compuerta AND: Cada compuerta tiene una o dos variables de entrada designadas por A y B y una salida binaria designada por x. La compuerta AND produce la unión lógica AND: esto es: la salida es 1 si la entrada A y la entrada B están ambas en el binario 1: de otra manera, la salida es 0. Estas condiciones también son especificadas en la tabla de verdad para la compuerta AND. La tabla muestra que la salida x es 1 solamente cuando ambas entradas A y B están en 1.

El símbolo de operación algebraico de la función AND es el mismo que el símbolo de la multiplicación de la aritmética ordinaria (*). Podemos utilizar o un punto entre las variables o concatenar las variables sin ningún símbolo de operación entre ellas. Las compuertas AND pueden tener más de dos entradas y por definición, la salida es 1 si cualquier entrada es 1.

* Compuerta OR:  La compuerta OR produce la función OR inclusiva, esto es, la salida es 1 si la entrada A o la entrada B o ambas entradas son 1; de otra manera, la salida es 0. El símbolo algebraico de la función OR (+), similar a la operación de aritmética de suma. Las compuertas OR pueden tener más de dos entradas y por definición la salida es 1 si cualquier entrada es 1.

* Compuerta NOT (Inversor):  El circuito inversor invierte el sentido lógico de una señal binaria. Produce el NOT. O función complemento. El símbolo algebraico utilizado para el complemento es una barra sobra el símbolo de la variable binaria. Si la variable binaria posee un valor 0, la compuerta NOT cambia su estado al valor 1 y viceversa. El círculo pequeño en la salida de un símbolo gráfico de un inversor designa un complemento lógico. Es decir cambia los valores binarios 1 a 0 y viceversa.

* Compuerta Separador:  Un símbolo triángulo por sí mismo designa un circuito separador no produce ninguna función lógica particular puesto que el valor binario de la salida es el mismo de la entrada. Este circuito se utiliza simplemente para amplificación de la señal. Por ejemplo, un separador que utiliza i volt para el binario 1 producirá una salida de 3 volt cuando la entrada es 3 volt. Sin embargo, la corriente suministrada en la entrada es mucho más pequeña que la corriente producida en la salida. De ésta manera, un separador puede excitar muchas otras compuertas que requieren una cantidad mayor de corriente que de otra manera no

se encontraría en la pequeña cantidad de corriente aplicada a la entrada del separador.

* Compuerta NAND:  Es el complemento de la función *AND,* como se indica por el símbolo gráfico que consiste en un símbolo gráfico *AND* seguido por un pequeño círculo. La designación *NAND* se deriva de la abreviación NOT - A*ND.* Una designación más adecuada habría sido *AND* invertido puesto que Es la función *AND* la que se ha invertido.

* Compuerta NOR:  La compuerta *NOR* es el complemento de la compuerta OR y utiliza un símbolo gráfico *OR* seguido de un círculo pequeño. Tanto las compuertas *NAND* como la *NOR* pueden tener más de dos entradas, y la salida es siempre el complemento de las funciones *AND* u OR, respectivamente.

* Compuerta OR exclusivo (XOR):  La compuerta *OR* exclusiva tiene un símbolo gráfico similar a la compuerta *OR* excepto por una línea adicional curva en el lado de la entrada. La salida de esta compuerta es 1 si cada entrada es 1 pero excluye la combinación cuando las dos entradas son 1. La función OR exclusivo tiene su propio símbolo gráfico o puede expresarse en términos de operaciones complementarias *AND,* OR.

* Compuerta NOR exclusivo (XOR):  El *NOR* exclusivo como se indica por el círculo pequeño en el símbolo gráfico. La salida de ésta compuerta es 1 solamente si ambas entradas tienen el mismo valor binario. Nosotros nos referiremos a la función *NOR* exclusivo como la función de equivalencia. Puesto que las funciones OR exclusivo y funciones de equivalencia no son siempre el complemento la una de la otra. Un nombre más adecuado para la operación *OR* exclusivo sería la de una función impar; esto es, la salida es 1 si un número impar de entrada es 1. Así en una función OR (impar) exclusiva de tres entradas, la salida es 1 si solamente

la entrada es 1 o si todas las entradas son 1. La función de equivalencia es una función par; esto es, su salida es 1 si un número par de entradas es 0.

5.7 CIRCUITOS LÓGICOS

La relación que existe entre la lógica booleana y los sistemas de cómputo es fuerte, de hecho se da una relación uno a uno entre las funciones booleanas y los circuitos electrónicos de compuertas digitales. Para cada función booleana es posible diseñar un circuito electrónico y viceversa, como las funciones booleanas solo requieren de los operadores AND, OR y NOT podemos construir nuestros circuitos utilizando exclusivamente éstos operadores utilizando las compuertas lógicas homónimas.

Un hecho interesante es que es posible implementar cualquier circuito electrónico utilizando una sola compuerta, ésta es la compuerta NAND.

Para probar que podemos construir cualquier función booleana utilizando sólo compuertas NAND, necesitamos demostrar cómo construir un inversor (NOT), una compuerta AND y una compuerta OR a partir de una compuerta NAND, ya que como se dijo, es posible implementar cualquier función booleana utilizando sólo los operadores booleanos AND, OR y NOT. Para construir un inversor simplemente conectamos juntas las dos entradas de una compuerta NAND. Una vez que tenemos un inversor, construir una compuerta AND es fácil, sólo invertimos la salida de una compuerta NAND, después de todo, NOT (NOT (A AND B)) es equivalente a A AND B. Por supuesto, se requieren dos compuertas NAND para construir una sola compuerta AND, nadie ha dicho que los circuitos implementados sólo utilizando compuertas NAND sean lo óptimo, solo se ha dicho que es posible hacerlo. La otra compuerta que necesitamos sintetizar es la compuerta lógica OR, esto es sencillo si utilizamos los teoremas de De Morgan, que en síntesis se logra en tres pasos, primero se reemplazan todos los "·" por "+" después se invierte cada literal y por último se niega la totalidad de la expresión:

A OR B

A AND B.....Primer paso para aplicar el teorema de De Morgan

A' AND B'.....Segundo paso para aplicar el teorema de De Morgan

(A' AND B')'.....Tercer paso para aplicar el teorema de De Morgan

(A' AND B')' = A' NAND B'.....Definición de OR utilizando NAND

Si se tiene la necesidad de construir diferentes compuertas de la manera descrita, bien hay dos buenas razones, la primera es que las compuertas NAND son las más económicas y en segundo lugar es preferible construir circuitos complejos utilizando los mismos bloques básicos. Observe que es posible construir cualquier circuito lógico utilizando sólo compuertas de tipo NOR (NOR = NOT(A OR B)).

PROBLEMAS

1.- Considerar D110= {1, 2, 5, 10, 11, 22, 55,110}, el álgebra de Boole de los divisores de 110. Las operaciones son las definidas en el ejemplo 7.1 (e).

a) Evaluar:

X1=2+11´= 10 X3=22*(5+10)= 2 X5= (55*10) ´ + 2= 22

X2=5*10+2=10 X4= 5´ * 55= 11 X6= (2 + 11´) * 22= 2

b) Dibujar el diagrama de D110

c) Encontrar los átomos de D110

R= 2, 5,11

2.- Para el álgebra de Boole del ejemplo 7.1(a), evaluar:

X= 1*(0 + 1´)= 1*0= 0 Y= (1 + 1) * (0´ + 0)=1 Z= (1´ + 0´) + (1*0´) ´=1

3.- Un elemento M de una álgebra de Boole B se llama maxterm si el elemento unidad 1 es su único sucesor estricto.

a) Encontrar los maxternos del álgebra de Boole en el problema 7.1. 10, 14, 35
b) Encontrar los maxternos del álgebra de Boole en el problema 7.18. 10, 22, 55

4.- Escribir el dual de cada ecuación de Boole:

a) a (a´ + b)= ab= a+a b=a+b
b) (a + 1) (a + 0)= a = a*0+a*1=a
c) (a +b) (b + c)= ac + b= ab + bc= (a+c) b

5.-Escribir cada expresión de Boole E(x, y, z) como suma de productos, y luego la forma completa de suma de productos:

a) x (xy´ + x´y + y´z)= xy + xy z= xy z + xy z
c) (x + y´z) (y + z´)= xy + xz = xyz + xyz +xy z
e) (x´ + y) ´ + y´z= xy + y z= xy z + xy z + x y z
b) (x´y) ´ (x´ + xyz´)= xyz + x y= xyz + x y z +x y z
d) (x + y) ´ (xy´) ´ = x y= x y z + x y z
f) y(x + yz) ´= x yz

6.- Escribir el siguiente conjunto de expresiones que usan A, B y C como uniones de intersecciones:

a) (A U B) ∩ (C U B)= A ∩ B ∩ C
b) (B ∩ C) ∩ (A U C)= (A ∩ B ∩ C) U (A ∩ C)

7.- Determinar la salida de cada compuerta.

a) 1101110111 b) 1000010000 c) 0001110101

8.- Si A= 1100110111, B= 0001110110, C= 1010110011, evaluar:

a) A + B= 1101110111
b) A+C= 1110110111
c) Ā= 0011001000
d) Ĉ= 0101001100
e) A * C= 1000110011
f) B * C= 0000110010
g) Ā + B * Ĉ= 0011001100
h) B (A + C) = 00010000000

9.- Dadas cuatro entradas A, B, C, D, encontrar sucesiones especiales que den todas las posibles combinaciones diferentes de entradas.

A=1111111100000000 B= 1111000011110000
C=1100110011001100 D= 1010101010101010

10.- Considerar el circuito lógico en la fig. 7.16.

a) Dar la salida Y como una expresión de Boole con entradas A, B, C.
b) Encontrar la tabla de verdad del circuito.

11.- Considerar el circuito lógico en la fig. 7.17.
a) Dar la salida Y como una expresión de Boole con entradas A, B, C.
b) Encontrar la tabla de verdad del circuito.

12.- Considerar el circuito lógico de la fig. 7-18.

a) Dar la salida Y como una expresión de Boole con entradas A, B, C.
b) Encontrar la tabla de verdad del circuito.

13.- Considerar el circuito lógico de la fig. 7-19, contiene compuertas NAND y NOR.

a) Dar la salida Y como una expresión de Boole con entradas A, B, C.
b) Encontrar la tabla de verdad del circuito.

14.-Dibujar el circuito lógico que corresponde a cada expresión de Boole:

a) E1= AB + ABC b) E2= A + BC + B c) E3= AB + A + C

15.- Encontrar la expresión de Boole que corresponde a cada circuito de interruptores de la fig. 7-20.

CAPÍTULO 6. TURBO C

6.1 ORÍGENES DE C

C es un lenguaje de programación de propósito general que ofrece economía sintáctica, control de flujo y estructuras sencillas y un buen conjunto de operadores. No es un lenguaje de muy alto nivel, es más bien un lenguaje pequeño, sencillo, y no está especializado en ningún tipo de aplicación. Esto lo hace un lenguaje potente, con un campo de aplicación ilimitado y sobre todo, se aprende rápidamente. En poco tiempo, un programador puede utilizar la totalidad del lenguaje.

Este lenguaje ha sido estrechamente ligado al sistema operativo UNIX, puesto que fueron desarrollados conjuntamente. Sin embargo, este lenguaje no está ligado a ningún sistema operativo ni a ninguna máquina concreta. Se le suele llamar lenguaje de programación de sistemas debido a su utilidad para escribir compiladores y sistemas operativos, aunque de igual forma se pueden desarrollar cualquier tipo de aplicación.

La base del C proviene del BCPL, escrito por Martin Richards, y del B escrito por Ken Thompson en 1970 para el primer sistema UNIX en un DEC PDP-7. Estos son lenguajes sin tipos, al contrario que el C que proporciona varios tipos de datos. Los tipos son caracteres, números enteros y en coma flotante, de varios tamaños. Además se pueden crear tipos derivados mediante la utilización de punteros, vectores, registros y uniones. El primer compilador de C fue escrito por Dennis Ritchie para un DEC PDP-11 y escribió el propio sistema operativo en C.

C trabaja con tipos de datos que son directamente tratables por el hardware de la mayoría de computadoras actuales, como son los caracteres, números y direcciones. Estos tipos de datos pueden ser manipulados por las operaciones

aritméticas que proporcionan las computadoras. No proporciona mecanismos para tratar tipos de datos que no sean los básicos, debiendo ser el programador el que los desarrolle. Esto permite que el código generado sea muy eficiente y de ahí el éxito que ha tenido como lenguaje de desarrollo de sistemas. No proporciona otros mecanismos de almacenamiento de datos que no sea el estático y no proporciona mecanismos de entrada ni salida. Ello permite que el lenguaje sea reducido y los compiladores de fácil implementación en distintos sistemas. Por contra, estas carencias se compensan mediante la inclusión de funciones de librería para realizar todas estas tareas, que normalmente dependen del sistema operativo.

Originariamente, el manual de referencia del lenguaje para el gran público fue el libro [1] de Kernighan y Ritchie, escrito en 1977. Es un libro que explica y justifica totalmente el desarrollo de aplicaciones en C, aunque en él se utilizaban construcciones, en la definición de funciones, que podían provocar confusión y errores de programación que no eran detectados por el compilador. Como los tiempos cambian y las necesidades también, en 1983 ANSI establece el comité X3J11 para que desarrolle una definición moderna y comprensible del C. El estándar está basado en el manual de referencia original de 1972 y se desarrolla con el mismo espíritu de sus creadores originales. La primera versión de estándar se publicó en 1988 y actualmente todos los compiladores utilizan la nueva definición. Una aportación muy importante de ANSI consiste en la definición de un conjunto de librerías que acompañan al compilador y de las funciones contenidas en ellas. Muchas de las operaciones comunes con el sistema operativo se realizan a través de estas funciones. Una colección de ficheros de encabezamiento, *headers*, en los que se definen los tipos de datos y funciones incluidas en cada librería. Los programas que utilizan estas bibliotecas para interactuar con el sistema operativo obtendrán un comportamiento equivalente en otro sistema.

6.2 ALGORITMO

En computación uno de los temas más importantes para comenzar a estudiarla son los algoritmos.

La palabra algoritmo se deriva de la traducción de la palabra árabe *alkhowarizmi*, nombre de un matemático y astrónomo que escribió un tratado sobre manipulación de número en el siglo IX.

Un algoritmo es una serie de pasos organizados que describe el proceso para dar solución a un problema específico.

Las características fundamentales de un algoritmo o proceso algorítmico son:

Un algoritmo debe de ser preciso e indicar el orden de realización de cada paso.

Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces se debe obtener el mismo resultado cada vez.

Un algoritmo debe ser finito si se sigue un algoritmo se debe terminar en algún momento.

La función de un algoritmo debe describir 3 partes: entrada, proceso y salida.

6.3 CICLO DE VIDA DE UN PROGRAMA

ANÁLISIS DEL PROGRAMA

Un equipo de analistas y usuarios debe decidir exactamente lo que el programa debe hacer, qué datos debe procesar y que información producirá.

DISEÑO DE LA SOLUCIÓN

El programador especifica las funciones del procesamiento de los datos que el programa debe ejecutar. Las relaciones entre las funciones establecen en una serie de diagrama de flujo.

CODIFICACIÓN

El programador escribe el código fuente del programa. Este código fuente consta de los pasos del programa descritos en un lenguaje de computadora el código fuente se traduce en un programa que la computadora puede ejecutar.

IMPLEMENTACIÓN

El programador implanta y ejecuta el programa en una computadora.

EVALUACIÓN

El programador comprueba el programa para asegurarse que produce la información requerida. Durante esta fase se podría modificar el programa.

DOCUMENTACIÓN

El programador describe el funcionamiento y uso del programa en una documentación técnica y usual.

AJUSTES

Si el tipo de información requerida necesitara cambios el programa debe tener que ser modificado. Así mismo los usuarios del programa pueden descubrir

errores o a su vez introducir cambios que la experiencia les dicte y modificar el programa.

6.4 PSEUDOCÓDIGO

Pseudo (código) o *seudo* (*código*) significa falso o imitación y código se refiere a las instrucciones escritas en el lenguaje de programación. Seudo código no es realmente un código sino una imitación y una versión abreviada de instrucciones reales para la computadora.

El *pseudocódigo* es un lenguaje utilizado para definir algoritmos con una sintaxis simple similar a la de un lenguaje de programación.

Mezcla del lenguaje de programación y español o inglés, o cualquier otro idioma que se emplea dentro de la programación estructurada para realizar el diseño de un programa. En esencia el seudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un programa determinado. El seudocódigo utiliza palabras que indican el proceso a realizar.

6.5 LOGARITMOS

Es el exponente o palabra a la que un número fijo llamado base se ha de elevar para dar un número dado. Por ejemplo en la expresión $10^2 = 100$, el logaritmo de 100 en base 10 es 2, se escribe $\log_{10} 100 = 2$. Los logaritmos fueron inventados para simplificar los procedimientos aritméticos de multiplicación, división, potencia y extracción de raíces, pero actualmente tiene muchas aplicaciones tanto en matemáticas puras como aplicadas.

Las primeras tablas de logaritmos fueron aplicados por el matemático escocés John Napier.

Un algoritmo es la base elevada a la potencia del número dado. Por ejemplo, el antilogaritmo de 2 en base 10 es 10^2.

Para elevar un número a una potencia cualquiera se multiplica el logaritmo del número por la potencia deseada y se calcula el antilogaritmo.

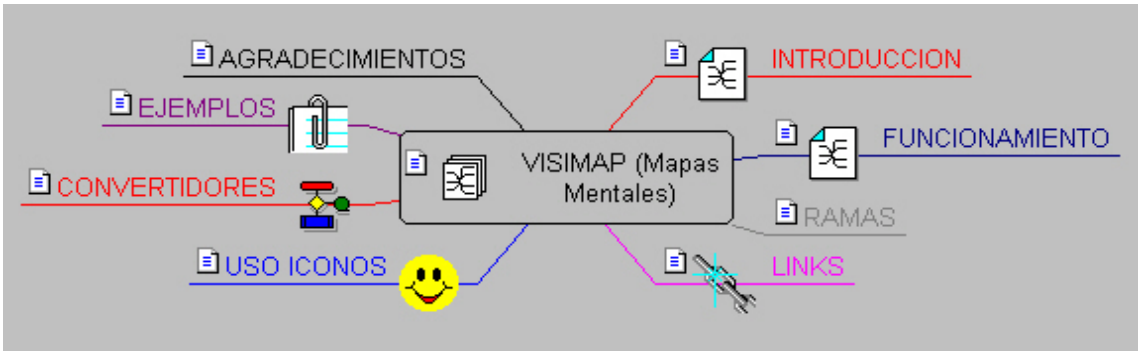
EJEMPLO:

4^3: $\log_2 4=2$, 3*2=6 antilog 6=64, que es 4 a la tercera potencia. La extracción de raíces se calcula dividiendo el logaritmo del radicando por la raíz. Para calcular la raíz quinta de 32: $\log_2 32=5$, 5; antilog 1=2 que es la raíz quinta de 32.

6.6 MAPAS MENTALES

6.6.1 INTRODUCCIÓN

Un mapa mental es una representación gráfica de una serie de ideas. Siempre es más fácil recordar una imagen gráfica que no un texto lineal.
Existe diversos software para la creación de estos gráficos



6.6.2 FUNCIONAMIENTO

Todo mapa mental tiene una idea principal en el centro del mismo del que luego surgen las ramas.

Estas ramas pueden estar conexionadas a otros mapas y así sucesivamente.

La mejor manera de crearlo es crear lo que se llama una "tormenta de ideas" (brainstorming). De esta forma unas ideas van llamando otras. Así pues podemos crear un mapa para crear el guión de una conferencia o simplemente este artículo al cual he ido añadiendo en un principio las ideas sobre las que quiero hablar y posteriormente he ido desarrollando.

6.6.3 RAMAS

Las ramas surgen del centro del mapa o de cualquier otra rama de esta forma podemos ir directamente a la información que nos interese sin tener que leer todo un documento. Es más fácil buscar un apartado de un tema seleccionando la rama que no utilizando el típico sistema lineal.

Lo aconsejable para poder recordar fácilmente estos mapas es crear varios mapas en lugar de un mapa con muchas ramas, así como el uso de colores diferentes para cada una de ellas.

Mediante el uso de colores podemos poner por ejemplo los aspectos más importantes en color rojo.

6.6.4 FUNDAMENTO DE LOS MAPAS MENTALES

Constituyen un recurso pedagógico y didáctico muy útil para organizar, clasificar y categorizar la información que conocemos con respecto a un tema determinado.

Son la expresión externa del proceso natural que realiza el cerebro al buscar o recordar informaciones. "Es una poderosa técnica gráfica que nos ofrece una llave maestra para acceder al potencial del cerebro" (T.Buzán*, p.69). Lo que se pretende, por tanto, es poner en marcha procesos asociativos de pensamiento.

A través del mapa mental, y a partir de una idea central o tema, nacen varias ramas (Ideas Organizadoras Básicas - I.O.B.-) que serán el soporte gráfico para expresar las asociaciones que nos vayan surgiendo. La conocida "lluvia de ideas" es una técnica relacionada directamente con el proceso de creación de mapas mentales.

PODRÍAMOS RESUMIR LA DEFINICIÓN DE MAPAS MENTALES EN ESTAS PALABRAS:

"Representación gráfica de un proceso integral que facilita la toma de notas y repasos efectivos. Permite unificar, separar e integrar conceptos para analizarlos y sintetizarlos, secuencialmente; en una estructura creciente y organizada, compuesta de un conjunto de imágenes, colores y palabras, que integran los modos de pensamiento lineal y espacial".

Un tipo de mapa mental en computación son los diagramas de flujo que es el tema que a continuación aprenderemos a realizar.

6.7 DIAGRAMAS DE FLUJO

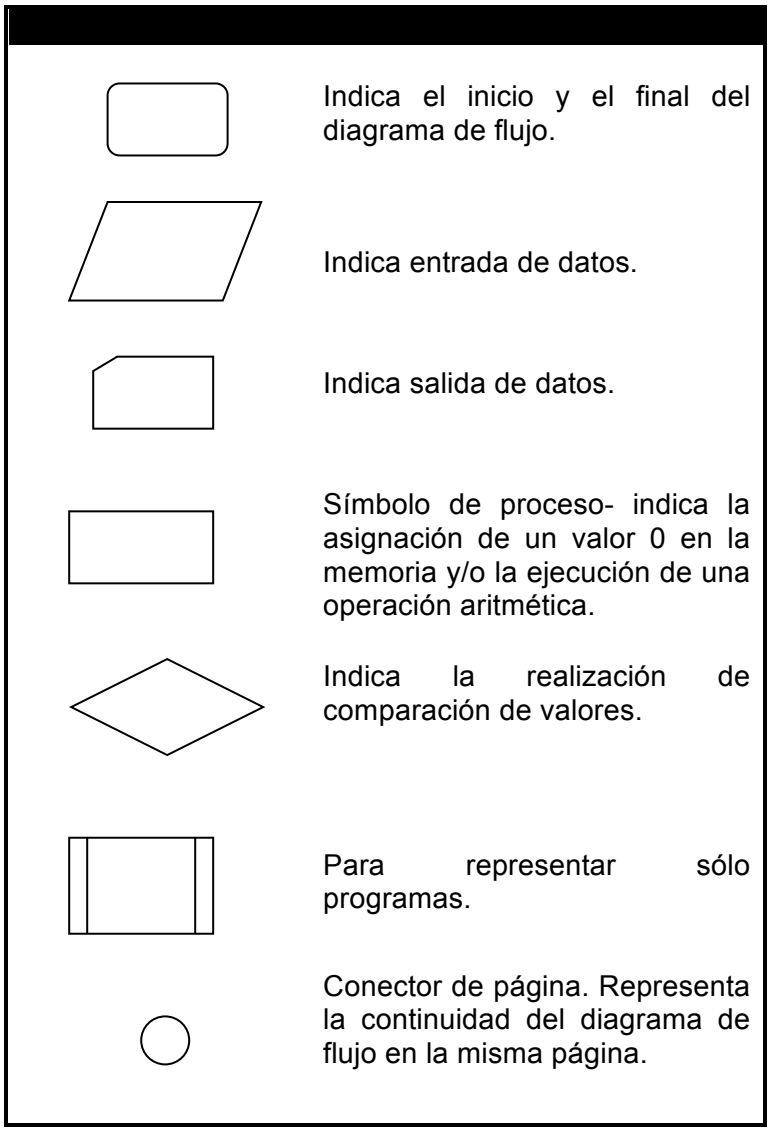
Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de cómo deben realizarse los pasos en la computadora para producir resultados. Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre sí mediante líneas que indican

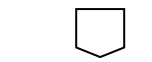
el orden en que se deben ejecutar los procesos. Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización.

6.7.1 FUNCIÓN DEL DIAGRAMA DE FLUJO EN LA FASE DE DISEÑO

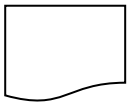
En la fase de diseño se suele poner diagramas que indiquen como se manipularan los datos en el programa o sistema, es decir, el programa nos podrá decir la interacción entre los datos de entrada proceso y salida.

6.7.2 SIMBOLOGÍA UTILIZADA

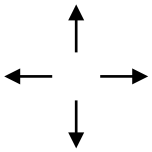




Conector fuera de página. Representa la continuidad del diagrama.



Salida a pantalla indica la salida de datos a la pantalla o monitor.



Línea de flujo o dirección. Indica la secuencia en que se realizan las operaciones.

6.7.3 REGLAS PARA LA CONSTRUCCIÓN DE UN DIAGRAMA DE FLUJO

1. Todo diagrama debe tener un inicio y un fin.
2. Las líneas de conexión o de flujo deben ser siempre rectas y si es posible, que sean sólo verticales y horizontales (no cruzadas, ni inclinadas).
3. Las líneas que enlazan los símbolos entre sí debe estar todas conectadas. Cada línea o flecha debe entrar en un símbolo de decisión, en un sí o unirse a otra flecha.
4. Se deben dibujar todos los símbolos de modo que se pueda seguir el proceso visualmente de arriba abajo y de izquierda a derecha.
5. Realizar un gráfico claro y equilibrado, procurando que el flujo central del diagrama sea parte central de la hoja de papel.
6. Evitar la terminología especifica de un lenguaje de programación máquina sobre todo en las expresiones donde se tiende a ello.
7. En las operaciones lógicas recurrir preferentemente a la lógica positiva que a la negativa.
8. A cada bloque o símbolo se accede por arriba y/o por la izquierda y se sale por abajo y/o por la derecha. La salida es única excepto en los símbolos de decisión.

6.7.4 RELACIÓN ENTRE ALGORITMO, DIAGRAMA DE FLUJO Y CODIFICACIÓN

El algoritmo y el diagrama de flujo. A partir del algoritmo y diagrama de flujo se puede llegar a la codificación (programa) aunque muchos programadores prefieren hacer directamente la codificación sin pasar por los anteriores.

6.7.5 ELEMENTOS BÁSICOS PARA EL DISEÑO DE UN ALGORITMO

DATOS:

- DATOS NUMÉRICOS

Los datos numéricos se representan en dos formas: números enteros y números reales.

Los enteros corresponden a números completo, no tienen componente decimal y pueden ser negativos y positivos.

Por ejemplo: -2526, 45, 45553.

Los números reales tienen siempre un punto decimal y pueden ser positivos y negativos.

Por ejemplo: 64.3, -6, 25.2.

Al objeto de poder representar números reales muy grandes o muy pequeños se le denomina notación científica, o coma o de punto flotante y tiene el siguiente formato n=m*b^e donde m es la matiza, e es el exponente y es igual a un número, y b es la base del sistema de numeración generalmente diez.

- DATOS NO NUMÉRICOS

Existen dos tipos: datos alfanuméricos y datos lógicos.

Los datos alfanuméricos se agrupan en series o cadenas de caracteres de la A…Z, a…z, los dígitos 0, 1, 2,3,..,9; caracteres especiales (#, $, -, etc.). Los datos lógicos son aquellos que se pueden tomar como falsos o verdaderos.

- DATOS DEFINIDOS POR EL SISTEMA

Son un conjunto de constantes con valores definidos por la plataforma u ordenador o computadora que se esté utilizando.

- DATOS DEFINIDOS POR EL USUARIO

Constantes y variables que el usuario les asigna un valor o rango.

CONSTANTES:

Existen cuatro tipos de constantes las cuales son:

- CONSTANTE ENTERA (INTEGER).

Una constante entera es un número con un valor entero positivo o negativo, consistente en una secuencia de dígitos.

Por ejemplo: 8, 3, - 4.

- CONSTANTE DE COMA FLOTANTE

Es un número en base 10 que contiene un punto decimal o un exponente (o ambos).

Por ejemplo: 30000 = 3e^5.

- CONSTANTE DE CARÁCTER

Es un solo carácter encerrado por comillas simples.

Por ejemplo: 'x', 'z', '#', '0'.

- CONSTANTE DE CADENA DE CARACTERES

Consta de cualquier número de caracteres consecutivos (o ninguno) encerrado entre comillas.

Por ejemplo: "verde", "19.95", "2700.502", "Juan Pérez".

VARIABLES:

Una variable es un identificador que se utiliza para representar cierto tipo de información dentro de una determinada parte del programa.

Una variable contendrá un dato simple, esto es numérico o una constante de carácter, dependiendo del lenguaje de programación existen diferentes tipos de variables, como enteras, reales, caracteres, objetos, variantes, etc.

EXPRESIONES

Se pueden evaluar datos y obtener nuevos valores, evaluando expresiones. Las expresiones son combinaciones de constantes, variables, símbolos de combinación (operadores), paréntesis de apertura y cierre y nombre de funciones.

CONTADORES

En los procesos repetitivos se utiliza normalmente contar los sucesos o acciones internos del ciclo. Un contador es un campo de la memoria que está destinado a tener los diferentes valores que se están incrementando o decrementando en cada iteración.

El incremento en las sumas parciales es siempre constante pudiendo ser positivo o negativo, es decir, el contador se irá incrementando o decrementando. El contador se representa en un programa como una variable.

ACUMULADORES

Un acumulizador o totalizador es un campo o zona de memoria cuya misión es almacenar cantidades variables resultantes de sumas sucesivas, realiza la función de un contador con la diferencia del incremento o decremento de cada suma.

BANDERAS (INTERRUPTORES O CONMUTADORES)

Un interruptor (a veces llamado centinela) es un campo de memoria (variable) que toma diversos valores a lo largo de la ejecución del programa y que permite comunicar información de una parte a otra del mismo, es decir, variar la secuencia de ejecución de un programa dependiendo del valor que tenga en cada modelo. Los dos únicos valores que pueden tomar un interruptor son 0 y 1.

6.8 UNIDADES DE MEMORIA

En la computadora existen pequeñas celdas de memoria llamadas bit que son las encargadas de guardar y procesar en ellas la información que deseamos.

BIT

En informática, acrónimo de Binary Digit (dígito binario), que adquiere el valor 1 o 0 en el sistema numérico binario. En el procesamiento y almacenamiento informático un bit es la unidad de información más pequeña manipulada por la computadora, y está representada físicamente por un elemento como un único pulso enviado a través de un circuito, o bien como un pequeño punto en un disco magnético capaz de almacenar un 0 o un 1. La representación de información se logra mediante la agrupación de bits para lograr un conjunto de valores mayor que permite manejar mayor información.

BYTE

En informática, unidad de información que consta de 8 bits; en procesamiento informático y almacenamiento, el equivalente a un único carácter, como puede ser una letra, un número o un signo de puntuación. Como el byte representa sólo una pequeña cantidad de información, la cantidad de memoria y de almacenamiento de una máquina suele indicarse en kilobytes, en megabytes o en gigabytes

Bit	Byte	kaByte	MegaByte	gigaByte	teraByte
0/1	00000000	1000 bite	1000 kabite	1000 megabite	1000 gigabite
	Un caracter	Mil caracteres	Un million de caracteres	Mil milliones de caracteres	Un billion de caracteres
		1000000 de bits	1000000000 de bits	1000000000000 de bits	1000000000000000 bits

Como en este tema hemos hablado ya del sistema binario, conozcamos ahora los dos sistemas de numeración más usados por la computadora.

6.9 SISTEMAS DE NUMERACIÓN: BINARIO Y HEXADECIMAL

Puesto que sólo se necesitan dos dígitos (o bits), el sistema binario se utiliza en los ordenadores o computadoras. Un número binario cualquiera se puede representar, por ejemplo, con las distintas posiciones de una serie de interruptores. La posición "encendido" corresponde al 1, y "apagado" al 0. Además de interruptores, también se pueden utilizar puntos imantados en una cinta magnética o disco: un punto imantado representa al dígito 1, y la ausencia de un punto imantado es el dígito 0. Los biestables —dispositivos electrónicos con sólo dos posibles valores de voltaje a la salida y que pueden saltar de un estado al otro mediante una señal externa— también se pueden utilizar para representar números binarios. Los circuitos lógicos realizan operaciones con números en base 2. La conversión de números decimales a binarios para hacer cálculos, y de números binarios a decimales para su presentación, se realizan electrónicamente.

El sistema binario desempeña un importante papel en la tecnología de los ordenadores. Los primeros 20 números en el sistema en base 2 son 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001, 10010, 10011 y 10100. Cualquier número se puede representar en el sistema binario, como suma de varias potencias de dos. Por ejemplo, el número 10101101 representa, empezando por la derecha, $(1 \times 2^0) + (0 \times 2^1) + (1 \times 2^2) + (1 \times 2^3) + (0 \times 2^4) + (1 \times 2^5) + (0 \times 2^6) + (1 \times 2^7) = 173$.

Dos dígitos —0 y 1— son suficientes para representar un número en el sistema binario; 16 guarismos —0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (diez), B (once), C (doce) ... y F (quince)— son necesarios en el sistema hexadecimal. El número 2EF del sistema hexadecimal es el número $(2 \times 16^2) + (14 \times 16^1) + (15 \times 16^0) = 751$ en el sistema decimal.

Para convertir un número n dado en base 10 a base b, se divide (en el sistema decimal) n por b, el cociente se divide de nuevo por b y así sucesivamente hasta

que se obtenga un cociente cero. Los restos sucesivos de esta serie de divisiones son los dígitos que expresan n en base b (la base se suele escribir como un subíndice del número).

Ejemplos:

Convertir el número 10011_2 al sistema decimal:

10011_2 $(1x2^0)+ (1x2^1)+ (0x2^2)+ (0x2^3)+ (1x2^4)=19$

Convertir el número $34DF_{16}$ al sistema decimal:

$34DF_{16} = (3x16^3)+ (4x16^2)+ (13x16^1)+ (15x16^0)= 13535$

Convertir el número 45_{10} al sistema binario:

45/2=22 residuo 1
22/2=11 residuo 0
11/2= 5 residuo 1
5/2=2 residuo 1
2/2=1 residuo 0
1/2=0 residuo 1

Los residuos en sentido inverso representan el número decimal en el sistema binario, es decir:

$45_{10} = 101101_2$

Convertir el numero165_{10} al sistema hexadecimal:

165/16=10 residuo 5

10/16=0 residuo 10

Los residuos en sentido inverso representan el número decimal en el sistema hexadecimal, es decir:

$165_{10} = A5_{16}$

A medida que la base sea mayor, se necesitan más guarismos (*), pero la representación de un número requiere menos dígitos.

(*)GUARISMO: Cada uno de los signos o cifras arábigas que expresan una cantidad.

Ejercicios Resueltos

Ejercicio 1.- CONVERTIR LA EDAD EN AÑOS DE UNA PERSONA A MESES.

Ejercicio 2.- CONVERTIR PESOS A DOLARES.

Ejercicio 4.- CALCULAR EL AREA DE UN CIRCULO CON LA FORMULA:

$$AREA = PI * RADIO^2$$

Ejercicio 5.- EVALUAR LA FUNCION $Y = 5X^2 - 3X + 2$ PARA CUALQUIER VALOR DE X.

Observar para el caso de constantes fijas o conocidas (PI) que no se debe dar como dato de entrada su valor, en cambio colocar directamente su valor dentro de la formula, en la parte de operaciones del problema.

Pero recordar también que existirán problemas sencillos donde:

No se ocupan entradas o no se ocupan operaciones, pero todos ocupan salida.

Una formula grande o muy compleja puede ser más segura y fácil de resolver, si es descompuesta y resuelta en partes, juntando al final los parciales para obtener el resultado final.

Un problema puede tener más de una solución correcta.

El problema no está suficientemente explicado o enunciado, entonces, estudiarlo, analizarlo y construirlo de manera genérica.

Problemas Propuestos

Resuelva el problema y haga su respectivo diagrama de flujo.

1.- Leer dos números, hacer su suma e imprimir el resultado.

2.- Leer dos números enteros e imprimir en la pantalla los números desde el cero hasta el número que se leyó.

3.- Leer dos números e imprimir el mayor de los números.

4.- Leer números y obtener su media.

5.- Leer dos números complejos y hacer su suma.

6.- Convertir millas a kilómetros (caso normal).

7.- Convertir 125 metros a centímetros (no ocupa entradas).

8.- Se calcula que en promedio hay 4.7 nidos en cada árbol en la UABC, también se calcula que en cada nido existen un promedio de 5.8 pájaros, se pide calcular la cantidad total de nidos y de pájaros en los 227 árboles que existen en la UABC (no ocupa entradas).

9.- La gorda Sra. López y sus 8 hijos solo compran una vez al mes su mandado en conocido supermercado, en dicha tienda el kilogramo de frijol cuesta $8.75, el paquete de tortillas cuesta $3.55 y el frasco de café vale $14.25, si solo compran de estos tres productos para su mandado, calcular su gasto total (problema no claro).

10.- Capturar y desplegar los cinco datos más importantes de un automóvil (no ocupa operaciones).

11.- La distancia Tijuana - Ensenada es de 110 kilómetros. Si un automóvil la recorre a una velocidad constante de 30 millas por hora, cuanto tiempo tarda en llegar (1 milla = 1.609 Km.) (Dos maneras correctas de resolverlo).

12.-Evaluar la función para cualquier valor de x (caso normal).

13.-Evaluar la función para cuando x vale 4 (No ocupa entradas).

14.- Evaluar el factorial de cualquier número usando la fórmula: n!=n!-1.

15.-La distancia que recorre un auto es de 50 km. y su velocidad es de 30 millas por hora. ¿Cuánto tiempo tardará en llegar?

16.-Encontrar la derivada de x para cualquier valor con la formula (d/dx(x)=1).

17.-Calcular l interés que gana un capital de x pesos a una tasa de interés del 15% anual en un periodo de n años.

18.-Que aceleración tiene un tren que parte de Tijuana a 10 km/hr y pasa por Ensenada una hora después a 50 km/hr.

19.-Calcular el número de aulas en una escuela que tiene 10 edificios y cada edificio 3 pisos y cada piso 5 aulas, excepto un edificio que solo tiene dos pisos.

20.-Si en una escuela hay 30 maestros y 15 son hombres que atienden a 10 alumnos cada uno. ¿Cuantas maestras hay?

21.-Calcular la corriente de un circuito con un voltaje de 15v y una resistencia de 6 ohms. Formula (I= V/R).

22.-Calcular la normal estándar (z) dados los datos por el usuario: X=dato, μ=media, d=desviación. Formula (Z = X-M / d).

23.-Dado un número(N) cualesquiera obtener su raíz y potencia cuadrada.

24.-Determinar la media de 5 números diferentes.

25.-Determinar la velocidad v requerida para recorrer una distancia d en un tiempo t. Formula (V = d * t).

26.-Determinar la pendiente de una recta. Formula (y = m x + b).

27.-Calcular la función de $y= x^2 + 8x + 3$ para cualquier x.

28.-Convertir minutos a horas.

29.-Aplicar la formula general para a=1, b=2, c=3.

30.-Se desea instalar un cable de red, el cliente pide 30 pies, considerando que se venden en metros, cuantos deberá comprar.

31.-Un campesino siembra trigo en un área cuadrada de 25 m., ¿cuál es el largo del cerco frontal en cm?

32.-Resolver $x^2 + 15x - 8$ para cualquier variable (X).

33.-Convertir °C a °F.

34.-Si cada salón de la escuela tiene 40 alumnos y son 30 salones ¿Cuantos alumnos son en toda la escuela?

35.-Si Juan trabaja 5 días a la semana y descansa 2 ¿Cuantos días trabajo en 4 años?

36.-Si en una oficina se procesan 20 facturas cada 10 minutos, ¿cuantas se procesaran si se trabajan 5 horas?

37.-Si una empresa tiene _____ de activo y un pasivo de _____ ¿Cuál es su capital? Formula (C = A-P).

38.-Calcule el voltaje de un circuito dada una intensidad I y una resistencia R. Formula (V=IR)

39.-Calcule la frecuencia de una onda que circula con un tiempo t. Formula (F=1/t).

40.-Calcule la potencia de un circuito con un voltaje V y una intensidad I. Formula (f = VI).

41.-Calcule el total que tendrá que pagar una persona que va al cine dependiendo del No. de boletos a comprar y el precio.

42.-Calcule las anualidades que tendrá que pagar una persona que pidió un préstamo. Dependiendo del tiempo que el elija y el interés por año. Formula (Anualidad= (Préstamo/Tiempo)+interés).

43.-Determinar cuánto ganara una persona en base a la horas trabajadas. Tomando en cuenta el pago por hora.

44.-Convertir horas a segundos.

45.-Calcular la fuerza. Formula (fuerza = trabajo / tiempo).

6.10 PROGRAMAS

La mejor forma de aprender un lenguaje es programando con él. El programa más sencillo que se puede escribir en C es el siguiente:

```
Main ( )
{
}
```

Como nos podemos imaginar, este programa no hace nada, pero contiene la parte más importante de cualquier programa C y además, es el más pequeño que se puede escribir y que se compile correctamente. En él se define la función main, que es la que ejecuta el sistema operativo al llamar a un programa C. El nombre de una función C siempre va seguida de paréntesis, tanto si tiene argumentos como si no. La definición de la función está formada por un bloque de sentencias, que está encerrado entre llaves { }.

Un programa algo más complicado pero que hace algo, es el siguiente:

```
#include <stdio.h>
```

```
Main ( )
{
        Printf ("Hola amigos!\n");
}
```

Con él visualizamos el mensaje Hola amigos! en el terminal. En la primera línea indica que se tengan en cuenta las funciones y tipos definidos en la librería stdio (*standard input/output*). Estas definiciones se encuentran en el fichero *header* stdio.h. Ahora, en la función main se incluye una única sentencia que llama a la función printf. Esta toma como argumento una cadena de caracteres, que se imprimen van encerradas entre dobles comillas " ". El símbolo \n indica un cambio de línea. Hay un grupo de símbolos, que son tratados como caracteres individuales, que especifican algunos caracteres especiales del código ASCII. Los más importantes son:

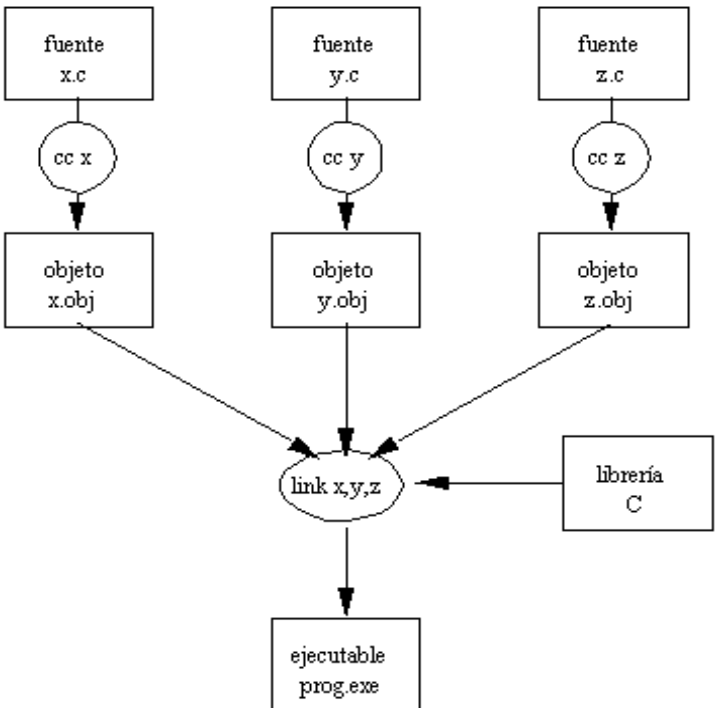
\a	*alert*
\b	*backspace*
\f	*formfeed*
\n	*newline*
\r	*carriage return*
\t	*horizontal tab*
\v	*vertical tab*
\\	*backslash*
\'	*single quote*
\"	*double quote*
\OOO	Visualiza un carácter cuyo código ASCII es OOO en octal.
\xHHH	Visualiza un carácter cuyo código ASCII es HHH en hexadecimal.

Las funciones de entrada y salida y los formatos utilizados los explicaremos con más detalle en otro capítulo.

6.10.1 EL DESARROLLO DE UN PROGRAMA

Un programa C puede estar formado por diferentes módulos o fuentes. Es conveniente mantener las fuentes de un tamaño no muy grande, para que la compilación sea rápida. También, al dividirse un programa en partes, puede facilitar la legibilidad del programa y su estructuración. Las diferentes fuentes son compiladas de forma separada, únicamente las fuentes que han sido modificados desde la última compilación, y después combinados con las librerías necesarias para formar el programa en su versión ejecutable.

Los comandos necesarios para compilar *linkar* y ejecutar un programa dependen del sistema operativo y debemos dirigirnos a los manuales correspondientes para conocer la sintaxis exacta.



6.11 CONSTANTES, VARIABLES Y OPERADORES

CONSTANTES

Una constante en C es un valor establecido en el programa. El lenguaje tiene cuatro tipos establecidos: constantes enteras, constantes de coma flotante, constantes de carácter y constantes de cadenas de caracteres.

Las siguientes reglas se aplican a todas las constantes numéricas.

1. no se pueden incluir comas ni espacios en blanco en la constante.
2. si se desea puede ir presidida de un signo menos. Realmente el signo menos es un operador que cambia el signo de una constante positiva, aunque se puede ver como parte de la constante mínima.
3. el valor de una constante no puede exceder un límite máximo y mínimo que estén especificados.

- Carácter (char): son aquellos caracteres de ascci que se muestran entre comillas simples o aquellos enteros entre 0 y125.
- Enteros (int): cualquier número entre el rango establecido en la aplicación.
- Float cualquier número en el rango del tipo float.

Además se incorporan constantes que solo son aplicables en una cadena de caracteres:

\b espacio atrás
\f salto de página
\n nueva línea
\r salto de carro
\” dobles comillas
\’ comillas simples

\0 carácter nulo o terminación de línea
\\ barra invertida
\v tabulación vertical
\o constante octal
\x constante hexadecimal
\+ tabulador horizontal.

VARIABLES

En C los nombres usados para referenciar funciones y otros objetos definidos por el programador se conocen como identificadores. Los identificadores deben tener las siguientes características:

1. el primer carácter debe ser una letra o el carácter de subrayado (_). La extensión del nombre no puede exceder a treinta caracteres pero solo seis son significativos.
2. solo son válidos dentro del nombre de caracteres alfanuméricos o el carácter de subrayado, es decir, solo se pueden usar letras, números y el carácter de subrayado para definir un identificador.
3. C diferencia mayúsculas y minúsculas.
4. un nombre de identificador no puede ser igual a una palabra clave del lenguaje, a una función escrita o a una función definida en la biblioteca de C.

Variable: es un identificador que se utiliza para representar cierto tipo de información dentro de un determinado programa, una variable contendrá un dato simple, una cantidad numérica o una cadena de caracteres.

MODIFICADORES DE TIPO

TIPO	ORDEN DE FORMATO	LONGITUD EN BIT	RANGO
Char	%c	8	Caracter ascci
Unsigned char	%c	8	0 a155
Signed char	%c	8	-128 a122
Int	%d%	16	-32768 a32767
Unsigned int	%d%	16	0 a 65535
Signed int	%d%	16	Igual a int
Short int	%d%	8	-128 a 127
Unsigned short int	%hu	8	0 a 255
Signed short int	%hu	8	Igual a short in
Long int	%li %ld	32	-2147483648 a 2147836449
Signed lon int	%li %ld	32	Igual a long int
Unsigned long int	%li %ld	32	0 a 4294967296
Float	%f %	32	Aprox 6 dig de precisión
Double	%e %f %g	64	Aprox 12 dig de precisión
Long double	%e %f %g	128	Aprox 128 dig de precisión

OPERADORES *ARITMÉTICOS:*

- resta
+ Suma
* Multiplicación
/ División
% módulo de división 10%3=1
-- decremento a=100 --a=99
++ Incremento a=100 ++a=100

RELACIÓN:

> Mayor que

>= mayor o igual que

< Menor que

<= menor o igual que

== igual

¡= diferente

LÓGICOS:

&& y

¡! o

¡ no.

6.12 JERARQUIA DE OPERACIONES

El problema de no tomar en cuenta la jerarquía de los operadores al plantear y resolver una operación casi siempre conduce a resultados muchas veces equivocados como estos:

Ejemplos:

2+ 3* 4 = 20 (incorrecto)

= 14 (correcto)

---

b) si calif1= 60 y calif2 = 80 y si entonces en programa se usa

Promedio = calif1 + calif2/2

La pc daría como resultado promedio = 100

Recordar siempre, que antes de plantear una formula en un programa se deberá evaluar contra el siguiente:

Orden de operaciones:

1.- Paréntesis

2.- Potencias y raíces

3.- Multiplicaciones y divisiones

4.- Sumas y restas

5.- Dos o más de la misma jerarquía u orden, entonces resolver de izquierda a derecha.

6.13 PRINCIPIOS DE TURBO C

Una de las primeras diferencias que se verán en Turbo C es que es sensible al tamaño, es decir, trata las letras mayúsculas como caracteres separadas. Por ejemplo los nombres de variable contador, Contador, CONTADOR representan tres formas diferentes de la misma variable.

Para empezar, ejecutar Turbo C y seleccionar la opción Edit. Después, introducir el programa.

Un ejemplo de Turbo C es:

```
#include <stdio.h>
```

```
/* Programa de ejemplo */
main ( )
{
   int edad;
   edad = 37;
    printf (“Mi edad es %d \ n”, edad);
}
```

Después de acabar la edición, pulsar F10 para volver al menú principal. Para compilar y ejecutar el programa, seleccionar Run del menú principal y la opción Run del submenú. Cuando acabe de compilarlo, se borrará la pantalla y Turbo C visualizará la línea (Mi edad es 37), seguida por un retorno de carro (enter) y una nueva línea.

Ahora comenzaremos por examinar el programa.

La primera línea

```
#include <stdio.h>
```

Incluye el archivo stdio.h en la compilación. Este archivo contiene la información que necesita el programa para asegurar el correcto funcionamiento de las funciones de librería estándar (E/S) de Turbo C.

La segunda línea

```
/* Programa de ejemplo */
```

es un comentario.

Los comentarios empiezan con la secuencia /* y terminan con la */. Turbo C ignora todo lo que está entre los símbolos de comienzo y final del comentario.

Las líneas vacías están permitidas y no afectan el programa, como se observa en el programa de ejemplo.

La Línea
main ( )

especifica el nombre de una función, con esta función los programas de Turbo C comienzan su ejecución.

La siguiente línea consta únicamente de una llave de apertura que significa el comienzo de la función main ( ).

La primera línea de código dentro de la función main ( ) es
int ( )

declara la variable llamada edad y le dice al compilador que edad es un entero. En Turbo C se declaran todas las variables antes de usarlas (especificación del nombre de la variable y su tipo).

La siguiente línea
edad = 37;

es una sentencia de asignación, aquí se observa que el signo de igual es para la asignación, así como también que termina con punto y coma, todas las sentencias en Turbo C acaban con un punto y coma.

La próxima línea, que saca la información a la pantalla, es
printf (“Mi edad es %d \ n”, edad);

Esta función es importante por dos razones: es un ejemplo de una llamada a una función e ilustra el uso de una función estándar de salida de Turbo C printf (). Esta línea de código consta de dos partes: el nombre de la función, que es printf () y sus dos argumentos: mi edad es %d \ n y edad.

El primer argumento es una cadena de caracteres que puede contener caracteres o códigos. Un código de formato comienza con el signo del tanto por ciento, cuando Turbo C encuentra %d, actúa como una señal de que un entero se visualizara en forma decimal.

El segundo argumento presenta el valor que se visualizara – en este caso, edad-. \ n es un código de formato especial que la dice a printf () que envié una secuencia de retorno de carro y alimentación de línea llamado new-line (nueva línea) en la terminología C.

Llamar a una función es fácil. Simplemente escribir su nombre y proporcionar los argumentos necesarios. (Un argumento es un valor pasado a la función cuando se llama.)

La última línea del programa es una llave cerrada, que indica el final de la función main ( ). Cuando Turbo C encuentra el final de main ( ), se termina la ejecución del programa.

En Turbo C los errores que se puedan cometer en el programa se detectan al tratar de compilar el programa, turbo C ilumina el primer error en la ventana de mensajes, así como ilumina la línea en el programa en la que se detecta el error, no hay que preocuparse del segundo mensaje de error ya que se generó debido a que el primer error hizo que Turbo C interpretase la siguiente línea de un modo incorrecto.

En el siguiente programa ilustra una segunda función de biblioteca llamada scanf ( ), que se usa para leer información introducida por un usuario.

```
#include <studio.h>
/ * Programa para convertir pies a metros */
main  ( )
{
int pies;
     float metros;
printf (“Introduce el número de pies: “ );
   scanf (“%d”, &pies);
metros = pies * 0.3084; /* conversión de pies a metros */
printf (“%d pies es %f metros \ n”, pies, metros);
}
```

El programa introduce algunas características nuevas importantes. Primero declara dos variables: pies del tipo int y metros del tipo flota, que significa que puede tener un componente fraccional.

Scanf ( ) es una función de biblioteca que se utilizó en el programa para leer un entero introducido por el teclado. El %d en el primer argumento le dice a scanf () que lea un entero y lo ponga en la variable que sigue. El &que precede a pies es necesario para que scanf ( ) funcione adecuadamente. El * significa la multiplicación.

Un programa C es una colección de una o más funciones. Cada función tiene un nombre y una lista de argumentos que recibe la función. En general se puede dar a una función dada en nombre que se quiera, con la excepción de main, que reserva C para la función que comienza la ejecución del programa. Una función tendrá paréntesis después del nombre de la función.

Un argumento de una función es un valor que se pasa a la función en el momento que se llama. El término argumento se refiere al valor usado para llamar a la función. El término parámetro formal se refiere a la variable de una función que recibe el valor de los argumentos en su llamada. Las funciones que toman argumentos se llaman funciones parametrizadas. La distinción importante que hay que entender es que la variable utilizada como argumento en una llamada de función no tiene nada que ver con el parámetro formal que recibe su valor.

El tipo de argumento que se utiliza para llamar a una función debe ser el mismo tipo que el parámetro formal que recibe el argumento.

El formato general de una función de C:

```
        Nombre de función tipo – return (lista de parámetros)
{
  Cuerpo del código
}
```

Para las funciones sin parámetros, no habrá lista de parámetros.

6.14 OTRAS FUNCIONES DE I/O

CAPTURA:

a) Getchar ( ); Getch ( ); Getche ( );

Estas tres instrucciones están especializadas en captura de un solo carácter y sustituyen a scanf %c en situaciones especiales.

Estas funciones trabajan por igualación, es decir para usarse

Varchar = getchar ( );

```
ej;
char vocal;
Vocal = getchar ( );
```

NOTA: DEPENDIENDO DEL SISTEMA OPERATIVO EN OCASIONES UN GETCHAR ( ) NO ATRAPA EL CARACTER A LA PRIMERA VEZ, EN ESTE CASO USAR MAS DE UN GETCHAR EN FORMA CONTINUA.

En este caso poner más getchar ( ); después de getchar ( ) hasta que capture el carácter o también para detener la corrida.

Getchar es un macro que lee un carácter del teclado y regresa el carácter leído.

La diferencia es que getchar y getche devuelven el carácter leído a la pantalla y el getch no devuelve el carácter a la pantalla.

Estas instrucciones deberán usarse en las siguientes situaciones:

a) Cuando la captura es de un solo carácter
b) También para retener pantallas en este caso sin igualdad
c) Para evitar los saltos en capturas, cuando se captura un entero y una string después, en este caso es conveniente usar una serie de getchar ( ) para quitar el carácter de return del teclado, en particular es conveniente usar un getchar después de un scanf (entero) como lo muestra el ejemplo de abajo prog2.cpp.

B) GETS ( );

Esta función es la alternativa para captura de strings que tengan espacios en blanco intermedios cosa que scanf %s no puede hacer.

Su formato completo es:

Gets (variable string);

DESPLIEGUE:

a) Putch ( ); Putchar ( );

Estas instrucciones despliegan directamente un carácter en el dispositivo de salida standar.

A diferencia de getchar, getch y getche, estas instrucciones si llevan un argumento.

Ejemplo;

```
putchar (alfa);
putchar ('\n');
putchar ('\064');
```

b) Puts (variable string).

Es la función complemento de gets esta función despliega una string en el dispositivo de salida standar y debe llevar el argumento o parámetro.

```
Puts (nombre);
Puts ("pato").
```

Programa ejemplo

```
#include <stdio.h>
Void main ( )
{
```

```
/*Declaración*/
int var1, var2,var3;
char var4[15], var5[15];
/*Captura y observar donde se ponen getchars
Unos para limpiar return y otros para para parar la pantalla*/
printf ("dame entero1:");
scanf ("%d",&var1);getchar ( );
printf ("dame string1:");
scanf ("%s",&var4);
printf ("dame entero2:");
scanf ("%d",&var2);
printf ("dame entero3:");
scanf ("%d",&var3); getchar ( );
printf ("dame string2:");
Gets (var5);
/*Despliegue*/
printf ("entero1 %d \n",var1);
printf ("string1 %s \n",var4);
printf ("entero2 %d \n" ,var2);
printf ("entero3 %d \n",var3);
printf ("string2 %s \n",var5);
getchar ( );
}
```

Recordar que scanf ("%s") no quiere espacios en blanco, por eso es mejor usar gets(var string).

Observar que después de un scanf (entero) que este antes de un gets (varstring) deberán usar un getchar ( ) para quitar el return del teclado o la pc no va a capturar la string sino que va a saltarse.

FORMATO PANTALLA

#include<conio.h> Librería a usar

1.- clrscr ( ); Limpia pantalla

ESTA INSTRUCCIÓN NO OCUPA ACLARACIÓN

2.- gotoxy (#de col,#de ren); posiciona cursor en lugar indicado

RECORDAR QUE UNA CONSOLA NORMAL TIENE 24 RENGLONES Y 80 COLUMNAS

En particular poner un gotoxy ( ); antes de cada printf ( ) en el programa.

REARREGLAR LOS PROGRAMAS CONSTRUIDOS PARA SU MEJOR PRESENTACIÓN.

PALABRAS RESERVADAS C++

auto	break	case	char	const	continue	default	do
double	Else	enum	extern	float	for	goto	if
int	Long	register	return	short	signed	sizeof	static
Struct	Switch	typedef	union	unsigned	void	volatile	while

6.15 IDENTIFICADORES Y TIPOS

Para dar un nombre a una variable tenemos que usar un identificador. La longitud de un identificador puede variar entre uno y varios caracteres, por lo general, 32. En la mayoría de los casos el primer carácter debe ser una letra o un símbolo de subrayado. A continuación se muestran varios ejemplos de nombres de identificadores correctos e incorrectos:

Correcto	Incorrecto
cuenta	1cuenta
prueba23	hola!
puerto_paralelo	puerto..paralelo

El lenguaje C es **sensible a mayúsculas y minúsculas** (*case sensitive*), de modo que no es lo mismo para el compilador el identificador cuenta con otro denominado Cuenta.

Los compiladores reservan determinados términos o palabras claves (*keywords*), para el uso sintáctico del lenguaje, tales como: asm, auto, break, case, char, do, for, etc. Si bien estas palabras están definidas para el ANSI C, los distintos compiladores extienden esta definición a otros términos, por lo que es aconsejable leer la tabla completa de palabras reservadas del compilador que se vaya a usar, para no utilizarlas en nombres de variables.

Para crear una variable en un lugar determinado de un programa escribiremos primero el tipo de variable y luego el identificador con el que queremos nombrar la variable, seguido todo de un ';'. A esto se le denomina **definir una variable**. La forma general de la definición es:

tipo lista_identificadores;

Por ejemplo:

```
int numero;   /* crea la variable número, de tipo número entero */
char letra;   /* crea la variable letra, de tipo carácter*/
float a, b;   /* crea dos variables a y b, de tipo número de coma flotante */
```

Las variables se pueden inicializar, es decir, establecer un valor inicial, en el momento de creación. Para ello, detrás del identificador ponemos el carácter '=' seguido del valor inicial. Los valores iniciales pueden ser cualquier constante válida para el tipo de variable que creemos. Por ejemplo:

```
int numero = 0;       /* crea la variable entera número  y la inicializa a 0*/
char letra = 'p';     /* crea la variable carácter letra y la inicializa a 'p' */
float a=57.1, b=1.2E-5;   /* crea las variables a y b, de tipo número de
coma flotante, y los inicializa  a 57.1 y 0.000012, respectivamente*/
```

Existen tres sitios donde se pueden definir variables:

- fuera de todas las funciones (variables globales)
- dentro de las funciones (variables locales)
- en la definición de parámetros de funciones (parámetros formales).

El tipo de datos carácter consta de un único carácter y se suele representar por su carácter en código ASCII situado entre apóstrofes. Por ejemplo:

```
'p' /* la letra p minúscula */
'1' /* el número 1 */
' ' /* el carácter en blanco */
```

Hay otras formas de representar caracteres, que se emplean cuando es un carácter que no se puede introducir directamente desde el teclado. Para ello debemos conocer su código ASCII. Para representar el carácter de número ASCII 27, (el código para el carácter ESCAPE), basta colocar el número ASCII en el sistema octal precedido de la barra atrás, y todo ello entre apóstrofes, tal y como hacemos para los demás caracteres:

```
'\27' /* representa el código ESCAPE, de ASCII 27 */
```

En C hay algunos caracteres especiales que se usan frecuentemente. Estos caracteres tienen una representación especial. Algunos de ellos son:

Código	Significado	Valor ASCII (decimal)	Valor ASCII (hexadecimal)
'\n'	nueva línea	10	0x0A
'\r'	retorno de carro	13	0x0D
'\f'	nueva página (*form feed*)	2	x0C
'\t'	tabulador horizontal	9	0x09
'\b'	retroceso (*backspace*)	8	0x08
'\''	comilla simple	39	0x27
'\"'	Comillas	4	0x22
'\\ '	barra invertida	92	0x5C
'\? '	Interrogación	63	0x3F
'\nnn'	cualquier carácter (donde nnn es el código ASCII expresado en octal)		
'\xnn'	cualquier carácter (donde nn es el código ASCII expresado en hexadecimal)		

Un tipo entero consiste en un número sin parte decimal, aunque puede tener signo. Generalmente con el tipo de datos enteros no representamos números muy grandes. Son ejemplos:

0,124, -2000

El tipo de datos enteros nos permitir hacer operaciones aritméticas, como la suma y la multiplicación. El tipo de datos enteros es quizás el más importante de todos los tipos de datos, y muchas veces es el tipo de datos por defecto, es decir, cuando el compilador se encuentre con un dato y no le hayamos indicado cuál es su tipo, supondrá que es un entero.

C también posee el tipo de datos en coma flotante, que es el que usan las calculadoras científicas en sus operaciones aritméticas.

Los **tipos** de datos atómicos definidos por C son:

- **caracteres**
- **números enteros**
- **números en coma flotante**

Los caracteres son representados por char, los enteros por short, int, long, y los números en coma flotante por float y double. Los cinco tipos de datos atómicos son:

Identificador	Tipo	Tamaño en bits	Intervalo
Char	Carácter	8 (normalmente)	0 a 255
Int	Entero con signo	16 (depende de la implementación)	-32 768 a 32 767
Float	Flotante simple	32 (normalmente)	3.4E-38 a 3.4E+38 (aprox. 6 dígitos de precisión)
Doublé	Flotante doble	64 (normalmente)	1.7E-308 a 1.7E+308 (aprox. 12 dígitos de precisión)
Void	Nada Genérico	0	sin valor

El tipo representado por void puede significar dos cosas distintas, según su utilización. Puede significar nada, o sea, si una función devuelve un valor de tipo

void no devuelve ningún resultado; o puede significar cualquier cosa, como puede ser un puntero a void (un puntero genérico a cualquier tipo de dato). Más adelante veremos su utilización.

A excepción del tipo void, los otros tipos básicos admiten variantes. Esto se consigue utilizando **modificadores de tipo** que precedan la definición del tipo:

- **signed**
- **unsigned**
- **long**
- **short**

La palabra unsigned en realidad es un modificador aplicable a tipos enteros, aunque si no se especifica un tipo se supone int. El modificador es una palabra clave de C que indica que una variable, o función, no se comporta de la forma normal. Hay también un modificador signed, pero como los tipos son por defecto con signo, casi no se utiliza. Las variables de coma flotante son siempre con signo y, en el caso en que el exponente sea positivo, puede obviarse el signo del mismo. La siguiente tabla muestra todas las combinaciones permitidas que se ajustan al estándar ANSI.

Tipo	bits	Intervalo
char	8	-128 a 127
unsigned char	8	0 a 255
signed char	8	-128 a 127
int	16	-32 768 a 32 767
unsigned int	16	0 a 65535
signed int	16	igual que int
short int	8	-128 a 127
unsigned short int	8	0 a 255
signed short int	8	igual que short int

long int	32	-2 147 483 648 a 2147 483 647
signed long int	32	-2 147 483 648 a 2 147 483 647
unsigned long int	32	0 a 4 294 967 296
float	32	3.4E-38 a 3.4E+38
double	64	1.7E-308 a 1.7E+308
long double	128	3.4E-4932 a 3.4E+4932 (aprox. 24 dígitos de precisión)

Es preciso tener presente que en sistemas que permiten longitudes de palabra mayores de 16 bits un **short int** puede no ser igual que un **signed char**.

Veamos el siguiente programa de ejemplo acerca del uso de variables:

```
#include <stdio.h>  main( ) {       float cels, fahr;       cels = 25.0;
        /* Temperatura en º C  */   fahr = 32.0 + 9.0 * fahr / 5.0;       /*
Temperatura en º Fahrenheit */   printf(">>> %f ºC son %f ºF\n", cels, fahr );  }
```

En él se definen dos variables float, se asigna un valor a la primera y se calcula la segunda mediante una expresión aritmética. Las asignaciones en C también son una expresión, por lo que se pueden utilizar como parte de otra expresión, pero según que prácticas de este tipo no son muy recomendables ya que reducen la legibilidad del programa. En la instrucción printf, el símbolo %f indica que se imprime un número en coma flotante.

Un tipo de datos importante en C es la **cadena de caracteres**. Está formada por un conjunto de caracteres encerrados entre comillas. Podemos usar todos los caracteres del conjunto ASCII, incluso los especiales. Los caracteres normales se incluyen entre las comillas tal cual, sin necesidad de apóstrofes, y los especiales se incluyen utilizando la representación del C. Por ejemplo:

"Bienvenido a la programación en lenguaje C\n"

En este ejemplo observamos la cadena de caracteres "Bienvenido a la programación en lenguaje C", a la que hemos añadido un carácter de retorno de carro al final. El motivo de ello es que cuando la imprimamos el carácter de retorno de carro, \n, actuará como una orden que obligar al cursor a avanzar una línea y situarse al principio de la siguiente.

6.16 TAMAÑO DE LAS VARIABLES

En muchos programas es necesario conocer el tamaño (cantidad de bytes) que ocupa una variable; por ejemplo, en el caso de querer reservar memoria para un conjunto de ellas. Lamentablemente, este tamaño es dependiente del compilador que se use, lo que producirá, si definimos rígidamente (con un número dado de bytes) el espacio requerido para almacenarlas, un problema seria si luego se quiere compilar el programa con un compilador distinto del original. Para salvar este problema y mantener la portabilidad, es conveniente que cada vez que haya que referirse al tamaño en bytes de las variables, se haga mediante un operador llamado sizeof que calcula sus requerimientos de almacenaje. Está también permitido el uso de sizeof con un tipo de variable, es decir:

Sizeof (int); sizeof (char); sizeof (long double); etc.

6.17 DURACIÓN DE LAS VARIABLES

Las variables pueden ser de dos tipos:

- **estáticas**
- **dinámicas**

Las estáticas se crean al principio del programa y duran mientras el programa se ejecute. Las variables son dinámicas si son creadas dentro de una función. Su

existencia está ligada a la existencia de la función. Se crean cuando la función es llamada y se destruyen cuando la función o subrutina devuelve el control a la rutina que la llamó.

Las variables estáticas se utilizan para almacenar valores que se van a necesitar a lo largo de todo el programa. Las variables dinámicas se suelen utilizar para guardar resultados intermedios en los cálculos de las funciones.

Como regla general, una variable es estática cuando se crea fuera de una función y es dinámica cuando se crea dentro de una función.

Por ejemplo en el siguiente programa:

```
#include <stdio.h>  int número 1 = 1;  main ( ) {          int número 2 = 2;     printf ("%d, %d\n", numero1, numero 2); }
```

Hemos creado la variable estática número 1, que dura todo el programa, y la variable número 2, que dura sólo mientras se ejecuta la función main ( ). En este programa tan pequeño, la función main ( ) es la que ocupa todo el tiempo de ejecución, por lo que no apreciaremos diferencia en el uso de ambas, aunque más adelante si se verá su uso.

6.18 ÁMBITO O ALCANCE (SCOPE) DE LAS VARIABLES

Otra característica de las variables es su alcance. El alcance se refiere a los lugares de un programa en los que podemos utilizar una determinada variable. Distinguiremos así dos tipos principales de variables:

- **globales**
- **locales**

Una variable es global cuando es accesible desde todo el programa, y es local cuando solo puede acceder a ella la función que la creó. También hay una norma general para el alcance de las variables: una variable es global cuando se define fuera de una función, y es local cuando se define dentro de una función. En nuestro ejemplo anterior número 1 es una variable global y número 2 es una variable local.

Variable Global	Variable Local
#include <stdio.h> int x; main ( ) { }	#include <stdio.h> main ( ) {

Dentro de las variables globales hay dos tipos: las que son accesibles por todos los ficheros que componen nuestro programa y las que son accesibles solo por todas las funciones que componen un fichero. Esto es debido a que normalmente los programas en C se fragmentan en módulos más pequeños, que son más fáciles de manejar y depurar. Por ello hay veces que nos interesar que una variable sea accesible desde todos los módulos, y otras solo queremos que sea accesible por las funciones que componen un determinado modulo. Por defecto todas las variables globales que creemos son accesibles por todos los ficheros que componen nuestro programa.

6.19 ESPECIFICADORES DE CLASE DE ALMACENAMIENTO Y MODIFICADORES DE ACCESO

Podemos fácilmente modificar el alcance y la duración de una variable que tiene asignado por defecto. Esto es una operación muy común y útil. Para hacerlo antepondremos al tipo de la variable un **especificador**, que es una palabra reservada, que cambia esta característica.

Especificador(o modificador) tipo identificador

El primero es la palabra clave **static**. Cuando a una variable local se le añade static pasa de ser dinámica a ser estática. Así la duración de la variable se amplía a la duración del programa completo. Obsérvese que una variable estática sólo se crea una vez, al principio del programa, por lo que la inicialización solo se produce una vez. Se dice que tienen memoria asignada durante toda la ejecución del programa. Su valor es recordado incluso si la función donde está definida acaba y se vuelve a llamar más tarde. Ejemplo:

```
series (void) {          static int num;          num = num + 23;     return (num); }
```

Además, el especificador static tiene otro uso. Si se lo añadimos a una variable global, definida fuera de una función, entonces modificamos su alcance; pasa de tener alcance global a todos los ficheros del programa a ser solo accesible por las funciones del fichero en el que se crea.

Otro especificador usual es **extern**. Éste se usa cuando una variable que se creó en otro módulo se quiere usar en el actual. Cuando añadimos a la variable este especificador el compilador queda advertido de que la variable ya existe en otro módulo, por lo que el compilador no tiene que crearla, sino simplemente usarla. Entonces a este tipo de proceso se le llama **declaración** de tipo de variable. Por ejemplo:

```
extern int numero;  main ( ) {        printf ("%d\n", numero); }
```

Es un programa en el que declaramos la variable externa número, que habremos creado en otro módulo.

Una diferencia muy importante entre una definición y una declaración es que en la declaración no se reserva espacio en la memoria para la variable, y en la definición si se crea.

Hay otros especificadores que no son tan usuales: **auto** y **register**. El modificador auto indica que una variable local es dinámica (en la terminología del C, automática). Observar que por defecto las variables locales a una función son automáticas, por lo que no se usa. Sin embargo todos los compiladores la reconocen y no protestan si la usamos. Por ejemplo:

```
Main ( ) {      auto número = 1;    printf ("%d\n", numero); }
```

Crea una variable automática entera. Si quitamos auto el programa no se diferencia.

El especificador **register** se usa más a menudo, sobre todo en la llamada "programación de sistemas". Recordemos que el C fue creado para este tipo de programación. Le pide al compilador que almacene la variable en un registro de la máquina, que es el lugar más eficiente para guardar las variables. Esto se hace porque el trabajo con los registros del procesador es mucho más rápido que el trabajo con la memoria central. Hay dos detalles importantes: normalmente no hay muchos registros libres en el procesador, pues el compilador los usa para otros propósitos. Entonces el especificador register es más bien un ruego que una orden. Otro aspecto es que muchos compiladores realizan trabajos de optimización, que son modificaciones en el código que generan que hace trabajar al programa más de prisa. Aun así, en rutinas críticas, que deben ejecutarse de prisa se suele usar. Sólo se puede aplicar a variables locales y a los parámetros formales de una función. Son ideales para el control de bucles.

```
potencia_ent (int m, register int e) {      register int temp;    temp=1;      for ( ;
e; e--) temp *= m;    return (temp); }
```

El modificador volatile se usa para decirle que el contenido de la variable puede ser modificado en cualquier momento desde el exterior de nuestro programa, sin

que podamos evitarlo. Lo que hace el modificador es instruir al compilador para que lea de nuevo el valor de la variable de la memoria cuando tenga que usarlo. Este modificador evita que el compilador no genere código para optimizar la variable. Evidentemente el uso de volatile excluye el uso de register y viceversa.

El modificador **const** se usa para indicar que el contenido de la variable no puede ser cambiado durante la ejecución del programa. El compilador es libre de situar las variables de este tipo en memoria de sólo lectura (ROM). Por ejemplo:

```
const int a;
```

crea la variable entera a, que no puede ser modificada por el programa pero sí ser usada en otros tipos de expresiones.

6.20 FUNCIONES E/S

6.20.1 INTRODUCCIÓN: FUNCIONES DE BIBLIOTECA PARA E/S

En la biblioteca estándar de C se definen las dos principales vías de comunicación de un programa:

- la **entrada estándar (stdin)**, y
- la **salida estándar (stdout)**.

Generalmente están ambas asociadas a nuestro terminal, de manera que cuando se imprimen datos en la salida estándar los caracteres aparecen en el **terminal**, y cuando leemos caracteres de la entrada estándar los leemos del **teclado** del terminal. La entrada y salida estándar trabajan con caracteres (en modo carácter), con datos o números binarios. Es decir, todos los datos que enviemos a la salida estándar deben ser cadenas de caracteres. Por ello para imprimir cualquier dato en la salida estándar primero deberemos convertirlo en texto, es decir, en cadenas

de caracteres. Sin embargo esto lo haremos mediante las funciones de biblioteca, que se encargan de realizar esta tarea eficientemente:

- putchar y getchar
- puts y gets
- printf
- scanf
- gotoxy
- clrscr

6.20.2 PUTCHAR Y GETCHAR

Comenzaremos con las dos funciones principales de salida y entrada de caracteres: putchar y getchar.

La función putchar escribe un único carácter en la salida estándar. Su uso en sencillo y generalmente está implementada como una macro en la cabecera de la biblioteca estándar. Por ejemplo:

```
#include <stdio.h>  main() {        putchar('H');  putchar('o');   putchar('l');
        putchar('a');   putchar(32);          putchar('m'); putchar('u');   putchar('n');
        putchar('d');   putchar('o');   putchar('\n'); }
```

El resultado es:

```
Hola mundo
```

En el código anterior putchar (32); muestra el espacio entre ambas palabras (32 es el código ASCII del carácter espacio ' ') y putchar ('\n'); imprime un salto de línea tras el texto.

La función getchar devuelve el carácter que se halle en la entrada estándar. Esta función tiene dos particularidades. La primera es que aunque se utiliza para obtener caracteres no devuelve un carácter, sino un entero. Esto se hace así, ya que con un entero podemos representar tanto el conjunto de caracteres que cabe en el tipo carácter (normalmente el conjunto ASCII de caracteres) como el carácter EOF de fin de fichero. En UNIX es habitual representar los caracteres usando el código ASCII, tanto en su versión de 7 bits como en su versión ampliada a 8 bits. Estos caracteres se suelen representar como un entero que va del 0 al 127 o 256. El carácter EOF entonces es representado con un -1. Además esto también lo aplicaremos cuando leamos los ficheros binarios byte a byte. Un ejemplo:

```
#include <stdio.h>  main ( ) {      int c;   c = getchar ( );      /*  Nótese  que
getchar ( ) no devuelve nada                    hasta  que  se  presiona  ENTER
*/       putchar (c); }
```

Aquí se almacena en el entero C el carácter pulsado en el teclado. Posteriormente se muestra con putchar.

6.20.3 PUTS Y GETS

La función puts simplemente imprime una cadena de caracteres en la salida estándar (y produce un salto de línea). Le debemos proporcionar la dirección donde encontrar la cadena de caracteres. El código:

```
#include <stdio.h>  main ( )  {      puts ("Bienvenido a la programación");  puts
(" en lenguaje C"); }
```

Produce el resultado:

```
Bienvenido a la programación
```

```
en lenguaje C
```

La función gets simplemente toma una cadena de caracteres de la entrada estándar (cuya introducción es preciso terminar con un ENTER) y la almacena en una variable *string*. Supongamos este código:

```
#include <stdio.h>  main() {         char cadena[50];     puts ("Escriba un texto:");
        gets (cadena);         puts ("El texto escrito es:");         puts (cadena); }
```

La declaración char cadena [50]; crea una variable llamada cadena que puede almacenar hasta 50 caracteres. Este código produce, cuando escribimos con el teclado el texto Bienvenido a la programación en lenguaje C, el resultado:

```
Escriba un texto:

Bienvenido a la programación en lenguaje C

El texto escrito es:

Bienvenido a la programación en lenguaje C
```

Esta función nos permite introducir frases enteras, incluyendo espacios.

Algunos detalles más; la sintaxis es:

*char *gets (char *buffer);*

Si se ha almacenado algún carácter en buffer le añade un '\0' al final y devuelve un puntero a su dirección. Si no se ha almacenado ninguno (error o carácter fin-de-fichero, CTRL+Z) devuelve un puntero NULL.

```
#include <stdio.h>  main ( ) {  char cadena [50];  char *p;  puts ("Escriba un texto:");  p = gets (cadena);  if (p) {  puts ("El texto escrito es:");  puts (cadena);  }  else  puts ("No se ha guardado nada."); }
```

Esta función puede ser peligrosa porque no comprueba si nos hemos pasado del espacio reservado (de 50 caracteres en este ejemplo: 49 caracteres + '\0').

6.20.4 PRINTF: IMPRESIÓN EN PANTALLA

Para imprimir datos de un modo más general el C dispone de la función printf, que se ocupa de la impresión con formato en la salida estándar. Cuando se empieza con un nuevo lenguaje suele gustar el ver los resultados, ver que nuestro programa hace "algo". Por eso la mayor parte de programas de principiantes utilizan rápidamente la función printf, que sirve para sacar información por pantalla.

Para utilizar la función printf en nuestros programas debemos incluir la directiva:
#include <stdio.h>

Al principio del programa, como se ha visto en el primer programa de la sección Estructura de un programa en C. Esto es porque esta función se halla definida en el archivo de cabecera stdio.h de la siguiente manera:

*int printf (const char *format [, argumentos, ...]);*

Si sólo queremos imprimir una cadena basta con hacer:

```
Printf ("Cadena");
```

Esto dará como resultado en la pantalla:

```
Cadena
```

Lo que se ponga entre las comillas es lo que se observa en la pantalla.

Si volvemos a usar otro printf, por ejemplo:

```
#include <stdio.h>  main ( ) {      printf ("Cadena");    printf ("Segunda"); }
```

Obtendremos:

```
Cadena Segunda
```

Este ejemplo nos muestra cómo funciona printf. Para escribir en la pantalla se usa un cursor que no vemos. Cuando escribimos algo el cursor va al final del texto. Cuando el texto llega al final de la fila, lo siguiente que pongamos irá a la fila siguiente. Si lo que queremos es sacar cada una en una línea deberemos usar "\n". Es el indicador de retorno de carro. Lo que hace es saltar el cursor de escritura a la línea siguiente:

```
#include <stdio.h>  main ( ) {      printf ( "Cadena\n" );      printf   (Segunda");
}
```

Y tendremos:

```
Cadena
Segunda
```

También podemos poner más de una cadena dentro del printf:

```
Printf ("Primera cadena" "Segunda cadena");
```

Lo que no podemos hacer es incluir *cosas* entre las cadenas:

```
Printf ("Primera cadena" texto en medio  "Segunda cadena");
```

Esto no es válido. Cuando el compilador intenta interpretar esta sentencia se encuentra "Primera cadena" y luego texto en medio, no sabe qué hacer con ello y da un error.

Pero, ¿qué pasa si queremos imprimir el símbolo " en pantalla? Por ejemplo imaginemos que queremos escribir: Esto es "extraño". Si para ello hacemos:

```
Printf ("Esto es "extraño");
```

Obtendremos unos cuantos errores. El problema es que el símbolo "se usa para indicar al compilador el comienzo o el final de una cadena. Así que en realidad le estaríamos dando la cadena "Esto es", luego extraño y luego otra cadena vacía". La función printf no admite esto y de nuevo tenemos errores. La solución es usar \". Veamos:

```
Printf ("Esto es \"extraño\"");
```

Esta vez funcionará. Como vemos, la contrabarra '\' sirve para indicarle al compilador que escriba caracteres que de otra forma no podríamos. Pero, ¿y si lo que queremos es usar '\' como un carácter normal e imprimir, por ejemplo, Hola\Adiós? La respuesta es volver a usar '\':

```
Printf ("Hola\\Adiós");
```

Esta doble '\' indica a C que queremos es mostrar una contrabarra.

Como hemos visto, la sintaxis de printf acepta un *string* de formato (const char *format) y cualquier número de argumentos. Estos se aplican a cada uno de los especificadores de formato contenidos en format. Un especificador de formato toma la forma *% [flags] [width] [.prec] [h|lL] type*.

El tipo (*type*) puede ser:

d, i	entero decimal con signo
o	entero octal sin signo
u	entero decimal sin signo
x	entero hexadecimal sin signo (en minúsculas)
X	entero hexadecimal sin signo (en mayúsculas)
f	coma flotante en la forma: [-]dddd.dddd
e	coma flotante en la forma: [-]d.dddd e[+/-]ddd
g	coma flotante según el valor
E	como e pero en mayúsculas
G	como g pero en mayúsculas
c	un carácter
s	cadena de caracteres terminada en '\0'
%	imprime el carácter %
p	puntero

Los *flags* pueden ser los caracteres:

+	siempre se imprime el signo, tanto + como -

-	justifica a la izquierda el resultado, añadiendo espacios al final
blank	si es positivo, imprime un espacio en lugar de un signo +
#	especifica que el argumento se convertirá según una forma alternativa

En el campo *width* se especifica la anchura mínima del valor de salida:

n	Se imprimen al menos *n* caracteres.
0n	se imprimen al menos *n* caracteres y, si la salida es menor, se anteponen ceros
*	la lista de parámetros proporciona el valor de anchura

Hay tres modificadores de tamaño, para los tipos int y double:

h	imprime un entero short
l	imprime un entero long
L	imprime un double long

Si se usa el flag # con una conversión de formato de printf, tiene el siguiente efecto sobre el argumento:

c s d i u	no tiene efecto
0	antepone 0 a un argumento no nulo
x o X	antepone 0x (o 0X) al argumento
e E f	el resultado siempre contiene un punto decimal
g G	Igual que E. Los ceros sobrantes se eliminan

6.20.5 SCANF: LECTURA DEL TECLADO

Algo muy usual en un programa es esperar que el usuario introduzca datos por el teclado. Para ello contamos con varias posibilidades: Usar las funciones de la biblioteca estándar, crear nuestras propias interrupciones de teclado (MS-DOS) o usar funciones de alguna biblioteca diferente. Vamos a estudiar la primera, usando las funciones de la biblioteca estándar.

Para utilizar la función scanf debemos incluir la directiva:

```
#include <stdio.h>
```

Al principio del programa puesto que esta función se halla definida en el archivo de cabecera stdio.h de la siguiente manera:

*int scanf (const char *format [, direcciones, ...]);*

Como se observa, el uso de scanf es muy similar al de printf con una diferencia fundamental: nos da la posibilidad de que el usuario introduzca datos en vez de mostrarlos. No nos permite mostrar texto en la pantalla; por eso, si queremos mostrar un mensaje previo, usamos un printf delante. Un ejemplo:

```
#include <stdio.h>  main ( ) {  int num;  printf ("Introduce un número:");
    scanf ("%i", &num);  printf ("Has tecleado el número %i\n", num); }
```

Obsérvese que en la sentencia scanf la variable num lleva adelante el símbolo **&**. Éste sirve para indicar al compilador cuál es la dirección (o posición en la memoria) de la variable en la que se almacenará el valor introducido. Es imprescindible no olvidar este símbolo.

Scanf no mueve el cursor de su posición actual, así que en el ejemplo queda (_ indica dónde se halla el cursor):

Introduce un número _

Esto es porque en printf no hemos puesto al final el símbolo de salto de línea '\n'. Además, hemos dejado un espacio al final de Introduce un número: para que cuando tecleemos el número no se "pegue" al mensaje.

Veamos cómo funciona scanf. Fijémonos que hay una cadena entre comillas. Ésta es similar a la de printf; sirve para indicarle al compilador qué tipo de datos estamos introduciendo. Como en este caso se trata de un entero, usamos %i. Después de la coma se halla la variable donde almacenamos el dato, en este caso num.

Podemos preguntar por más de una variable a la vez en un solo scanf, de modo que es preciso poner un %i por cada variable:

```
#include <stdio.h> main ( ) { int a, b, c; printf ("Introduce tres números:"); scanf ( "%i %i %i", &a, &b, &c ); printf ("Has tecleado los números %i %i %i\n", a, b, c ); }
```

De esta forma, cuando el usuario ejecuta el programa debe introducir los tres datos separados por un espacio.

También podemos pedir en un mismo scanf variables de distinto tipo:

```
#include <stdio.h> main ( ) { int a; float b; printf ("Introduce dos números: "); scanf ("%i %f", &a, &b); printf ("Has tecleado los números %i %f\n", a, b); }
```

A cada modificador (%i, %f) le debe corresponder una variable de su mismo tipo. Es decir, al poner %i el compilador espera que su variable correspondiente sea de tipo int. Si ponemos %f espera una variable tipo float.

Veamos ahora el uso de scanf con cadenas (*strings*). La función scanf almacena en una porción temporal de memoria (un *buffer*) lo que vamos escribiendo. Cuando pulsamos ENTER (o Intro, o Return) lo analiza, comprueba si el formato es correcto y, por último, lo almacena en la variable que le indicamos. Veamos el ejemplo:

```
#include <stdio.h>  main ( ) {      char cadena [30]; printf ("Escribe una palabra:"); scanf ("%s", cadena);      printf ("He guardado: \"%s\" \n", cadena); }
```

Ejecutamos el programa e introducimos la palabra "hola". Esto es lo que tenemos:

```
Escribe una palabra: hola He guardado: "hola"
```

Si ahora introducimos "hola amigos" esto es lo que tenemos:

```
Escribe una palabra: hola amigos He guardado: "hola"
```

Sólo ha recogido la palabra "hola" y se ha olvidado de amigos. ¿Por qué? Porque scanf toma una palabra como cadena y usa los espacios para separar variables.

Es importante siempre asegurarse de que no vamos a almacenar en cadena más caracteres que los que caben. Para ello debemos limitar el número de los mismos que va a introducir scanf. Si, por ejemplo, queremos un máximo de 5 caracteres, usaremos %5s:

```
#include <stdio.h> main ( ) { char cadena[6]; printf ("Escribe una palabra:"); scanf ("%5s", cadena); printf ("He guardado: \"%s\" \n", cadena); }
```

Si introducimos una palabra de 5 caracteres (no se cuenta '\0') o menos, la recoge sin problemas y la guarda en cadena:

```
Escribe una palabra: Frodo He guardado: "Frodo"
```

Si introducimos más de 5 caracteres cortará la palabra y dejará sólo 5.

```
Escribe una palabra: Gandalf He guardado: "Ganda"
```

scanf permite controlar qué caracteres introducimos. Supongamos que sólo queremos recoger las letras mayúsculas:

```
#include <stdio.h> main ( ) { char cadena [30]; printf ("Escribe una palabra:"); scanf ("%[A-Z]s", cadena); printf ("He guardado: \"%s\" \n", cadena); }
```

Guarda las letras mayúsculas en la variable hasta que encuentra una minúscula:

```
Escribe una palabra: Aragorn He guardado: "A"
Escribe una palabra: GOLLUM He guardado: "GOLLUM"
```

Cómo funcionan los *buffer*

Veamos qué sucede cuando scanf no recoge todas las pulsaciones efectuadas. Tomemos el código:

```
#include <stdio.h>
Main ( ) {       char c;
char nombre[20], apellido[20];
print f ("Escribe tu nombre:");       scanf ( "%[A-Z]s", nombre);
printf  (Lo que recogemos de scanf es: %s\n", nombre );
printf ("Lo que había quedado en el buffer: ");
while ( (c = getchar())!= '\n' )
putchar ( c ); }
```

Imaginemos que introducimos un nombre con mayúsculas y minúsculas:

```
Escribe tu nombre: GANdalf
Lo que recogemos de scanf es: GAN
Lo que había quedado en el buffer: dalf
```

Veamos otro ejemplo en el que hay que obrar con precaución respecto a scanf:

```
#include <stdio.h>  main ( ) {       char nombre[20], apellido[20];     printf
("Escribe tu nombre: ");     scanf ("%s", nombre);       printf       ("Escribe       tu
apellido: ");    gets ( apellido ); }
```

Cuando se teclea un nombre se obtiene:

```
Escribe tu nombre: Gandalf Escribe tu apellido:
```

No hay opción a introducir el apellido puesto que en el buffer queda un ENTER (el de finalizar la introducción del nombre) que es recogido directamente por gets.

6.20.6 POSICIONADO DEL CURSOR Y BORRADO DE PANTALLA

- gotoxy: posicionado del cursor

- clrscr: borrado de la pantalla

gotoxy: posicionado del cursor (DOS)

Esta función sólo está disponible en compiladores de C que dispongan de la biblioteca <conio.h>.

Hemos visto que cuando usamos printf se escribe en la posición actual del cursor y se mueve éste al final de la cadena que hemos escrito. Pero ¿qué sucede cuando queremos escribir en una posición determinada de la pantalla? La solución está en la función gotoxy. Supongamos que queremos escribir 'Hola' en la fila 10, columna 20 de la pantalla:

```
#include <stdio.h> #include <conio.h>  main ( ) {       gotoxy  (20,  10);  printf ("Hola"); }
```

Obsérvese que primero se indica la columna (x) y luego la fila (y). La esquina superior izquierda es la posición (1, 1).

clrscr: borrado de la pantalla (DOS)

Ahora ya sólo resta saber cómo se limpia la pantalla. Ello es tan fácil como usar:

```
Clrscr (  )
```

(*clear screen*, borrar pantalla).

Esta función no sólo borra la pantalla, sino que sitúa el cursor en la posición (1, 1), en la esquina superior izquierda.

```
#include <stdio.h> #include <conio.h>  main ( ) {       clrscr (  );      printf
```

```
("Hola"); }
```

Este método sólo sirve para compiladores que incluyan el fichero conio.h.

6.21 ESTRUCTURAS DE CONTROL

Instrucciones de control de programa permiten alterar la secuencia normal de ejecución de un programa.

Estas instrucciones se dividen en tres grandes categorías:

- Instrucciones Condicionales que en c se implementan con las instrucciones if( ) y switch ( ).
- Instrucciones de ciclos con, for, while, do-while.
- Instrucción de salto incondicional goto.

INSTRUCCIONES CONDICIONALES

Una de las más poderosas características de cualquier computadora es la capacidad que tiene de tomar decisiones.

Es decir al comparar dos alternativas diferentes el computador puede tomar una decisión, basándose en la evaluación que hace de alguna condición.

Ejemplo de instrucciones condicionales;

---

a)

Si sueldo > 3000

desplegar "rico"

si no

desplegar "pobre"

fin-si

b)

si sexo = 'm'

imprime mujer

si no

imprime hombre

fin-si

De los ejemplos observar que los caminos por el computador dependerán de la evaluación que el computador hace con y de la condición.

Todo lenguaje de programación debe tener instrucciones que permitan formar condiciones e instrucciones que pueden evaluar esas condiciones.

El formato general de una instrucción condicional es:

Como se observa son cuatro partes bien diferenciadas entre sí;

La propia instrucción condicional en si
La condición
El grupo cierto de instrucciones
El grupo falso de instrucciones

Cuando el computador evalúa una condición, el resultado de esa evaluación solo es evaluado de dos maneras o la condición es CIERTA o la condición es FALSA.

Esto dependerá del valor que tenga asignado o que se haya capturado para la variable que está en la condición, por ejemplo si se capturo 6000 en sueldo en el ejemplo a), entonces el computador indicaría que la condición es CIERTA, pero en otro caso, si a la variable sueldo primero se le asignó un valor de 250 entonces el computador indicaría que la condición es FALSA.

Ya dependiendo del resultado de la evaluación, el computador ejecuta las instrucciones contenidas en la parte CIERTA o en la parte FALSA de la condición.

Empezaremos el análisis por la CONDICIÓN.

CONDICIONES SIMPLES

En general todas las condiciones simples se forman con:

Variables operadores relaciónales constante o var.

sexo = 'm'

sueldo > 300000

Una condición simple se define como el conjunto de variables y/o constantes unidas por los llamados operadores relaciónales.

Los operadores relacionales que reconoce el lenguaje C son:

Operador	**Significado**
==	Igual que
>	Mayor que
<	Menor que

>=	Mayor o igual que
<=	Menor o igual que
!=	No es igual que o es diferente que

Observar y tener cuidado sobre todo con el operador de igualdad (=), y el operador relacional de comparación por igualdad (==), es decir;

sueldo = 500, Se está pidiendo cargar o asignar la variable sueldo con el valor 500

sueldo == 500, se está pidiendo que se compare el valor o dato que se encuentra en la variable sueldo, contra el numero 500.

Solo este último formato es válido dentro de una condición en una instrucción condicional.

NOTA IMPORTANTE. Para el caso de objetos de tipo string, los operadores mencionados arriba funcionan, es decir es válido usar la siguiente condición:

string carrera;

if (carrera=="informática”) etc, etc, etc,

Pero para el caso de arreglos de caracteres, se tendrá que usar la función strcmp (str1,str2) que regresa cero si str1=str2, regresa un positivo si str1>str2 y regresa un negativo si str1<str2, el mismo ejemplo:

char carrera[20];

if (strcmp(carrera,"informatica")== 0) etc, etc, etc

Expresiones y operadores

Los distintos operadores permiten formar expresiones tanto aritméticas como lógicas. Los operadores aritméticos y lógicos son:

+, -	suma, resta
++, --	incremento, decremento
*, /, %	multiplicación, división, módulo
>>, <<	rotación de bits a la derecha, izquierda
&	AND booleano
|	OR booleano
^	EXOR booleano
~	complemento a 1
!	complemento a 2, NOT lógico
==, !=	igualdad, desigualdad
&&, ||	AND, OR lógico
<, <=	menor, menor o igual
>, >=	mayor, mayor o igual

En estos operadores deben tenerse en cuenta la precedencia de operadores y las reglas de asociatividad, que son las normales en la mayoría de lenguajes. Se debe consultar el manual de referencia para obtener una explicación de tallada. Además hay toda una serie de operadores aritméticos con asignación, como pueden ser += y ^=.

En la evaluación de expresiones lógicas, los compiladores normalmente utilizan técnicas de evaluación rápida. Para decidir si una expresión lógica es cierta o falsa muchas veces no es necesario evaluarla completamente. Por ejemplo una expresión formada <exp1> || <exp2>, el compilador evalúa primero <exp1> y si es cierta, no evalúa <exp2>. Por ello se deben evitar construcciones en las que se modifiquen valores de datos en la propia expresión, pues su comportamiento puede depender de la implementación del compilador o de la optimización

utilizada en una compilación o en otra. Estos son errores que se pueden cometer fácilmente en C ya que una asignación es también una expresión.

Debemos evitar:

```
if (( x++ > 3 ) || ( x < y ))
```

y escribir en su lugar:

```
x++;

if (( x > 3 ) || ( x < y ))
```

Hay un tipo especial de expresión en C que se denomina expresión condicional y está representada por los operadores? : . Su utilización es como sigue: <e>? <x>: <y>. Se evalúa si e entonces x; si no, y.

```
int mayor ( int a, int b )
{
        return ( a > b ) ? TRUE: FALSE;
}
waste_time ( )
{
        float a, b = 0.0;
        ( b > 0.0 ) ? sin(M_PI / 8) : cos (M_PI / 4);
}
```

Conversión de tipos

Cuando escribimos una expresión aritmética a+b, en la cual hay variables o valores de distintos tipos, el compilador realiza determinadas conversiones antes de que evalúe la expresión. Estas conversiones pueden ser para 'aumentar' o 'disminuir' la precisión del tipo al que se convierten los elementos de la expresión.

Un ejemplo claro, es la comparación de una variable de tipo int con una variable de tipo double. En este caso, la de tipo int es convertida a double para poder realizar la comparación.

Los tipos pequeños son convertidos de la forma siguiente: un tipo char se convierte a int, con el modificador signed si los caracteres son con signo, o unsigned si los caracteres son sin signo. Un unsigned char es convertido a int con los bits más altos puestos a cero. Un signed char es convertido a int con los bits más altos puestos a uno o cero, dependiendo del valor de la variable.

Para los tipos de mayor tamaño: si un operando es de tipo double, el otro es convertido a double. Si un operando es de tipo float, el otro es convertido a float. Si un operando es de tipo unsigned long, el otro es convertido a unsigned long. Si un operando es de tipo long, el otro es convertido a long. Si un operando es de tipo unsigned, el otro es convertido a unsigned. Si no, los operandos son de tipo int.

Una variable o expresión de un tipo se puede convertir explícitamente a otro tipo, anteponiéndole el tipo entre paréntesis.

```
cambio_tipo ( )

{
        float    a;
        int              b;
        b = 10;
        a = 0.5;
        if ( a <= (float) b )
                menor ( );
}
```

Control de flujo

La sentencia de control básica es if (<e>) then <s> else <t>. En ella se evalúa una expresión condicional y si se cumple, se ejecuta la sentencia s; si no, se ejecuta la sentencia t. La segunda parte de la condición, else <t>, es opcional.

```
int cero ( double a )
{
        if ( a == 0.0 )
                return (TRUE);
        else
                return (FALSE);
}
```

6.22 SENTENCIA IF

Hay dos formas en que se puede usar la sentencia if, la primera es donde solo evalúa una expresión y si es verdadera se ejecuta la sentencia (o sentencias) dependiendo de la forma en que lo utilice, a continuación presentamos su sintaxis:

If (expresión) *sentencia1*;

Se debe tener cuidado al utilizar la primera ya que si tengo 2 instrucciones en sentencias, solo se ejecutará una, sentencia1, pero si por el contrario utilizo la segunda forma se pueden tener más sentencias que se ejecuten dentro de la sentencia if como se muestra a continuación:

```
if (expresión) de la sentencia if es:{
        Sentencia1;
        Sentencia2;
```

```
        Sentencia3;
        .
        .
        .
        Sentencian
 }
```

Donde sentencia(n) puede ser una sentencia simple o un bloque, es decir un grupo de sentencias encerradas entre llaves.

La segunda forma de utilizar la sentencia if es utilizando la instrucción “else”, la cláusula else es opcional. La forma general del if sin bloques de sentencias es:

If (expresión) *sentencia*;

Else *sentencia;*

La forma del if con bloques de sentencias es:

```
If (expresión)
 {
        Sentencia de sentencias
 }
else
 {
         sentencia de sentencias
 }
```

si la expresión es cierta, se ejecuta la sentencia o el bloque de sentencias que constituye; en cualquier otro caso se ejecuta la sentencia o el bloque de

sentencias que constituye el objetivo del else. Recuerde que solo se ejecuta el código asignado al if o al else, nunca ambos.

Esta sentencia es equivalente a la que poseen la mayoría de lenguajes de programación y sirve para bifurcar en un punto de programa. Permite tomar decisiones al programa.

Tras evaluarse la expresión if y ejecutarse la sentencia adecuada, el programa continúa con la línea siguiente a la de la última sentencia del if. Para la sentencia if vale como expresión cualquiera válida en C, incluso las asignaciones y llamadas a funciones. El caso en que la expresión es una asignación suele ser sorprendente, ya que en la mayoría de los lenguajes este tipo de expresiones no es válido. Como sentencia vale cualquier tipo de sentencia válida en C, entre ellas la propia sentencia if. En este caso hablaremos de sentencias if anidadas. Por ejemplo:

```
if (num > 0)
        if (num == 1)
                puts ("num es igual a 1");
        else
                puts ("num es mayor que 1);
else
    puts ("num es menor que 1");
```

Cuando hay dos if anidados y a continuación hay un else, éste pertenece al último if. Así en el caso anterior el primer else corresponde al segundo if. Si queremos que un else pertenezca al primer if de un if anidado deberemos encerrar al segundo entre llaves. Por ejemplo:

```
if (num > 0)
 {
```

```
  if (num == 1)
     puts ("num es igual a 1");
 }
 else
puts ("num es menor que 0");
```

Cuando necesitamos ejecutar varias sentencias que dependen de un if, utilizaremos la sentencia de tipo bloque de sentencias. Un bloque de sentencias es un grupo de sentencias encerradas entre llaves { y }. Por ejemplo:

```
if (num >= 0)
{
  Printf ("num %d\n");
  if (num == 0)
     puts ("num 0");
     if (num >= 1)
        puts ("num mayor o igual a 1");
}
```

Ahora veamos el siguiente programa, que ejecuta una versión muy simple del juego “adivine el número mágico”. Imprime el mensaje “**Correcto**”cuando el jugador lo adivina.

Ejemplo 1

```
/*programa del número mágico.*/
main ( )
{
  int mágico= 123; /*número mágico*/
  int intento;
  printf(“adivine el número mágico.”);
```

```
  scanf (“%d”,&intento);
  if (intento==mágico)
    {
      printf (“**Correcto**”);
    }
return 0;
}
```

este programa usa el operador igualdad para determinar si el intento del jugador coincide con el número mágico. Si es así, imprime el mensaje en pantalla.

Ampliando el programa del número mágico, la siguiente versión muestra el uso de la sentencia else para presentar el mensaje de la respuesta a un número equivocado.

Ejemplo 2

```
/*programa del número mágico. mejora 1*/
main ( )
{
  int mágico= 123; /*número mágico*/
  int intento;
  printf (“adivine el número mágico.”);
  scanf (“%d”,&intento);
  if (intento==mágico)
    {
      printf (“**Correcto**”);
    }
  else
    {
      Printf (“.. Incorrecto ..”);
```

```
    }
  return 0;
}
```

Ejemplo 3

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
/* declaracion variables*/
int edad;
char ciudad[30];
/*capturando*/
Clrsc ( );
Gotoxy (10,5);printf ("dame edad : ");
Scanf ("%d",&edad); getchar ( );
Gotoxy (10,7); printf ("dame ciudad: ");
gets(ciudad);
/*comparando*/
If (edad>20)
  {
    Gotoxy (30,5);
    Puts ("mayor de 20");
  }
else
  {
     Gotoxy (30, 5);
     Puts ("menor de 20");
  }
```

```
If (strcmp (ciudad,"tijuana") ==0)
  {
     Gotoxy (35,7);
     Puts ("es de tijuana");
   };
Getchar ( );
}
```

Puede ser mejorado el programa del número mágico, proporcionando al jugador información sobre su equivocación. Esto se lleva a cabo mediante el uso del if anidado.

```
/*programa del número mágico. mejora 1*/
main ( )
{
int mágico= 123; /*número mágico*/
int intento;
printf (“adivine el número mágico.”);
scanf (“%d”,&intento);
if (intento==mágico) {
printf (“**Correcto**”);
printf (“%D es el número mágico”, mágico);
}
else {
printf (“.. Incorrecto ..”);
if (intento>mágico) printf (“demasiado alto”);
else printf (“demasiado bajo”);
return 0;
}
```

if es propiamente una sentencia y su mejor uso es precisamente como tal, pero también se pude usar como bucle o ciclo si se añade la sentencia **goto**.

```
etiqueta:

If (expresión) {
        Sentencia de sentencias
}
else {
sentencia de sentencias
}
goto etiqueta;
```

Esta no es la única posición que puede ocupar la sentencia goto. Por ejemplo:

```
# include <stdio.h>
Main ( )
{
long int a, i, x;
clrscr ( );
printf (“inserte un número”);
scanf (“%d”,&x);
a=1;
I=1;
Antonio:
If (i<=x) {
a=a*i;
++I;
goto Antonio;
}
Printf (el factorial de su número es %d”,a);
```

```
Getch ( );
}
```

este programa realiza el factorial de un número, cuando el programa llega a ejecutar el goto, esta etiqueta manda el control hasta la sentencia que sigue después del nombre de la etiqueta. El uso de la sentencia del goto no está restringida solo a este tipo de problemas, puede usarse como bandera y regresar el control a donde se requiera.

Realice los ejercicios del 5 al 9 del apartado EJERCICIOS.

6.23 SENTENCIA SWITCH

Ocurre con frecuencia, tanto en programación como en la vida real que las decisiones que se nos pide tomar son más complicadas que una simple selección entre dos alternativas.

Aunque la escalera ***if-then-else*** puede llevar a cabo pruebas múltiples, no se puede decir que resulte muy factible. El código puede ser muy difícil de seguir, y puede incluso confundir al propio autor. Por esto C posee una estructura de control que nos permite manejar esta situación de forma más práctica, que se denomina switch. En la sentencia switch se va comparando sucesivamente en una variable con una lista de constantes enteras o tipo carácter. Cuando se encuentra una coincidencia, se ejecuta la sentencia o secuencia asociada con esa constante. Las constantes no necesitan estar en ningún orden especial.

Antes de ver la forma de el switch veremos una sentencia muy importante utilizada en el switch, la cual se llama break, aunque esta sentencia casi siempre son usadas dentro de un switch, realmente son opcionales; estas se utilizan para concluir la secuencia de sentencias asociada a una constante, sin embargo si omitimos la sentencia break la ejecución continuara en el case siguiente hasta que

se llegue al final de la sentencia switch. La ejecución empieza en la etiqueta que coincida con el valor de la variable y continua hasta que se encuentra una sentencia break, o hasta que se termina la sentencia switch. Ahora veremos la forma más general de la sentencia switch:

```
Switch (variable) {

        case constante 1:
                secuencia de sentencias;
                break;

        case constante 2:
                secuencia de sentencias;
                break;

        case constante 3;
                secuencia de sentencias;
                break;
        .
        .
        .
        default:
                secuencia de sentencias
          }
```

en donde la sentencia **default** se ejecuta si no se encuentra ninguna coincidencia. El default es opcional y si no está presente, no se hace nada en caso de que no se produzca ninguna coincidencia. Cuando se encuentra una coincidencia, se ejecutan las sentencias asociadas con ese case hasta que llega a un break, o en el caso de default (o del ultimo case si no existe), hasta que se encuentra el final

de la sentencia switch. Como vemos esta sentencia es muy similar al de case en Pascal.

TRES COSAS IMPORTANTES ACERCA DE LA SENTENCIA SWITCH
1° Se diferencia de la sentencia if en que la sentencia switch sólo puede comprobar la igualdad, mientras que la expresión condicional del if puede ser de cualquier tipo.
2° No puede haber dos constantes de case en un mismo switch que tengan valores idénticos. Desde luego, una sentencia switch que este encerrada por un switch exterior puede tener constantes case que no sean las mismas
3° Una sentencia switch es más eficiente que una sentencia ***if-then-else***

ACERCA DE DEFAULT…

Añadiendo una sentencia default, es posible especificar secuencia de sentencias en el switch que se ejecutará si no se encuentran coincidencias.

El modo de funcionamiento de la sentencia Switch es el siguiente:

1. Primero se evalúa la expresión de control (en este caso variable).
2. A continuación se compara con la expresión de la primera etiqueta case. Si son iguales, se ejecuta la sentencia 1.
3. Luego se vuelve a comparar la expresión de control con la etiqueta del segundo case. De nuevo, si son iguales, se ejecuta la sentencia 2.
4. Se repite el proceso hasta agotar todas las etiquetas case. Si al llegar a la etiqueta default no se ha ejecutado ninguna otra sentencia ésta es la acción por defecto. La etiqueta default es opcional. Si no la ponemos el programa simplemente salta a la línea siguiente.

Hay que tener cuidado con un aspecto de este bucle. Cuando una expresión de una etiqueta case es igual a la sentencia de control, se ejecutan todas las sentencias que sigan hasta que se alcance una nueva etiqueta case, y luego se vuelve a comparar la expresión de control. Este mecanismo tiene una ventaja y un inconveniente. La ventaja es que no necesitamos encerrar entre llaves el grupo de sentencias a ejecutar para cada etiqueta. El inconveniente es que al agotar las sentencias de una determinada etiqueta la sentencia switch prosigue con la siguiente etiqueta case. Habitualmente lo que se pretende es que tras ejecutar el grupo de sentencias se finalice el switch. Para evitar el que se ejecuten más sentencias, habitualmente se acaba cada grupo de sentencias con una sentencia break. La sentencia break pasa entonces la ejecución a la siguiente línea de programa que prosiga al bucle switch.

También se permite poner etiquetas múltiples para un mismo grupo de sentencias. Si dejamos una etiqueta case sin sentencias a ejecutar entonces se asocia a la siguiente etiqueta. Esto es útil para ejecutar una misma acción para distintos valores de la expresión.

Vamos a ver un ejemplo de múltiples casos con if-else y luego con switch:

```
#include <stdio.h>
Main ( )
{
 int num;
 printf ("Introduce un número ");
 scanf ("%i", &num);
 if (num == 1)
   printf ("Es un 1\n");
 else if (num == 2)
        printf ("Es un 2\n");
```

```
      else if (num == 3)
           printf ("Es un 3\n");
  else
    printf ("No es ni 1, ni 2, ni 3\n");
}
```

Ahora con switch:

```
#include <stdio.h>
Main ( )
{
int num;
printf ("Introduce un número ");
scanf ("%i", &num);
switch (num)
 {
   case 1:
         printf ("Es un 1\n");
         break;
   case 2:
         printf ("Es un 2\n");
         break;
   case 3:
         printf ("Es un 3\n");
         break;

   default:
         printf ("No es ni 1, ni 2, ni 3\n");
 }
}
```

Como vemos, el código con switch es más cómodo de leer.

Vamos a ver qué pasa si nos olvidamos algún break:

```
#include <stdio.h>
Main ( )
{
  int num;
  printf ("Introduce un número");
  scanf ("%i", &num);
  switch (num) {
              case 1:
                    printf ("Es un 1\n");
/* Nos olvidamos el break que debería haber aquí */
              case 2:
                    printf ("Es un 2\n");
                    break;
              default:
                     printf ("No es ni 1, ni 2, ni 3\n");
}
}
```

Si al ejecutar el programa escribimos un 2 tenemos el mensaje Es un dos. Todo correcto. Pero si escribimos un 1 lo que obtenemos en pantalla es:

```
Es un 1 Es un 2
```

¿Por qué? Pues porque cada caso empieza con un case y acaba donde hay un break. Si no ponemos break aunque haya un case el programa sigue hacia adelante. Por eso se ejecuta el código del case 1 y del case 2.

También switch tiene algunas limitaciones; por ejemplo, no podemos usar condiciones en los case.

Ejemplos del Uso del Switch

```
/*programa  de cambio de base n 3 sentencia switch
decimal-->hexadecimal
*/
#include<stdio.h>
Main ( )
{
  int opcion;
  int valor;
  clrscr ( );
  printf ("transformar:\n");
  printf ("1: decimal en hexadecimal\n");
  printf ("2: hexadecimal en decimal\n");
  printf ("3: decimal en octal\n");
  printf ("4: octal en decimal\n");
  printf (" escriba su opcion:   \n");
  scanf ("%d", &opcion);
       switch (opcion) {
            case1:
                  printf ("introduzca un valor decimal: ");
              scanf ("%d", &valor);        printf ("%d en hexadecimal es %x",valor,valor);
              break;
              case 2:
              printf ("introduzca un valor hexadecimal:  ");
              scanf ("%x", &valor);
              printf ("%x en decimal es %d", valor, valor);
              break;
```

```
            case 3:
           printf ("introduzca un valor en decimal:  ");
          scanf ("%d", &valor);
           printf ("% d en octal es  %o", valor, valor);
           break;
            case 4:
            printf ("introduzca un valor en octal:  ");
             scanf ("%o",&valor);
           printf ("%o en decimal es %d", valor,valor);
             break;
          default:
                printf (“esa opción no es correcta. Inténtelo de nuevo\n”);
            }
           Getch ( );
      }
/*BREVE EJEMPLO DE DEFAULT*/
#include<stdio.h>
Main ( )
{
    int i;
      clrscr ( );
        for (i=1;i<5;i++)
      switch (i)
      {
      case 1:
            printf (" estoy en  case 1 \n");
            break;
      case 2:
            printf (" caso 2\n");
            break;
```

```
          case 3:
              printf ("caso 3 \n");
              break;
        default:
          printf (" por defecto\n");
          }
           }
        /*PROGRAMA QUE “ADIVINA” ESPACIO EN BLANCO O DIGITO.*/
#include<stdio.h>
#include<conio.h>
Main ( )
{
          char car;
        clrscr ( );
        printf (" INTRODUZCA UN CARACTER: \n");
        scanf ("%c",&car);
      switch (car)
          {
         Case '0':
         Case '1':
         Case ‘2’:
         Case ‘3’:
         Case '4':
         Case '5':
         Case '6':
        Case '7':
         Case '8':
         Case '9':
         Printf ("\n es un digito\n");
         break;
```

```
        case ' ':
       case '\t':
        case '\n':
              printf (" es un espacio en blanco.\n");
              break;
        default:
              printf (" Ninguno de los dos\n");
           break;
        }
        Getch ( );
         }

/*PROGRAMA CON SWITCH ANIDADOS “PROGRESION ARITMETICA*/
#include<stdio.h>
#include<conio.h>
main()
{ int opcion,n,t,primertr,termin,termino,dif,serie[30],i,a1,an,suma,medios
          Clrscr ( );
gotoxy (20,3); printf ("PROGRESION ARITMETICA\n")
 printf ("1: DEFINICION\n");
printf (" 2: CALCULAR EL TERMINO n DE UNA PROGRESION ARITMETICA\n");
printf (" 3: CALCULAR LA SUMA DE LOS PRIMEROS n TERMINOS\n");
 printf (" 4: INTERPOLAR n MEDIOS ARITMETICOS\n");
 gotoxy (20,10); printf (" ESCRIBA SU OPCION:   ");
   scanf ("%d",&opcion);
       switch (opcion) {
              case 1:
                 for (t=1;t<6;t++)
                     switch (t) {
case 1:
printf ("UNA PROGRESION ARIMETICA ES UNA SUCESION DE NUMEROS\n");
```

```
break;
case 2:
printf ("TAL QUE CADA UNO DE LOS TERMINOS POSTERIORES AL PRIMERO\n");
break;
case 3:
printf ("SE OBTIENE AÑADIENDO AL TERMINO ANTERIOR UN NUMERO FIJO\n");
break;
case 4:
printf ("LLAMADA LA DIFERENCIA DE LA PROGRESION ARITMETICA\n");
        printf ("Y SE PUEDE ESCRIBIR COMO:  \n");
        printf (" a1, a1+d, a1+2d,a1+3d,....,a1+(n-1)d\n");
        printf ("EN DONDE a1 ES EL PRIMER TERMINO Y d ES LA DIFERENCIA\n");
break;
            }
             break;
             case 2:
            printf("CUAL ES EL PRIMER TERMINO DE LA PROGRESION:  ");
             scanf ("%d",&primertr);
             printf ("CUAL ES LA DIFERENCIA:  ");
             scanf ("%d",&dif);
            printf ("CUAL ES EL n LUGAR TERMINO QUE DESEAS CALCULAR\n");
             scanf ("%d",&n);
                    termin= (n-1)*dif;
                          Termino= (primertr + termin);
                      Printf (" el termino  es %d\n “término, termino );
                      Printf (" LA PROGRESION ES: \n");
                      For (i=1; i<=n;i++){
                            Serie [i] = primertr+ (i-1)*dif;
```

```
            printf (" %d\t ", serie[i]);}
          break;
 case 3:
    printf ("HASTA CUAL TERMINO DESEAS SUMAR?\n");
     scanf ("%d",&n);
   printf ("DE EL VALOR DEL PRIMER TERMINO\n");
    scanf ("%d",&a1);
    printf ("CUAL ES LA DIFERENCIA?\n");
    scanf (" %d",&dif);
    an =a1+(n-1)*dif;
    scanf (" %d",&dif);
    an =a1+(n-1)*dif;
    suma =n/2*(a1+an);
    printf ("EL VALOR DE LA SUMA ES:  %d",suma,suma);
break;
case 4:
    printf (" DE EL PRIMER TERMINO:  \n");
    scanf ("%d",&a1);
    printf ("CUAL ES EL ULTIMO TERMINO:  \n");
     printf("CUANTOS MEDIOS ARITMETICOS DESEA
  INTERPOLAR ENTRE %d" ,a1,a1);
    printf (" y %d\n",an,an);
    scanf ("%d",&medios);
    n= (medios+2);
    dif=(an-a1)/(n-1);
    printf ("LOS MEDIOS ARITMETICOS ENTRE%d",a1,a1);
    printf (" y %d\n ",an,an);
          for (i=1;i<=medios;i++){
            a [i]= a1+(i-1)*dif;
           medioar[i] = (a[i]+dif);
           printf ("%d\t",medioar[i]);
```

```
                    }
            break;
             default:
printf ("¡ ESA OPCION NO ES CORRECTA VUELVA A     INTENTARLO !\n ");
             break;
             }
                 Getch ( );
                 }
```

Realice los ejercicios del 10 al 12 del apartado EJERCICIOS.

6.24 CICLO WHILE

Su fórmula general es:

While (condición) sentencia;

La condición puede ser cualquier expresión y cualquier valor. Cuando la condición se hace falsa, el control del programa pasa a la línea siguiente.

El bucle while comprueba la condición al principio, por lo que puede no ejecutarse.

Se necesitan varias condiciones para finalizar el bucle while, lo normal es utilizar una única variable para formar la expresión condicional.

No es necesario tener alguna sentencia en el cuerpo del bucle while, simplemente se puede repetir hasta que se pulse alguna tecla por ejemplo.

PROGRAMA CON EL CICLO WHILE:

/*Este programa me da los números del 1 al 10*/

```
#include<stdio.h>
#include<conio.h>
Main ( )
{
int x;
while (x<11)
{
Printf (“%d\n”,x);
x++
}
}
```

Ciclo do/while

A diferencia de los ciclos for y while, que analizan la condición del ciclo a principio del mismo, el ciclo de do /while analiza la condición al final del ciclo. Esto significa que el ciclo do/while siempre se ejecuta al menos una vez.

La forma general del ciclo do/while es

```
Do {
Secuencia de sentencias;
} while (condición);
```

aunque las claves no son necesarias cuando solo hay una sentencia se utilizan normalmente para evitar confusiones (al lector, no al compilador) con el while.

El siguiente ciclo do while lee números que el usuario introduce por el teclado hasta que encuentra un número menor o igual que cien.

```
#include<stadio.h>
#include<cotion.h>
Main ( )
{int num;
Do {
Scanf (“%d”,& num);
} while (num>100);
}
```

Quizás el uso más común del ciclo do while es para crear menús. Cuando el usuario introduce una respuesta válida, se devuelve como valor de la función. Una respuesta no valida hace que se muestre de nuevo el menú.

A continuación se presenta una versión mejorada del menú de comprobación ortográfica:

```
Void menú (void)
{
char c;
print (“1. comprobar ortografia\n”);
print (“2.corregir errores ortograficos\n”);
print (“3.mostrar errores ortograficos\n”);
print (“introduzca su opción:“);
do {
   c=getche ( );
        switch (c)  {
             case   ´1´ :
                comprobar ( );
                break;
              case´2´:
                corregir  ();
```

```
            break;
          case´3´:
            mostrar ( );
        }
    } while (c!=´1´  && c!=´2´  &&  c!=´3´);
}
```

Una función de menú se debe ejecutar siempre al menos una vez. Después de mostrar las opciones el programa intenta hasta que se selecciona una opción válida.

6.25 CICLO FOR

Probablemente estará familiarizado con la forma general del bucle for por encontrarlo en alguna forma en todos los lenguajes procedurales de programación. La forma general en C de la sentencia **for** es:

For (inicialización; condición; incremento) sentencia;

- La inicialización es una sentencia de asignación que el compilador usa para establecer la variable de control del bucle.
- La condición es una expresión que comprueba la variable de control del bucle cada vez para determinar cuándo salir del bucle.
- El incremento define la manera en que cambia la variable de control de bucle cada vez que la computadora repite el mismo.

Nota: se deben separar estas secciones mediante un punto y coma.

El bucle for continuará la ejecución del programa, mientras la condición sea verdad. Una vez que sea falsa la ejecución del programa continuara la sentencia que sigue al for.

El siguiente ejemplo imprime los números 1 a 100 en la pantalla:

```
# incluye < stdio.h >
   Main ( )
{
        int x;
        for (x=1; x<=100; x++) printf("%d ",x);
}
```

Explicación del programa: este programa inicializa **x** a 1, como x es menor que 100, el programa llama a **printf ( ),** el programa incrementa x en 1 y comprueba para ver si es todavía menor que 100. Este proceso se repite hasta que x es mayor que 100, en este momento el ciclo termina.

El bucle for no siempre necesita ir hacia adelante también puede decrementar, como lo muestra el siguiente ejemplo:

```
# include<stdio.h>
  Main ( )
{
  int x;
for ( x=100; x>0; x--)
printf ( " % d  " , x);
}
```

Sin embargo C no restringe a incrementar o decrementar el valor de la variable de control. En realidad se puede usar cualquier tipo de asignación que se quiera.

Por ejemplo el siguiente ejemplo imprime el número de 0 a 100, de 5 en 5.

```
# include< stdio.h>
  Main ( )
{
 int x;
 for ( x=0; x<=100; x= x+5) printf (“%d “,x);
}
```

Usando un bloque de código puede tener una sentencia for repitiendo sentencias múltiples como se muestra aquí. Este ejemplo 1 imprime el cuadra de los números 0 a 99, y el ejemplo 2 que presenta el código ascii.

```
   Ejemplo1
#include<stdio.h>
Main ( )
  {
       int i;
      clrscr ( );
       for (i=0; i<=99; i++) {
              printf ( " esto es i: %d", i);
              printf (" el cuadrado  de i: %d\n", i*i);
              getch ( );
                        }
}
 Ejemplo 2
#include <stdio.h>
Main ( )
 {   int x;
 Clrscr ( );
       For (x=14; x<=300; x++)
    Printf ("%d %c", x,x);
     getch ( );  }
```

6.26 PUNTERO

Los punteros son una de las más potentes características de C, pero a la vez uno de sus mayores peligros. Los punteros nos permites acceder directamente a cualquier parte de la memoria. Esto da a los programas C una gran potencia. Sin embargo son una fuente ilimitada de errores. Un error usando un puntero puede bloquear el sistema (si usamos MS-DOS o Windows, no en Linux), y a veces puede ser difícil detectarlo. Otros lenguajes no nos dejan usar punteros para evitar estos problemas, pero a la vez nos quitan parte del control que tenemos en C.

Direcciones de variables

Al declarar una variable estamos diciendo al ordenador que reserve una parte de la memoria RAM para almacenarla. Cada vez que ejecutemos el programa la variable se almacenará en un sitio diferente; eso no lo podemos controlar; depende de la memoria disponible y de otros varios factores. Puede que se almacene en el mismo sitio, pero es mejor no fiarse. Dependiendo del tipo de variable que declaremos, el ordenador reservará más o menos memoria. Como vimos en la sección Variables cada tipo de variable ocupa más o menos bytes. Por ejemplo, si declaramos un char, el ordenador reserva 1 byte (8 bits). Cuando finaliza el programa todo el espacio reservado queda libre.

Existe una forma de saber qué direcciones ha reservado el ordenador. Se trata de usar el operador **&** (operador de **dirección**). Vamos a ver un ejemplo: definimos la variable a y obtenemos su valor y dirección.

```
#include <stdio.h>
main()
{
 int a = 10;
```

```
  printf ("Dirección de a = %p, valor de a = %i\n", &a, a);
}
```

Para mostrar la dirección de la variable usamos %p en lugar de %i. Sirve para escribir direcciones de punteros y variables. El valor se muestra en formato hexadecimal.

No hay que confundir el valor de la variable con la dirección donde está almacenada. La variable a está almacenada en un lugar determinado de la memoria y ese lugar no cambia mientras se ejecuta el programa. El valor de la variable puede cambiar a lo largo del programa, lo cambiamos a voluntad mediante el código. Ese valor está almacenado en la dirección de la variable. El identificador (nombre) de la variable es equivalente a poner un nombre a una zona de la memoria. Cuando en el programa escribimos a, en realidad estamos diciendo, "el valor que está almacenado en la dirección de memoria a la que llamamos a".

¿Qué son los punteros?

Ahora ya estamos en condiciones de ver lo que es un puntero. Un puntero es una variable un tanto especial. Con un puntero podemos almacenar direcciones de memoria. En un puntero podemos tener guardada la dirección de una variable. Veamos la diferencia entre un variable puntero y las variables "normales".

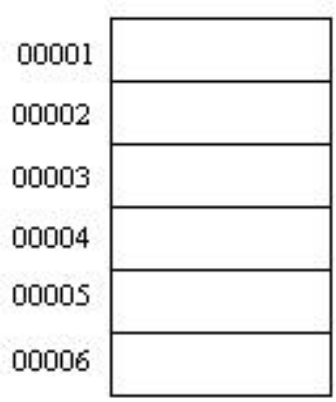
Un puntero es una variable que contiene una dirección de memoria.

Normalmente esta dirección es una posición de una variable en la memoria. Si una variable contiene la dirección de una variable, entonces se dice que la primera variable apunta a la segunda.

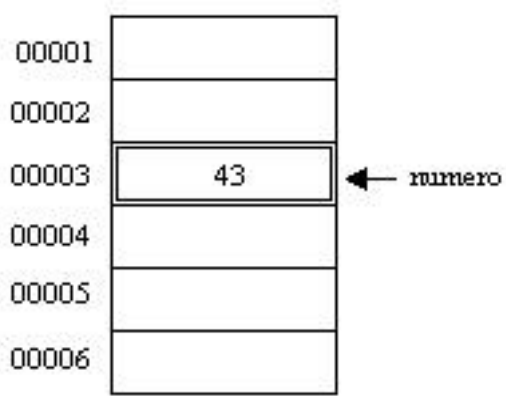
La declaración de un puntero consiste en un tipo base, el (*), y el nombre de la variable. El formato general para la declaración de una variable puntero es:

tipo *nombre_ de _variable

donde tipo puede ser cualquier tipo base en C (qué tipo de variables puede apuntar el puntero).



En el dibujo anterior tenemos una representación de lo que puede ser la memoria del ordenador. Cada casilla representa un byte de la memoria. Y cada número es su dirección de memoria. La primera casilla es la posición 00001. La segunda casilla es la posición 00002, y así sucesivamente. Supongamos que ahora declaramos una variable: char numero = 43. El ordenador guardaría, por ejemplo, esta variable en la posición 00003. Esta posición de la memoria queda reservada y ya no la puede usar nadie más. Además, esta posición a partir de ahora se denomina número. Como le hemos asignado el valor 43, el valor 43 se almacena en la posición de memoria 00003.



Si ahora usáramos el programa siguiente:

```
#include <stdio.h>
Main ( )
{
 int numero = 43;
 printf ("Dirección de numero = %p, valor de numero = %i\n", &numero, numero);
}
```

el resultado sería:

```
Dirección de numero = 00003, valor de numero = 43
```

Hemos dicho que un puntero sirve para almacenar las direcciones de memoria. Muchas veces los punteros se usan para guardar las direcciones de variables. Como cada tipo de variable ocupaba un espacio distinto, cuando declaramos un puntero debemos especificar el tipo de datos cuya dirección almacenará. En el próximo ejemplo queremos utilizar un puntero que almacene la dirección de una variable int. Así que para declararlo debemos hacer:

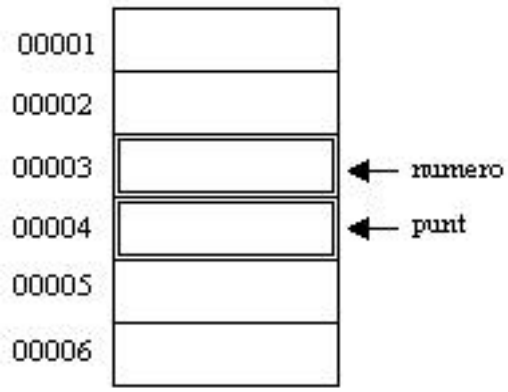
```
int *punt;
```

El * (asterisco) sirve para indicar que se trata de un puntero y debe ir justo antes del nombre de la variable, sin espacios. En la variable punt sólo se pueden guardar direcciones de memoria, no se pueden guardar datos. Vamos a volver sobre el ejemplo anterior un poco ampliado para ver cómo funciona un puntero:

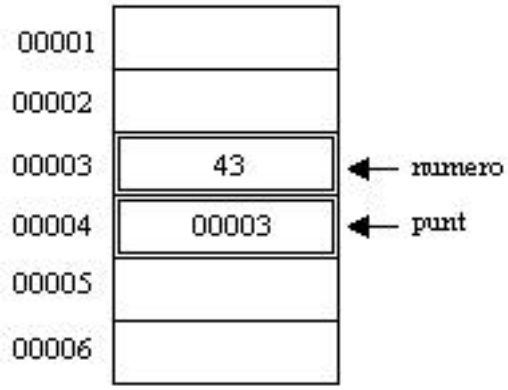
```
#include <stdio.h>
Main ( )
{
 int numero;
 int *punt;
 numero = 43;
 punt = &numero;
 printf ("Dirección de numero = %p, valor de numero = %i\n",  punt, numero);
}
```

Vamos a analizar línea a línea:

- En la primera, int numero; reservamos memoria para número (supongamos que queda como antes, en la posición 00003). Por ahora número no tiene ningún valor.
- Siguiente línea: int *punt; reservamos una posición de memoria para almacenar el puntero. Lo normal es que a medida que se declaran variables se guarden en posiciones contiguas. De modo que quedaría en la posición 00004. Por ahora punt no tiene ningún valor, es decir, no apunta a ninguna variable. Esto es lo que tenemos por ahora:



- Tercera línea: numero = 43; Aquí estamos dando el valor 43 a número. Se almacena 43 en la dirección 00003, que es la de número.
- Cuarta línea: punt = &numero. Por fin damos un valor a punt. El valor que le damos es la dirección de número (ya hemos visto que & devuelve la dirección de una variable). Así que punt tendrá como valor la dirección de número, 00003. Por lo tanto tenemos:



Cuando un puntero tiene la dirección de una variable se dice que ese puntero **apunta** a esa variable. La declaración de un puntero depende del tipo de dato al que queramos apuntar. En general, la declaración es:

*tipo_de_dato *nombre_del_puntero;*

Ahora veamos un programa que usa dos sentencias de asignación dando a imprimir el número 100 en la pantalla:

```
 #include <stdio.h>
Main ( )
{
int *con_dir, cont, val,
cont = 100
cont_dir = &cont;   /* coge la dirección del contador */
val = *cont_dir;    /* coge el valor de esta dirección */
printf ("&d",val);   /* visualiza 100 */
}
```

El & es un operador monetario que devuelve la dirección de la memoria de su operando.

El * es un operador monoario que devuelve el valor de la variable situada en la dirección que sigue.

En C se pueden usar solo dos operaciones aritméticas sobre punteros: + y - (cada vez que la computadora incrementa un puntero, apuntará a la posición de memoria del elemento siguiente en función de su tipo de base).

También se pueden sumar o restar enteros a punteros.

La expresión p = p + 9; hará que p apunte al noveno elemento del tipo de base de p después del que apuntaba actualmente (no se pueden realizar otras operaciones sobre punteros).

```
#include <stdio.h>
Main ( )
{
int x;
```

```
Int  *p1, *p2;
x = 101;
p1 = &x;
p2 = p1;

 /* imprime el valor hexadecimal de la dirección de x - -
no el valor de x */

printf ("la dirección de: %p ", p2);

 /* ahora imprime el valor de x */

printf ("el valor de x es %d\n", *p2);
}
```

¿Para qué sirve un puntero y cómo se usa?

Los punteros tienen muchas utilidades; por ejemplo, nos permiten pasar argumentos (o parámetros) a una función y modificarlos. También permiten el manejo de cadenas y de arrays. Otro uso importante es que nos permiten acceder directamente a la pantalla, al teclado y a todos los componentes del ordenador. Si sólo sirvieran para almacenar direcciones de memoria no serían de mucha utilidad. Nos deben dejar también la posibilidad de acceder al contenido de esas posiciones de memoria. Para ello se usa el operador ***** (operador de **dirección**), que no hay que confundir con el de la multiplicación.

```
#include <stdio.h>
main()
{
int numero;
int *punt;       numero = 43;          punt = &numero;
printf( "Dirección de numero = %p, valor de numero = %i\n",  punt, *punt );
}
```

Si nos fijamos en lo que ha cambiado con respecto al ejemplo anterior, vemos que para acceder al valor de numero usamos *punt. Esto es así porque punt apunta a número y *punt nos permite acceder al valor al que apunta punt.

```
#include <stdio.h>
main()
{
int numero;
int *punt;       numero = 43;
punt = &numero;
*punt = 30;
Printf ( "Dirección de numero = %p, valor de numero = %i\n",          &numero,
numero);
}
```

Ahora hemos cambiado el valor de número a través de *punt. El resultado sería:

```
Dirección de numero = 00003, valor de numero = 30
```

En resumen, usando punt podemos apuntar a una variable y con *punt vemos o cambiamos el contenido de esa variable.

Un puntero no sólo sirve para apuntar a una variable, también sirve para apuntar una dirección de memoria determinada. Esto tiene muchas aplicaciones; por ejemplo nos permite controlar el hardware directamente (en MS-DOS y Windows, no en Linux). Podemos escribir directamente sobre la memoria de video y así escribir directamente en la pantalla sin usar printf.

Uso de punteros en comparaciones

Veamos el siguiente ejemplo. Queremos comprobar si dos variables son iguales usando punteros:

```
#include <stdio.h>
main ( )
{
int a, b;
int *punt1, *punt2;
a = 5; b = 5;
punt1 = &a; punt2 = &b;
if ( punt1 == punt2 )
printf ("Son iguales\n");
}
```

Alguien podría pensar que el if se cumple y se mostraría el mensaje Son iguales en pantalla. Pero no es así, el programa es erróneo. Es cierto que a y b son iguales. También es cierto que punt 1 apunta a a y punt 2 a b. Lo que queríamos comprobar era si a y b son iguales. Sin embargo, con la condición estamos comprobando si punt 1 apunta al mismo sitio que punt 2, estamos comparando las direcciones donde apuntan. Por supuesto a y b están en distinto sitio en la memoria, así que la condición es falsa. Para que el programa funcione deberíamos usar los asteriscos:

```
#include <stdio.h>
Main ( ) {
int a, b;
int *punt1, *punt2;
a = 5; b = 5;
punt1 = &a; punt2 = &b;
if (*punt1 == *punt2)
```

```
printf ("Son iguales\n");
}
```

Ahora sí. Estamos comparando el contenido de las variables a las que apuntan punt 1 y punt 2. Debemos tener mucho cuidado con esto porque es un error muy frecuente.

Vamos a cambiar un poco el ejemplo. Ahora b no existe y punt 1 y punt 2 apuntan a a. La condición se cumplirá porque apuntan al mismo sitio.

```
#include <stdio.h>
Main ( )
{
int a;
int *punt1, *punt2;
a = 5;
punt1 = &a; punt2 = &a;
if (*punt1 == *punt2)
printf ("punt1 y punt2 apuntan al mismo sitio\n");
}
```

Punteros como argumentos de funciones

Hemos visto en la sección Funciones cómo pasar parámetros y cómo obtener resultados de las funciones (con los valores devueltos mediante return). Pero tiene un inconveniente, sólo podemos devolver un único valor. Ahora vamos a ver cómo los punteros nos permiten modificar varias variables en una función.

Hasta ahora para pasar una variable a una función hacíamos lo siguiente:

```
#include <stdio.h>
int suma (int a, int b)
{
return a+b;
}
Main ( ) {
int var1, var2;
var1 = 5; var2 = 8;
printf ("La suma es : %i\n", suma(var1, var2) );
}
```

Aquí hemos pasado a la función los parámetros a y b (que no podemos modificar) y nos devuelve la suma de ambos. Supongamos ahora que queremos tener la suma pero además queremos que var1 se haga cero dentro de la función. Haríamos lo siguiente:

```
#include <stdio.h>
int suma ( int *a, int b )
{
int c;
c = *a + b;
*a = 0;
return c; }
main ( ) {      int var1, var2;
var1 = 5; var2 = 8;
printf ("La suma es: %i y a vale: %i\n", suma(&var1, var2), var1 );
}
```

Fijémonos en lo que ha cambiado (con letra en negrita): En la función suma hemos declarado a como puntero. En la llamada a la función (dentro de main) hemos puesto & para pasar la dirección de la variable var1. Ya sólo queda hacer

cero a var1 a través de *a=0. También usamos una variable c que nos servirá para almacenar la suma a+b.

Es importante no olvidar el operador & en la llamada a la función ya que sin él no estaríamos pasando la dirección de la variable sino cualquier otra cosa.

FUNCIONES

Las funciones son los bloques constructores de C y el lugar donde se da toda la actividad del programa la forma general de una función es:

Especificador_de_tipo nombre_de_la_función (lista de parámetros)
{
cuerpo de la función
}

ESTRUCTURA BÁSICA DE UN PROGRAMA CON FUNCION.

```
#include <stdio.h>
int cuad (int x)       /*declaración de tipo de la función*/       (1)
main (void)
{
int t=10;
clrscr ( );
printf (“%d %d”,cuad(t),t);        /*uso de la función cuad*/          (2)
getch ( );                          /*detiene la pantalla hasta que le de un valor*/
return 0;                          /*fin del programa cuando se usa en main ( ) */
}
cuad(int x)                                                          }
{                                                                    }
x=x*x;                                                               } cuerpo de la función.   (4)
```

```
return (x);     /*valor que devuelve la función */   (3)
}
```

(1) El *Especificador_de_tipo* especifica el tipo de valor que devuelve la sentencia **return,** si no se especifica el tipo, el compilador asume que la función devuelve un resultado entero. La lista de parámetros es la lista de parámetros es la lista de variables separados por comas con sus tipos asociados que reciben los valores de los argumentos cuando se llama la función. Una función puede no tener parámetros en cuyo caso la lista de parámetros contiene solo la palabra clave **void.**

(2) al aparecer la función en el programa puede que ocurran dos cosas con real aplicación:

a) el valor que devuelve la sentencia return sea asignado a una variable usada dentro del cuerpo del programa.
b) El valor devuelto por la sentencia return sea usado dentro de otra función dentro del programa como en este ejemplo donde fue empleada dentro de la función printf.

```
#include <stedio.h>
int mul (int a, int b);
main (void)
{
int x, y, z;
x=10;
y=10;
z=mul (x,y);                          /*1*/
printf ("%d %d",mul (x, y),z);          /*2*/
mul (x,y) ;                           /*3*/
return 0;
```

```
}
Mul (int a, int b)
{
return a*b;
}
```

En la línea 1 está representado el inciso a, en la línea dos es el b y en la línea tres se observa que la función es una sentencia más del programa, en este último caso el valor devuelto por la sentencia return se perderá.

(3) Existen dos maneras de finalizar una función, la primera ocurre cuando se ha ejecutado la última sentencia de la función y, conceptualmente se encuentra } del final de la función. Una vez realizada la última sentencia la función no tiene nada que más que hacer y regresa el control al lugar que la llamó. Esto es parecido a un procedimiento de Pascal.

El segundo es el empleo de la sentencia return (como es el caso de nuestro ejemplo), en el que se quiere devolver un valor. Una función puede tener más de una sentencia return.

(4) la forma en que opera la función es el siguiente:

al momento de que el programa llega a la ejecución de una función, el control es asignado al cuerpo de la función llevando consigo los valores de las variables establecidas en ella y reasignándoselas a sus parámetros, el programa ejecutará las sentencias de la función y se devolverá el control a la parte de donde fue llamada la función. En el caso de que la última sentencia sea return, como se mencionó en el párrafo anterior, no solo se regresa el control sino que además se asigna el valor que devuelve la funciona, dicho valor es el argumento de la sentencia return del cuerpo de la función.

Una función nunca será objeto de una asignación de valor, es decir la expresión

Intercambia (x, y) =100

Es incorrecta y el programa marcara esta expresión como errónea.

Si una función es declarada como **void** no puede ser utilizada en ninguna expresión.

Realice los ejercicios del 5 al 9 del apartado EJERCICIOS.

6.27 FUNCIONES

6.27.1 FUNCIONES QUE DEVUELVEN VALORES NO ENTEROS

Cuando no está declarada explícitamente el tipo de una función, automáticamente por omisión el tipo es int. Sin embargo, cuando es necesario devolver un tipo de dato diferente, el proceso requiere dos pasos.

1. hay que dar a la función un especificado de tipo explícito.
2. el tipo de la función debe estar identificado antes de hacer la primera llamada.

Esta es la única forma en que turbo C puede generar código correcto para funciones que devuelven valores enteros.

Ejemplo:

- Dados dos números a (número entero) y b (número natural mayor o igual que cero) determinar a^b.
int potencia (int a, int b)

```
{
  if (b == 0) return 1;
  else return a * potencia (a, b-1);
}
```

La condición de parada se cumple cuando el exponente es cero. Por ejemplo, la evaluación de potencia (-2, 3) es:

potencia (-2, 3) ->
(-2) · potencia (-2, 2) ->
(-2) · (-2) · potencia (-2, 1) ->
(-2) · (-2) · (-2) · potencia (-2, 0) ->
(-2) · (-2) · (-2) · 1

y a la vuelta de la recursión se tiene:

(-2) · (-2) · ***(-2) · 1 /=/*** (-2) · (-2) · ***(-2) · potencia (-2,0)***
< (-2) · ***(-2) · (-2) /=/*** (-2) · ***(-2) · potencia (-2, 1)***
< ***(-2) · 4 /=/ (-2) · potencia (-2,2)***
< ***-8 /=/ potencia (-2,3)***

EJEMPLO

- Dado un array constituido de números enteros y que contiene N elementos siendo N >= 1, devolver el elemento mayor.

```
int mayor(int numeros[], int posicion)
{
  int aux;
  if (posicion == 0)
     return numeros[posicion];
  else {
```

```
    aux = mayor (numeros, posicion-1);
    if (numeros[posicion] > aux)
      return numerous [posicion];
    else
      return aux;
  }
}
...
int números [5] = {2,4,1,-3,-1};
int N = 5;
printf ("%d\n", mayor(numeros, 4));
```

USO DE PROTOTIPOS DE FUNCIONES

Un prototipo de una función tiene dos cometidos especiales. Primero, identifica el tipo devuelto en una función, de forma que turbo C pueda generar el código correcto para la devolución de datos. Segundo, especifica el tipo y el número de argumentos que utiliza la función.

La forma general de un prototipo es:

Tipo nombre_de_la_funcion (lista de parámetros)

Ejemplo:

PROTOTIPOS DE LA BIBLIOTECA ESTANDAR

Para hacerlo dentro del entorno integrado, seleccione el menú **options** (opciones). Dentro de options, seleccione **compiler** (compilador) y después **Messages** (mensage). En este menu, seleccione **frequent errors** (errores frecuentes). Finalmente active la opcion **callto funtion witthout prototipe** (llamada a función

sin prototipo). Si utiliza la función de línea de órdenes del compilador use la obcion **–wpro**

ARCHIVOS DE CABECERA DE TURBO C:

NOMBRE DEL ARCHIVO	
Alloc.h	
Assert.h	
Bios.h	
Conio.h	
Ctype.h	
Dir.h	
Dos.h	
Ermo.h	
Fcntl.h	
Flota.h	
Graphics.h	
Io.h	
Limits.h	
Locale.h	
Math.h	
Mem.h	
Process.h	
Setjmp.h	
Share.h	
Signal.h	
Stdargs.h	
Stddef.h	
Stdio.h	

6.28 RECURSIVIDAD

Se dice que algo es recursivo si se define en función de sí mismo o a sí mismo. También se dice que nunca se debe incluir la misma palabra en la definición de ésta. El caso es que las definiciones recursivas aparecen con frecuencia en matemáticas, e incluso en la vida real. Un ejemplo: basta con apuntar una cámara al monitor que muestra la imagen que muestra esa cámara. El efecto es verdaderamente curioso, en especial cuando se mueve la cámara alrededor del monitor.

En matemáticas, tenemos múltiples definiciones recursivas:

- Números naturales:

(1) 1 es número natural.
(2) el siguiente número de un número natural es un número natural.

- El factorial: n!, de un número natural (incluido el 0):

(1) si n = 0 entonces: 0! = 1
(2) si n > 0 entonces: n! = n · (n-1)!

Asimismo, puede definirse un programa en términos recursivos, como una serie de pasos básicos, o **paso base** (también conocido como condición de parada), y un **paso recursivo**, donde vuelve a llamarse al programa. En un computador, esta serie de pasos recursivos debe ser finita, terminando con un paso base. Es decir, a cada paso recursivo se reduce el número de pasos que hay que dar para terminar, llegando un momento en el que no se verifica la condición de paso a la recursividad. Ni el paso base ni el paso recursivo son necesariamente únicos.

Por otra parte, la recursividad también puede ser indirecta, si tenemos un procedimiento P que llama a otro Q y éste a su vez llama a P. También en estos casos debe haber una condición de parada.

6.28.1 FUNCIONES RECURSIVAS

Se denominan funciones recursivas a aquellas que se llaman a sí mismas.

En lógica matemática y computación, las **funciones recursivas** o también conocidas como **funciones recursivas-μ** son una clase de funciones de los números naturales en los números naturales que son «computables» en un sentido intuitivo. De hecho, en teoría de la computabilidad se demuestra que las funciones recursivas son precisamente las funciones que pueden ser calculadas con el formalismo de cómputo más general conocido como lo son las máquinas de Turing. Las funciones recursivas están relacionadas con las funciones primitivas recursivas y su definición inductiva se construye en base a la de las funciones primitivas recursivas. No toda función recursiva es primitiva recursiva. El ejemplo más conocido es la función de Ackermann.

Existen otros sistemas formales equivalentes en cuanto a poder de expresión, por ejemplo el Cálculo Lambda y las cadenas de Markov.

La Informática se interesa por las clases de problemas que puedan resolverse mediante reglas y las clases de máquinas que son equivalentes a esas reglas. Para resolver un problema, aquí el procedimiento consiste en reducir el problema a otro cuya solución se pueda alcanzar mediante la evaluación de funciones de enteros no negativos.

Es claro entonces que debemos encontrar transformaciones que nos pasen de funciones cualesquiera, asociadas al problema general, a funciones sobre enteros no negativos.

Funciones Recursivas son aquellas funciones de N^n N, de n-uplas (x_1, x_2, x_3, ..., x_n) de enteros no negativos en enteros no negativos, calculables por algoritmo.

El ejemplo clásico de una función recursiva es el cálculo de la factorial de un número. Por ejemplo, 4! (factorial) es igual a 4 X 3 X 2 X 1. Algebraicamente, podemos considerar un factorial como (n!) as n X (n-1) X (n-2) X (n-3)... (n - (n+1)). Ésta sería la definición recursiva de la factorial:

```
(defun factorial (n)
  (cond          ((zerop n) 1)
                 (T
    (* n (factorial (1- n))))
  ); _ fin de cond
); _ fin de defun
```

Se analizan dos condiciones:

1. que el argumento recibido sea cero (predicado ZEROP). En ese caso se devuelve 1.
2. en cualquier otro caso, se multiplica el argumento por el resultado que devuelve aplicar de nuevo la función de usuario factorial al argumento menos 1. Lógicamente la evaluación de esta operación deberá esperar al resultado de la nueva llamada a la función factorial, que si aún no es cero, provocará una nueva llamada a la misma función factorial y quedará a la espera de que esta sea evaluada y así sucesivamente. Al llegar a cero el argumento pasado a factorial, ya no se producirá una nueva llamada a la función y se comenzará a devolver respuestas a todas las evaluaciones que han quedado pendientes.

La recursión, aunque aporta soluciones de gran elegancia, debe ser utilizada con cautela. Lo ideal sería usarla con funciones que no utilicen, o utilicen muy pocos

argumentos, ya que la recursión coloca cada vez una nueva copia de la función en la memoria de pila junto con sus argumentos. Es muy posible que una función efectúe tantas recursiones que se agote la memoria disponible provocando un error de desbordamiento de la pila.

Empezaremos por distinguir un método iterativo frente a un método recursivo mediante el ejemplo del cálculo de la factorial de un número *n*.

$n!=n\cdot(n-1)\cdot(n-2)...2\cdot1$

El resultado de la factorial de un número entero es un número entero mucho más grande (**long**), que puede sobrepasar el rango de los enteros (**int**) si *n* es grande.

- **Método iterativo**

```
long factorial (int n){
    long resultado=1;
    for (int i=1; i<=n; i++){
            resultado*=i;
    }
    return resultado;}
```

- **Método recursivo**

```
long factorial (int n){
    if (n==0) return 1;
    return (n*factorial(n-1));
}
```

6.28.2 RECURSIÓN PRIMITIVA

Para aumentar el conjunto de las funciones calculables es necesario introducir la operación recursión primitiva.

Con esta operación se define una nueva función a partir de dos funciones dadas. Si la función h (x) a definir es de un sólo argumento, hemos de conocer un valor de base, por ejemplo en x = 0, lo que define el caso base h (0), y el procedimiento para calcular h (x + 1) a partir de h (x):

Datos: h (0) = k y g (x, y) con y = h (x), siendo k una función constante, se define h(x), para todo x ≥0, de la siguiente manera:

$$\begin{cases} h(0) = k \\ h(x+1) = g(x, h(x)) \end{cases}$$

Ejemplo:

h (x) = 2x .

Observamos: h (0) = 2* 0 = 0;

H (x +1) = 2 (x + 1) = 2x + 2.

En este caso: h (0) = N (x) = 0. Y para definir g (x, y) nos basamos en que

h (x + 1) = h (x) + 2, o sea:

h (x +1) = S (S (I^2_2 (x, h(x)))) = h (x) + 2.

En efecto:

h (0) = 0

h (1) = S (S (I22 (0, 0))) = S (S (0)) = 2

h (2) = S (S (I_2^2(1, h(1)))) = S (S (I_2^2(1, 2))) = S (S (2)) = 4

h (3) = S (S(I^2_2(2, h (2)))) = S (S (I^2_2(1, 4))) = S (S (4)) = 6

.....

h (n+1) = S (S (I^2_2 (n, h (n)))) = S (S (I^2_2 (n, 2n))) = S (S (2n)) = 2n + 2

Generalizando ahora la operación recursión primitiva para n argumentos:
Datos: f (x_1, x_2, ..., x_n) (función total de n argumentos) y g (y, z, x_1, x_2, ..., x_n) (función total de n + 2 argumentos).

A partir de estas funciones se define la función h (y, x_1, x_2, ..., x_n) de n + 1 argumentos mediante la operación recursión primitiva:

$$h(0, x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n)$$
$$h(y+1, x_1, x_2, ..., x_n) = g(y, h(y, x_1, ..., x_n), x_1, ..., x_n))$$

A y se le denomina **variable de recursión** y a x_1, x_2, ..., x_n **parámetros de la recursión** primitiva.

La recursión primitiva es una forma recurrente de generar funciones, porque el valor de h para y +1 viene dado por la composición g de y, de h (y) y de los parámetros.

O sea, a partir de una función base f y de la función que realiza la inducción g, conocido el valor de h (y), podemos obtener h (y + 1). Puesto que las funciones f y g son totales, la función h está definida para todos los puntos de la variable de recursión y todas las n-uplas de los argumentos. Por tanto, h: $N^{n+1} \rightarrow N$ es una función total.

6.28.3 FUNCIONES PRIMITIVAS RECURSIVAS

El conjunto de N (x), S (x), I^n_i (x_1,..., x_n) y todas las funciones que se pueden obtener aplicando las operaciones de composición y recursión primitiva es cerrado y se denomina conjunto de **funciones primitivas recursivas**.

Gödel, en 1931, demostró que toda función primitiva recursiva es total. En efecto, tanto la composición como la recursión primitiva, sobre funciones totales, calculan funciones también totales.

Para conseguir una nueva función primitiva recursiva, generada por la inducción (composición y recursión primitiva), ésta podrá contener las funciones de base N (x), S (x), I^n_i (x_1,..., x_n) y cualquier función que hayamos demostrado que es primitiva recursiva, como, por ejemplo, f (x) = 2.x.

Ejemplos:

La **suma** es una función primitiva recursiva de dos argumentos. En efecto,

h (x, y) = S (x, y) = x + y. Si tomamos y como variable de recursión:

{S(0, x) = I^1_1(x) Observamos que: x + 0 = x; x + (y +1) = (x + y)+1.

{ S (y +1, x) = S (I^3_2(y, S (x, y), x) = S (S (x, y)) = S (x, y) +1.

Es análogo a decir que: x + 0 = x

Y que: (y + 1) + x = (x + y) +1

En este caso f (x) = I^1_1 (x)

g (y, z, x) = S (I^3_2(y, z, x)), con z = h (x, y) = S (x, y).

El **producto** es una función primitiva recursiva de dos argumentos.

Tomando y como parámetro de la recursión, observamos que:

x* 0 = 0; (y+1)* x = y* x + x.

Denotamos: h (x, y) = M (x, y) = x* y. La recursión primitiva toma la forma:

{ h (0, x) = N (x)

{ h (y +1, x) = (SI^3_2 (y, h(x, y), x), I^3_3(y, h(x, y), x))

Siendo f (x) = N (X); g (y, z, x) = S (I^3_2 (y, z, x), I^3_3 (y, z, x)), con z = h (x, y) = M (x, y).

Observamos la necesidad de que la función g tenga una variable más que h.

La función de un argumento **predecesor** es primitiva recursiva.

Se define como: $P(x) = 0$ si $x = 0$ y $P(x) = x-1$ si $x \geq 1$.

Esta función puede ser obtenida mediante recursión primitiva (total):

$$P(0) = N(x)$$

$$P(x+1) = I^2_1(x, P(x))$$

En efecto es fácilmente comprobable:

x	0	1	2	3	4	5	.....
P(x)	0	0	1	2	3	4	.....

La **diferencia propia** (o resta natural) es una función primitiva recursiva de dos argumentos.

Definiéndola como:

$$D(x, y) = x - y \quad \text{si } x \geq y$$

$$D(x, y) = 0 \quad \text{si } x < y$$

Podemos obtener esta función total por recursión primitiva:

$$D(x, 0) = I^1_1(x)$$

$$D(x, y+1) = P(I^3_2(x, D(x, y), y))$$

En efecto, es comprobable:

x	2	2	2	2
y	0	1	2	3
D(x, y)	2	1	0	0

Esta función D(x, y) se suele notar también como $x \dot{-} y$.

La función **valor absoluto** es primitiva recursiva de dos argumentos:

Se define como:

$$|x - y| = x - y \quad \text{si } x > y$$
$$|x - y| = 0 \quad \text{si } x = y$$
$$|x - y| = y - x \quad \text{si } x < y$$

Podemos obtener esta función por recursión primitiva (total). Pero es más sencillo obtenerla como composición de la operación suma y diferencia propia:

$$|x - y| = (x, y) + (y, x) = D(x, y) + D(y, x) = S(D(x, y), D(y, x))$$

Todos estos ejemplos corresponden a funciones recursivas primitivas muy usadas. Se recurrirá a ellas frecuentemente para construir otras más complejas.

6.28.4 FUNCIONES TOTALES NO PRIMITIVAS RECURSIVAS

La clase de funciones primitivas recursivas es muy extensa. Pero, ¿puede identificarse con la clase de funciones calculables que sean totales? NO. Es difícil construir una función total calculable que no sea a la vez primitiva recursiva; pero existen funciones con esas características, como las funciones de Ackermann.

Función de Ackermann

Es una función total, doblemente recursiva, calculable, pero no es una función primitiva recursiva. Esta función es un caso especial de una clase de funciones múltiplemente recurrentes definidas por Ackermann, que demostró que no eran primitivas recursivas.

El hecho de que es calculable significa que existe un algoritmo que nos proporciona su solución; o sea que existe una Máquina de Turing que la calcula.

La función más simple de Ackermann tiene dos argumentos y está definida por las tres ecuaciones:

f (0, n) = n + 1 (1)
f (m, 0) = f (m -1, 1) (2)
f (m, n) = f (m-1, f (m, n -1)) (3) El procedimiento recursivo que la calcula es: (en PASCAL):

```
function f (m, n: integer): integer;
    begin
        if m = 0  then f: = n+1
            else if  n = 0 then f: = f (m-1,1)
                else f: = f (m-1, f (m, n-1))
    end;
```

El algoritmo es simple; luego si una computadora puede calcularla para valores enteros no negativos prácticos, es seguro que existe la Máquina de Turing que calcula la función de Ackermann. Luego esta función de Ackermann es calculable y total.

Las operaciones (1), (2) y (3) definen un procedimiento que generaliza la operación recursión primitiva. Ésta, como hemos visto, define un proceso iterativo lineal: el valor de la función en el paso m depende del valor en el paso m -1:

h (0) = k;
h (m) = g (m -1, h (m -1)).

En cambio, (1), (2) y (3) permiten definir un procedimiento análogo en dos dimensiones:

- La ecuación (1) define el valor de f (x, y) en todo punto del eje y: f (0, y) = y + 1.
Así, f (0,0) = 1; f (0, 1) = 2; f (0, 2) = 3; f (0, 3) = 4;..

- La ecuación (2) define el valor de f (x, y) en todo punto del eje x a partir de valores previamente hallados: f (x, 0) = f (x -1, 1).Ahora bien si al intentar aplicar (2) se requiere conocer el valor de la función cuando ninguna de las dos variables es cero, hay que echar mano de (3).

Así, f (1, 0) = f (0, 1) = 2

f (2, 0) = f (1, 1) valor que se calcula a partir de (3):

Entonces como f (1, 1) = f (0, f (1, 0)) = f (0, 2) = 3;

f (3, 0) = f (2, 1) - valor que se calcula a partir de (3): f (2, 1) = f (1, f (2, 0)) = f (1, 3).

De nuevo este valor hay que calcularlo a partir de (3): f (1, 3) = f (0, f (1, 2)),
con f (1, 2) = f (0, f(1, 1)) = f (0, 3) = 4. Así pues:

f (3, 0) = f (2, 1) = f (1, 3) = f (0, f (1, 2)) = f (0, f(0, 3)) = f (0, 4) = 5

Por tanto, la evaluación de los puntos del eje de abscisas no es tan simple como los del eje de ordenadas. Los valores de la función de Ackermann en los puntos del tipo (x, 0), para x = 3 dependen de los valores de la función en los puntos (2, 1), (1, 3), (1, 2), (0, 3) y (0, 4). Luego el valor en un punto no depende del valor en un punto vecino del plano (x, y), sino de todo un enjambre de valores que parten del eje y.

La ecuación (3), como acabamos de ver, define el valor de f (x, y) en todo punto del semiplano {x > 0, y > 0} cuando se conoce previamente el valor en el punto (x, y-1) y podemos calcular f (x, y) para cualquier punto de la recta x = m-1. Por ello antes de conseguir la evaluación de un f (x, y) cualquiera debemos calcular todos los anteriores.

Así, por ejemplo para calcular el valor de la función de Ackermann en (2, 2) debemos conocer su valor en (2, 1) (y todos los que nos permitan calcular éste) y, además, su valor en (1, f (2, 1)) (o sea, en toda la recta x = 1 desde y = 0 hasta y = f (2, 1)).

En efecto: f (2, 2) = f (1, f (2, 1)).

Calculamos primero f (2, 1) = f (1, f (2, 0)) = f (1, f (1, 1)) =
f (1, f (0, f (1, 0))) = f (1, f(0, f (0, 1))) = f (1, f (0, 2)) = f(1, 3) =
f (0, f (1, 2)) = f (0, f (0, f(1, 1))) = f (0, f (0, f (0, f (1, 0)))) =
f (0, f (0, f (0, f (0, 1)))) = f (0, f (0, f (0, 2))) = f (0, f (0,3)) =
f (0, 4) = 5.

Luego, f (2, 2) = f (1, 5) = f (0, f (1, 4)) = f (0, f (0, f (1, 3)))

Pero f (1, 3) = 5 según el desarrollo anterior.

Por tanto, f (2, 2) = f (0, f (0, 5)) = f (0, 6) = 7

Hemos visto entonces que para encontrar f (2, 2) = 7 tenemos que calcular la función de Ackermann en los puntos:

f (0, 6) = 7
f (0, 5) = 6 f (1, 5) = 7
f (0, 4) = 5 f (1, 4) = 6

f (0, 3) = 4	f (1, 3) = 5	
f (0, 2) = 3	f (1, 2) = 4	f (2, 2) = 7
f (0, 1) = 2	f (1, 1) = 3	f (2, 1) = 5
f (0, 0) = 1	f (1, 0) = 2	f (2, 0) = 3

La función de Ackermann crece explosivamente, más que las funciones exponenciales analíticas. Hemos visto que f (2, 2) = 7. Se ve fácilmente que:

$$f(1, n) = f(0, f(1, n-1)) = f(1, n-1) + 1 = n + 2$$

Se puede demostrar que: $(n + 3)^{\cdot^{\cdot^{\cdot^{2}}}}$ ($n + 3$ veces) $- 3$

$$f(2, n) = 2n + 3; \quad f(3, n) = 2^{n+3} - 3; \quad f(4,n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n\ \text{veces}} - 3$$

El cálculo de f (4, 4) es inabordable:

Nota: $f(4, 4) > 10^{19199}$

Número de partículas del Universo (Teoría "Big-Bang") = 10^{80}

Es sorprendente que un programa como éste, en el que la única operación de modificación de la información es un único bloque en el que se incrementa en una unidad el valor de la segunda variable, para valores tan pequeños de los argumentos (m, n) = (4, 4), pueda calcular un valor tan grande.

Conclusión: **Existen funciones**, como éstas, que son **totales y calculables** y, sin embargo, **no son funciones primitivas recursivas**, ya que no se pueden obtener por composición y recursión primitiva a partir de N (x), S (x) y I^n_i (x_1, x_2,..., x_n). Se

necesita alguna operación más sobre las funciones recursivas para poder obtener todas las funciones calculables totales.

6.28.5 FUNCIONES RECURSIVAS Y PARCIALMENTE RECURSIVAS

Además de completar las funciones totales no primitivas recursivas, también se necesita obtener funciones parciales. Para ello se necesita incluir nuevas operaciones. El conjunto de las operaciones se completa con la minimalización.

OPERACIÓN MINIMALIZACIÓN

Dada una función total f ($y, x_1, x_2, ..., x_n$) de n + 1 variables, la operación **minimalización** genera una nueva función de n variables h ($x_1, x_2, ...x_n$) de la siguiente forma: Para una n-upla ($x_1, x_2, ..., x_n$) el valor de la función h es el mínimo valor entero no negativo de y para el cual f ($y, x_1, x_2, ..., x_n$) = 0. Se denota así:

$$h(x_1, x_2, ..., x_n) = \min_{y \in N} (f(y, x_1, x_2, ..., x_n) = 0)$$

Obviamente la función h ($x_1, x_2,..., x_n$) no tiene que ser necesariamente total. En efecto:

- Si para cualquier n-upla ($x_1, x_2,..., x_n$) existe al menos un valor de y para el que se cumple f ($y, x_1, x_2,..., x_n$) = 0, entonces h será total.

- Si hay n-uplas ($x_1, x_2,..., x_n$) para las que no existe un valor de y que satisfaga f ($y, x_1, x_2,..., x_n$) = 0, entonces h será parcial. La función está indefinida en cada n- upla para la que no hay solución de la ecuación.

- Puede también darse el caso de que, para ninguna n-upla de enteros no negativos, exista un valor de y que satisfaga la ecuación. No existe ninguna n-upla en la que la función esté definida en este caso. Entonces la función h está indefinida.

Observación: En el caso de la recursión primitiva, el número de iteraciones es el valor de la variable de recursión. En el caso de la minimalización, el número de veces que se calcula f y se compara con 0 es impredecible (el resultado de calcular el valor de la función).

Ejemplo de función total obtenida por minimalización es la función **identidad** de un argumento: h (x) = x.

Se puede obtener a partir de la **diferencia propia**:

En efecto hay infinitos valores de y que cumplen x ≤ y, o sea x ·_ y = 0. Pero el mínimo es y = x. Por tanto la función también se puede expresar como min (x ·_ y = 0).

yÎN

Se define también a partir de la función **valor absoluto** de 2 argumentos.

La función f (y, x) = |x - y| es total. Aplicando la operación minimalización, se obtiene: h (x) = min (|x - y| = 0) = x

yÎN

que es también total, puesto que es primitiva recursiva, como ya vimos.

Observamos la siguiente **propiedad general sobre la igualdad**:

max (a £b) = min (b £a), siendo b **un punto fijo** y

a a

pudiendo tomar a todos los valores entero no negativos. Eso es igual que decir a = b. Ha de cumplirse, por tanto,

$(a \dot{-} b = 0) \wedge (b \dot{-} a) = 0$.

O bien, lo que es igual: $(a \dot{-} b) + (b \dot{-} a) = 0$. O sea, $| a - b | = 0$. Poniéndolo en la forma que exige la operación minimalización queda así: $\min_a (| a - b | = 0)$.

Cuando hay valores que una variable no puede tomar, porque la función a definir está indefinida en ellos, no se puede cumplir $a = b$, de manera que $\min_a (| a - b | = 0)$ dará cuenta tanto del valor de la función cuando está definida como de que no haya ningún valor que cumpla la ecuación cuando la función está indefinida. Lo aplicamos inmediatamente.

Ejemplo de función parcial obtenida por minimalización: $h(x) = x/3$:

x	0	1	2	3	4	5	6	--------
h (x)	0	?	?	1	?	?	2	--------

Aplicamos la operación minimalización. Se ha de encontrar una función f (y, x) total que dé cuenta del cumplimiento de las condiciones que exige la ecuación de la operación.

Fijémonos en que, si sólo se cumpliera la condición $(3\ y \leq x)$, sería insuficiente, puesto que existen muchos valores de y que la cumplen, desde y = 0. Además el máximo valor de y que la cumple sólo cumple $3y = x$ si, y sólo si, x es divisible por 3. Análogamente ocurre para la condición $x \leq 3y$. Hay infinitos valores de y que cumplen esa condición, pero, si x no es múltiplo de 3, ninguno de ellos cumple $x = 3y$. Se han de cumplir simultáneamente las condiciones $(3\ y \leq x)$ y $(x \leq 3y)$, porque esas dos condiciones se satisfacen simultáneamente sólo para valores de x múltiplos de 3, incluido cero, que es precisamente lo que queremos. Además si x no es un múltiplo de 3, no hay ningún valor de y que pueda cumplir las dos condiciones

a la vez. Esto equivale a decir que se ha de cumplir ((3y $\cdot$ x = 0) $\wedge$ (x $\cdot$ 3y = 0)) o, lo que es igual: (3y $\cdot$ x) + (x $\cdot$ 3y) = 0. O sea la función buscada es $f(y, x) = |3y - x|$.

Para cada valor de x, cuando hay solución, existe un sólo valor de y que cumpla la ecuación, pero h (x) es una función parcial. Dom (h) = {xÎ N | x = 3 n; n Î N}.
h (x) está definida sólo en los valores enteros no negativos múltiplos de 3.

$$h(x) = \min_{y \in N} (|3y - x| = 0)$$

Ejemplo de función indefinida:

Es muy sencillo llegar a funciones indefinidas. Veamos un ejemplo con una variable. Partimos con una función total de dos variables f (y, x) = x + y + 1. Al minimalizar, se obtiene: h (x) = min (x + y + 1 = 0), que es una función totalmente indefinida.

y ÎN

Ahora tenemos ya la oración base y la oración inductiva (composición, recursión primitiva y minimalización) para obtener el conjunto de las funciones recursivas.

Nos falta, entonces, llegar a la oración extremal o cierre del conjunto definido inductivamente por la propiedad calculabilidad: “Toda función parcialmente calculable es recursiva”. Es ésta la Tesis de Church.

Esto nos conduce a las siguientes definiciones:

- Toda función que se obtenga a partir de N (x), S (x) e $I^n_i(x_1, x_2, ..., x_n)$ mediante las operaciones composición y recursión primitiva es total (Gödel) y se denomina función **primitiva recursiva**. Las funciones primitivas recursivas pueden obtenerse también empleando la minimalización, aunque no es necesario.

- Una función total cuya minimalización conduce a una función también total se llama función **regular**. P.e. Las funciones de Ackermann son regulares. Son totales y no primitivas recursivas.

- Una función que pueda obtenerse a partir de N (x), S (x) e $I^n_i (x_1, x_2,..., x_n)$ mediante las operaciones composición, recursión primitiva y minimalización se denomina función **parcialmente recursiva.**

- Si, además de parcialmente recursiva, la función obtenida es total, se denomina **totalmente recursiva,** tanto si hay que emplear la minimalización (funciones regulares) como si no (funciones primitivas recursivas).

A veces es más fácil obtener funciones primitivas recursivas utilizando la minimalización.

Ejemplos de funciones totales importantes obtenidas por minimalización:

Las funciones **techo** (división entera por exceso) de x/2 y **suelo** (división entera por defecto) de x/2 son totales.

Son funciones de N en N definidas de la siguiente forma:

x	0	1	2	3	4	5	6	7	8
x/2	0	1	1	2	2	3	3	4	4
x/2	0	0	1	1	2	2	3	3	4

La función techo de x/2 se puede definir como $\lceil x/2 \rceil = \min_{y \in N} (y \geq x/2)$.

Teniendo en cuenta que $a \dot{-} b \Leftrightarrow b \dot{-} a = 0$, entonces $2y \geq x \Leftrightarrow x \dot{-} 2y = 0$. Por consiguiente, es $\lceil x/2 \rceil = \min (x \dot{-} 2y = 0) = \min (D(x, 2y) = 0)$.

La función suelo de x/2 se puede definir como $x/2 = \max (y \leq x/2) = \max (2.y \in x)$

Deseamos transformarla a la forma min (f (y, x) = 0).

El problema es encontrar la f (y, x) adecuada.

Se pueden dar 2 casos:

1.- Si x es par, el mayor valor de y que cumple $2\ y \leq x$ es el valor de y que cumple 2 y = x. Este valor de y es único.

2.- Si x es impar, el mayor valor de y que cumple $2\ y \leq x$ es el valor de y que cumple

2 y + 1 = x. También es único este valor de y.

Si sumamos 1 a y, ¿qué ocurre en la desigualdad?:

1- Si x es par: 2 (y +1) = 2 y + 2; 2 (y +1) = x + 2.

2- Si x es impar: 2 (y +1) = (2 y +1) +1 = x +1; 2 (y +1) = x +1.

O sea: 2 (y +1) > x, al menos, en 1 unidad.

Cuando x es impar, es 2 (y + 1) > x en 1 unidad y cuando x es par, es 2 (y +1) > x en 2 unidades. Por tanto:

$$\max_{y} (2\ y \pounds x) = \min_{y} (2\ (y+1) > x)$$

Deseamos ahora transformar ">" en "≥" para adecuar la inecuación a la definición de la operación minimalización.

Si sumamos 1 unidad en x:

1-caso x par 2 (y + 1) = (x + 1) +1

2-caso 1 impar 2 (y + 1) = x + 1

Luego en cualquier caso es $2\ (y + 1) \geq x + 1$.

O sea: $\min(2(y + 1) > x) = \min (2(y + 1) \geq x + 1)$.

y y

Así pues, la función que se necesita para la ecuación es f (y, x) = x ¸ 2 y ¸ 1. Y la función pedida es

$$x/2 = \min_{y \in N} (x \dot{-} 2y \dot{-} 1 = 0)$$

Luego las funciones techo y suelo se pueden obtener por minimalización, pero son primitivas recursivas. En general **las funciones primitivas recursivas se pueden obtener también por minimalización.** El teorema de Davis tiene en cuenta las limitaciones de esta aseveración.

P. ej. $f(x) = x^2$

Teniendo en cuenta que

$$(x + 1)^2 = x^2 + 2x + 1,$$

y que 2x es una función primitiva recursiva, se puede definir f(x) así:

Por recursión primitiva: f (0) = N (x); y apoyándonos en la función 2x,
$f(x+1) = S(I^2_1(f(x), x), I^2_1(2x, x), I^2_1(S(N(x)), x))$.

Por composición: $f(x) = S(I^1_1(x), I^1_1(x))$.
Por minimalización:
Empecemos observando que $f(x) = m\,a\,x\,(y / x \geq x)$

y

En efecto todos los valores de y tales que $y \pounds x^2$ cumplen $y / x \geq x$. El mayor valor de y que lo cumple es $y = x^2$. Independientemente del valor de x, siempre existe un valor de y que cumple $y / x = x$, puesto que el cuadrado de un número entero es un número entero. Este valor es $y = x^2$.

Esta definición de f (x) se apoya en la función cociente entero (y / x), definida como suelo o como techo (da igual puesto que el cociente es entero), y tal que para x = 0 es y / 0 = 0 para cualquier valor de y. Obsérvese que el menor valor de y que cumple y / 0 = 0 es y = 0.

También se cumple que es f (x) = min (x £ y / x)

y

Todos los valores de y tales que x^2 £ y cumplen x £ y/x. El menor valor de y que lo cumple es y = x^2. Por tanto, la función pedida es:

f (x) = min (x· y / x = 0),

y que es igual que decir que f (x) es el valor de y que cumple x = y / x.

No vale definir f (x) = min (y / x · x = 0), ya que hay muchos valores de y (y = 0, 1,...., x^2) que cumplen y / x £ x, de forma que el mínimo no satisface y / x = x en general. Se podría definir como max (y / x £ x), que define un valor único de y. O también como el valor único de y que satisface (min (y / x £ x = 0)) ∧ (y / x = x).

Englobando todo, si queremos que en la minimalización se cumpla y / x£ x, ha de exigirse que se cumpla simultáneamente x ≥ y / x. Todo ello se expresa definiendo la función como:

f (x)= min ((x≥ y/x) + (y/x£ x) = 0)

y

FUNCIÓN RECURSIVA FINAL

Función recursiva con un caso recursivo formado exclusivamente por una llamada a la función, sin que el resultado de ésta se combine (sea modificado) posteriormente. En este caso la función de combinación *c* es la identidad.

Su forma es la siguiente:

$\{Q(\overline{x})\}$
fun $f\ (\overline{x} : T_1)$ **dev** $(\overline{y} : T_2) \equiv$
    caso $B_t(\overline{x}) \rightarrow triv(\overline{x})$
      [] $B_{nt}(\overline{x}) \rightarrow f(s(\overline{x}))$
    fcaso
ffun
$\{R(\overline{x}, \overline{y})\}$

FUNCIÓN RECURSIVA NO FINAL

Función recursiva con un caso recursivo formado por una expresión que incluye una llamada recursiva a la función. El resultado de invocar recursivamente a la función se combina de alguna forma con las variables de entrada.

La función de combinación *c* es distinta de la identidad.

El esquema general es:

$\{Q(\overline{x})\}$
fun $f\ (\overline{x} : T_1)$ **dev** $(\overline{y} : T_2) \equiv$
    caso $B_t(\overline{x}) \rightarrow triv(\overline{x})$
      [] $B_{nt}(\overline{x}) \rightarrow c(f(s(\overline{x})), \overline{x})$
    fcaso
ffun
$\{R(\overline{x}, \overline{y})\}$

FUNCIÓN RECURSIVA MÚLTIPLE

Es una función recursiva que tiene en la expresión del caso recursivo más de una llamada a sí misma.

FUNCIÓN RECURSIVA LINEAL

También se denomina función recursiva simple .Es una función recursiva que tiene en la expresión del caso recursivo una única llamada a la función, esto es, tal que cada llamada externa produce sólo una llamada interna. Su forma es la siguiente:

$\{Q(\overline{x})\}$
fun $f\ (\overline{x} : T_1)$ **dev** $(\overline{y} : T_2) \equiv$
    caso $B_t(\overline{x}) \rightarrow triv(\overline{x})$
      [] $B_{nt}(\overline{x}) \rightarrow c(f(s(\overline{x})), \overline{x})$
    fcaso
ffun
$\{R(\overline{x}, \overline{y})\}$

CASO RECURSIVO

Es la opción de la alternativa que define a la función recursiva cuando los datos son tales que no admiten un tratamiento directo, y por tanto requieren para su cálculo ulteriores llamadas recursivas. Si la protección del caso recursivo está abierta (si se alcanza el caso recursivo) se devolverá el resultado de evaluar la expresión asociada a dicha protección y que incluye (al menos) una llamada recursiva (y una posible combinación del resultado de ésta con datos de entrada de la función).

EXPRESIÓN CASO RECURSIVO

La expresión general para un caso recursivo viene dada por la expresión $c(\overline{x}, f(s(\overline{x})))$, donde:

- La función sucesor es la tupla que se le pasa como argumento a la llamada recursiva de la función.

- La parte de los datos de entrada que se combinan con la llamada recursiva a la función son los formados por la expresión.
- La llamada se combina con la expresión anterior mediante una o más operaciones, que son las que vienen descritas como función de combinación.

6.28.6 DEFINICIÓN DE FUNCIONES PARCIALES

Lo fundamental de las funciones parcialmente recursivas es que existe un algoritmo para calcularlas aplicando en su dominio el proceso de minimalización sobre funciones primitivas recursivas (totales). En los puntos para los que la función está indefinida, la ecuación involucrada en la minimalización no tiene solución.

Ejemplos de funciones parciales:

Una función importante es la **resta natural**. La deduciremos por generalización a partir de valores particulares del segundo parámetro.

La ecuación y + 1. x = 0 nos da la solución.

En efecto, para x = 0 la ecuación es y + 1 = 0. No hay solución para y Î N.En ese punto f (x) está indefinida. Fijémonos en que para x = 1 es y = 0, como debe ser. Pero para x = 2, son solución los valores y = 0 y también y = 1. En general, para x = k, son solución.

y = 0, y = 1,..., y = k -1. O sea, f (x) = max (y + 1. x = 0).

El máximo valor de y para el cual es y + 1 £ x es el mínimo valor de y para el cual es

y + 1≥ x. O sea:

max (y + 1 . x = 0) = min (x . (y +1) = 0)

y y

Pero min (x. (y + 1) = 0) no está indefinido para x = 0, ya que se cumple 0·_ + 1) = 0 para cualquier valor de y. El valor mínimo para el que se cumple es y = 0. Por tanto es

f (0) = min (0 ·_(y + 1) = 0) = 0. y

Para que la función esté indefinida en x = 0 ha de exigirse que se cumpla x = y + 1 (sin solución para x = 0), además de estar definida en todos los demás puntos de dominio por f (x) = min (x. (y + 1) = 0).

Eso equivale a decir que se cumpla ((y + 1£ x) A (x£ y+1)),lo que se expresa así:

f (x) = min (((y + 1) ·_ x) + (x £ (y + 1)) = 0)

Podemos generalizar el algoritmo igualmente:

Función **restados** (x) = max (y + 2£ x).

Análogamente a la función resta uno, también la condición es ahora

(min (x . (y + 2) = 0) x = y + 2)).

y

La función pedida es: restados (x) = min ((y + 2. x) + (x. (y + 2) = 0)

Y, generalizando, la función **resta** de dos argumentos es:

Resta (x, y) = max (z + y. x = 0)

z

Se ha de cumplir:

La función resta (x, y) está indefinida para x = 0, 1,..., y-1.

Para x = y, es z = 0, resta (x, y) = 0.

Para x > y es z = x – y.

Todo ello se expresa así:

resta (x, y) = min(((z + y) $\dot{-}$ x) + (x $\dot{-}$ (z + y)) = 0)
z

6.28.7 TEOREMA DE DAVIS

La clase de las funciones parcialmente recursivas (calculables) puede ser generada mediante composición y minimalización (omitiendo la operación recursión primitiva) si extendemos la lista de funciones básicas, con lo cual quedaría:

(1)	N (x)	cero
(2)	S (x)	sucesor
(3)	I^n_i (x_1,..., xn)	identidad
(4)	S (x, y) = x + y	suma
(5)	D (x, y) = x $\dot{-}$ y	diferencia propia
(6)	M (x, y)= x* y	multiplicación.

6.28.8 PRECISIONES SOBRE LOS CONCEPTOS DE RECURSIVIDAD E ITERATIVIDAD

El término **recursivo** se usa para:

(1) Definir un objeto en términos de él mismo. O sea, que lo definido entra en la propia definición. Ejemplos frecuentes son las clases sintácticas definidas por la formalización de Backus-Naur (BNF) o diagramas sintácticos. De la misma forma se dice que una subrutina es recursiva cuando existe dentro del cuerpo de la subrutina una llamada a sí misma.

(2) Dar una definición recursiva de una correspondencia, aplicación, función, transformación u operación mediante una relación de recurrencia, sea ésta lineal

(como todo lo que abarca la operación recursión primitiva) o multilineal (como las funciones de Ackermann).

(3) Hablar acerca de funciones recursivas como funciones totales dentro de la clase de las funciones parcialmente recursivas (calculables). Pero también se habla de funciones recursivas en general, cometiendo un abuso de lenguaje, para referirse a las funciones parcialmente recursivas.

El término **iterativo** se usa para hablar de procesos en los que se produce una serie de repeticiones (finita o indeterminadamente grande) para calcular una función en cualquier punto de su dominio a partir de un valor inicial de la función, mediante una relación de recurrencia. Es bastante más que un ciclo repetitivo. La iteración tiene una potencia algorítmica mucho mayor que la operación recursión primitiva.

6.28.9 TESIS DE CHURCH

En virtud de su generación la clase de las funciones parcialmente recursivas es inductiva; es decir, cualquier función parcialmente recursiva está definida recursivamente o por inducción, que es lo mismo. La definición de la operación minimalización es, de esta forma, el algoritmo mismo de cálculo de la función.

La clase de las funciones parcialmente calculables es muy amplia y hay buenas razones para pensar que puede ser identificada con la clase de funciones efectivamente calculables (después veremos que función recursiva y máquina de Turing son equivalentes). Nadie hasta el momento ha sido capaz de definir una función efectivamente calculable que no sea parcialmente recursiva. La expresión formal de esta idea es la Tesis de Church: " La clase de las funciones efectivamente calculables y la clase de las funciones parcialmente recursivas son equivalentes".

Después de lo visto hasta ahora es claro que toda función parcialmente recursiva es potencial o efectivamente calculable. En efecto, toda función obtenida a partir de

N (x), S (x) e I_i^n (x_1, x_2,..., x_n) aplicando las operaciones composición (sobre su dominio), recursión primitiva y minimalización es "programable"; es decir, existe un algoritmo que permite obtener la función (al menos teóricamente). O sea, es potencialmente calculable ya que existen un conjunto de mecanismos automatizables (la programación de computadoras) que resuelve el problema. Existe, asociado con este problema, o con esa función, la máquina de Turing que lo calcula.

La equivalencia inversa: "**toda función potencialmente calculable es parcialmente recursiva**" es realmente la Tesis de Church, basada en que nadie ha podido definir una función potencial o efectivamente calculable que no pueda ser calculada por una Máquina de Turing o que no sea parcialmente recursiva.

De acuerdo a la Tesis de Church, el conjunto inductivo de la propiedad calculabilidad (conjunto de las funciones parcialmente recursivas), formado por la oración base: “N (x), S (x) e I^n_i (x_1,..., xn) son funciones primitivas recursivas” (por tanto, parcialmente recursivas) y la oración inductiva (las reglas de operación de composición, recursión primitiva y minimalización), es **consistente** y **completo.** O, lo que es igual, la propiedad calculabilidad de funciones recursivas (parcialmente recursivas) es decidible.

EJEMPLOS DESGLOSADOS DE FUNCIONES RECURSIVAS

Números de Fibonacci

En matemáticas, los números de Fibonacci, forman una secuencia recursiva dada por:

$$F_n := F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

El siguiente código calcula los *n* primeros números de Fibonacci.

```
/*                    fibonacci.c  */

#include <stdio.h>
#include <conio.h>
void main(){
int f [300], n, k, i;
clrscr ( );
printf ("Escribe cuantos números de fibonacci quieres:");
scanf ("%d", &n);
for (i=0; i<=n; i++) {
        if (!i) {
                f [i] = 0;
        } else if (i==1) {
                f [i] = 1;
        } else {
                f [i] = f[i-1] + f[i-2];
        }
        printf ("%d", f[i]); }
printf ("¿Cuál número de Fibonacci quieres? ");
scanf ("%d", k);
printf ("El número f(%d) es: ", f[k]); }
```

Función McCarthy 91

En matemáticas discretas, la función McCarthy 91 es una función recursiva la cual regresa 91 para todos los argumentos enteros $n \leq 100$ y regresa $n - 10$ para $n > 100$. Fue concebida por el científico computologo John McCarthy.

La función McCarthy 91 se define como

$$M(n) = \begin{cases} n - 10, & \text{if } n > 100 \\ M(M(n + 11)), & \text{if } n \leq 100 \end{cases}$$

El siguiente código muestra la función McCarthy 91

/* mccarthy.c */

```
#include <stdio.h>
#include <conio.h>
void main ( ) {
int m [300], n, k, i; // Declaro 'm' como un vector de números de McCarthy, igual se
pueden imprimir
clrscr ( );        // sin declarar ningún vector.
printf ("Escribe el valor que quieres calcular: ");
scanf ("%d", &n);
for (i=0; i<=n; i++) { //Pongo 'int i' porque tmb ahí se pueden declarar variables
        if (i <= 100) {          // Pongo '!i' porque es lo mismo que poner 'i==0'
                m [i] = n - 10;
        } else {
                m [i] = m[m[i + 11]];
        }
        Printf ("%d: %d", i, m[i]);             // Esto imprime todos los números
// con sus índices
}
//      Tmb puedes pedir un índice y obtener ese número de McCarthy
printf ("¿Cuál número de McCarthy quieres?");
scanf ("%d", k);printf("El número m(%d) es: ", m[k]); }
```

PROBLEMAS SUGERIDOS

1. En un estante se quieren acomodar 7 libros de matemáticas, 4 de química y 6 de física. De cuantas maneras diferentes se pueden acomodar si los libros de la misma área siempre deben estar juntos.

2. En un sorteo donde se juegan 50 números, se va a premiar al 1º, 2º y 3º. De cuantas maneras diferentes se pueden premiar.

3. De cuantas maneras tres niñas y dos niños se pueden sentar en una fila si los niños se sientan juntos y las niñas también.

4. Existen 10 puntos coplanares, en una línea no hay tres:
   a) ¿Cuántas líneas forman los puntos?
   b) ¿Cuántos triángulos se pueden formar?

5. Una persona desea hacer ramilletes de flores con 5 variedades diferentes de estas, ¿Cuántos ramilletes diferentes pueden hacerse si cada ramillete debe contener al menos una flor diferente?

6. Cuantas señales diferentes se pueden formar con 8 banderas de las cuales tres son rojas, 2 blancas, 2 amarillas y 1 negra.

7. Un estudiante tiene que resolver diez preguntas de trece en un examen. ¿Cuantas maneras diferentes tiene de escoger?

8. Un niño quiere llenar botes con canicas de doce colores diferentes. ¿Cuantos botes diferentes puede llenar si cada bote debe contener al menos una canica?

9.- Cuantos cócteles diferentes se pueden hacer con 15 frutas de las cuales 4 son duraznos, 3 fresas, 5 peras, 2 manzanas y un melón.

10. Una familia tiene seis hijos. Determinar la probabilidad de que sean tres niños y tres niñas.

11. Una caja contiene 12 medias de las cuales existen 5 negras, 4 cafés y tres blancas, se seleccionan aleatoriamente seis de ellas. ¿Cuál es la probabilidad de que la muestra contenga tres pares diferentes?

12. Para ir de A a B existen 5 caminos diferentes, si una persona desea viajar de A a B cuenta con cuatro tipos de transporte diferentes. De cuantas maneras puede viajar.

13. Una agencia de automóviles ofrece 5 tipos diferentes de autos, cada tipo con 6 colores diferentes y de 2,4 y 5 puertas. Otra agencia ofrece 4 tipos diferentes de autos, cada tipo con 5 colores diferentes y de 2, 4 y 5 puertas. ¿Cuántos autos diferentes tienen las dos agencias?

14.- Dos equipos jugaron un torneo de fútbol y básquetbol ganará aquel que gane dos juegos consecutivos o cuatro salteados ¿De cuantas maneras posibles puede acabar el torneo?

15.- Cuantas permutaciones se pueden hacer con las letras de la palabra computación si se toman todas las letras.

6.29 GRÁFICOS EN TURBO C

INTRODUCCIÓN

En esta sección se mostraran algunos elementos básicos que serán necesarias para modo gráfico:

Inicializar el modo gráfico.

Contar con un archivo de interfase (BGI).

Saber donde se encuentra el archivo de interfase (Ruta al BGI) y recomendable checar si hubo algún problema con la inicialización del modo gráfico antes de intentar cualquier cosa.

Cerrar el modo gráfico al terminar de usarlo.

Entre el initgraph y el closegraph es nuestra área de trabajo en modo gráfico, es ahí donde se realizan los trazos.

Empezaremos a conocer algunas de las funciones disponibles en C, serán las esenciales para poder trabajar en modo gráfico:

Función No. 1

Nombre: initgraph

Objetivo: Inicializa el modo gráfico. Inicialización del modo gráfico. ¿Por qué? Por omisión, el entorno del turbo C trabaja en modo texto, todo lo que hagamos será desplegado en modo texto en la pantalla, debido a esto, es necesario avisarle que queremos trabajar en modo gráfico, para poder hacer despliegues en la pantalla gráfica.

Sintaxis: initgraph (int *ControladorGráfico, int *ModoGráfico, char *RutaAlBGI);

Comentarios:

Controlador Gráfico: Es una variable de tipo entero que se pasa por referencia, la cual se utiliza para indicar el tipo de tarjeta gráfica que se está utilizando. En turbo C podemos trabajar con varias tarjetas gráficas aunque en la actualidad la mas utilizada es la VGA/EGA, que es la que se tomará como base en este tutorial, el turbo C, identifica las tarjetas controladoras por medio de números o también por medio de constantes predefinidas, en nuestro caso usaremos la constante VGA o número 9, por lo tanto es el valor que asignaremos a la variable con este fin.

Modo Gráfico: Es una variable de tipo entero que se pasa por referencia, la cual se utiliza para indicar el modo gráfico a utilizar, cada controlador gráfico soporta varios modos gráficos cada uno de ellos con distintas resoluciones. Para la controlado VGA/EGA los modos permitidos son:}

Modo Constante	Valor	Resolución	No. Colores	No. Págs. Video
VGALO	0	640 x 200	16	2
VGAMED	1	640 x 350	16	2
VGAHI	2	640 x 480	16	1

En nuestro caso usaremos el VGAHI o valor 2, por lo que será el valor que asignaremos a la variable con este fin.

Ruta AlBGI: Para que sea posible trabajar en modo gráfico desde el turbo C con la controladora y modo solicitados anteriormente se requiere de una interfase, turbo C proporciona éstas por medio de archivos que tienen extensión BGI (Borland Graphics Interface) en nuestro caso el nombre es EGAVGA.BGI. Así, este argumento de la función, es una cadena que le indica donde se encuentra este

archivo. Con respecto a esto, debemos saber donde se encuentra, comúnmente se encuentra en una subcarpeta de la carpeta del turbo C de nombre BGI, si fuera así, la carpeta sería c: \turboc\bgi, pero al manejar una ruta en una cadena las "\" deben ir dobles c:\\turboc\\bgi. Una sugerencia que podemos tomar en cuenta para facilitar este aspecto es la de realizar una copia del archivo BGI correspondiente en nuestro directorio de trabajo, caso en el cual la ruta quedaría vacía "".

Este aspecto del BGI es importante ya que todas las aplicaciones que incluyan el manejo de gráficos requerirán este archivo por lo que será necesario al distribuir esta aplicación incluir en el disco el archivo BGI correspondiente, además de que la ruta indicada en el initgraph coincida con la ubicación del archivo.

Posteriormente se explicará como se puede integrar en un solo archivo la aplicación y el BGI.

Programa

1

Inicialización del modo gráfico.

```
#include < graphics.h >   //Funciones gráficas
#include < stdio.h > // printf
#include < conio.h > // getch
void main ( )
{
   int gd,gm; //gd = graphdriver, gm= graphmode
   printf ("Presiona  para ir al modo gráfico");
   getch ( );
   gd =VGA; // Tarjeta VGA
   gm =VGAHI; // Modo VGA de alta resolución 640x480x16
   initgraph (&gd,&gm,"");
   printf ("Bienvenido al modo gráfico, !!  TRAZOS!!  Terminar");
```

```
    getch ( );
}
```

Si escribes este programa, puedes omitir todos los comentarios.

Podemos notar el cambio a modo gráfico por dos aspectos: El cursor de texto desaparece y las letras cambian un poco su aspecto se hacen más grandes.

Después de aplicar el initgraph y si no hay algún impedimento para tener acceso al modo gráfico, podemos empezar a realizar trazos en la línea siguiente.

Los letreros se imprimieron con printf que no es la forma adecuada de imprimir textos en el modo gráfico, al solicitar acceso al modo gráfico podría haber varios motivos por los que no sea posible tener acceso a él, si así fuera, ¿Qué caso tendría continuar el programa?, para esto, el initgraph retorna al sistema ciertos valores de referencia con los que se puede saber el resultado de intentar activar el modo gráfico, para esto pasemos a la función No. 2.

Función No. 2

Nombre: graphresult

Objetivo: Informa el resultado de la última acción relacionada con gráficos.

Sintaxis: int graphresult ( )

Comentarios:

Graphresult, retorna un número entero que corresponde a un código de error con respecto a la última acción relacionada con gráficos. Por el momento solo tomaremos la referencia de que un valor de 0 significa que se activó el modo gráfico y un valor diferente de cero que se presentó algún problema, es probable

que al empezar a trabajar con el modo gráfico el único problema que se presente sea el que la ruta al BGI no sea la adecuada.

Este código puede guardarse en alguna variable o hacer referencia directa a la función. Por ejemplo:

int cerror;

c error =graphresult ( );

if (cerror>0)

O también: if (graphresult ( ) > 0)

Programa

Inicialización del modo gráfico, verificando si hubo o no algún problema.

```
#include < graphics.h >
#include < stdio.h >
#include < stdlib.h > // exit
#include < conio.h >
void main ( )
{
     int gd,gm;
     printf ("Presiona  para ir al modo gráfico");
     getch ( );
      gd =VGA;
      gm =VGAHI;
      nitgraph (&gd,&gm,"");
```

```
  if (graphresult() >0 ) // Entraría a este bloque si el código
        {  // Devuelto por graphresult es diferente de 0
            printf ("Lo siento, no se puede iniciar el modo gráfico\n");
            printf ("quizá la ruta al BGI no es la adecuada");
            exit (0);
        }
      printf ("Bienvenido al modo gráfico, Terminar");
 getch ( );
}
```

Con esto, podemos empezar a practicar la programación estructurada junto con los gráficos:

Programa

Programa donde la parte de la inicialización del modo gráfico se realiza en una función, verificando si hubo o no algún problema.

```
#include < graphics.h >
#include < stdio.h >
#include < stdlib.h >
#include < conio.h >
void IniGra ( ); // Prototipo de la función
void main ( )
{
     printf ("Presiona  para ir al modo gráfico");
     getch ( );
     IniGra ( );
      printf ("Bienvenido al modo gráfico, Terminar");
      getch ( );
}
```

```
void IniGra ( )
{
      int gd = VGA, gm = VGAHI;
      initgraph (&gd,&gm,"");
        if (graphresult ( ) >0 )
       {
            printf ("Lo siento, no se puede iniciar el modo gráfico\n");
            printf ("quizá la ruta al BGI no es la adecuada");
            exit (0);
        }
}
```

Esta estructura es la que se utilizará de aquí en adelante, en los programas siguientes se considerará que ya se tiene escrita la función IniGra ( ) y solo se hará referencia a ella.

Para cerrar el ciclo de uso del modo gráfico, recordemos que partimos del modo texto y estamos cambiando al modo gráfico, es necesario que al terminar el trabajo con el modo gráfico le “avisemos” al sistema que regrese al punto de partida, veamos la función No. 3.

Función No. 3

Nombre: closegraph

Objetivo: Cerrar el modo gráfico.

Sintaxis: closegraph ( );

Comentarios:

Cierra el modo gráfico y libera todos los recursos asignados a él.

Debe utilizarse cuando ya no se desee utilizar el modo gráfico o se termine el programa en curso.

Programa

Programa para inicializar el modo gráfico y cerrarlo al terminar.

```
#include < graphics.h >
#include < stdio.h >
#include < stdlib.h >
#include < conio.h >
void IniGra ( );
void main( )
{
      printf ("Presiona  para ir al modo gráfico");
      getch ( );
      IniGra ( );
      printf ("Bienvenido al modo gráfico, !  TRAZOS!!  Terminar");
      getch  ( );
      closegraph ( ); //Se cierra el modo gráfico al terminar.
}
void IniGra ( )
{
        int gd=VGA,gm=VGAHI;
        initgraph (&gd,&gm,"");
          if (graphresult() >0 )
          {
              printf ("Lo siento, no se puede iniciar el modo gráfico\n");
```

```
            printf ("quizá la ruta al BGI no es la adecuada");
            exit (0);
        }
}
```

La siguiente tabla contiene los prototipos de las funciones para preparar y manipular esta parte gráfica

Funciones

arc	bar	bar3d	circle
cleardevice	clearviewport	closegraph	detectgraph
drawpoly	ellipse	fillellipse	fillpoly
floodfill	getarccoords	getaspectratio	getbkcolor
getcolor	getdefaultpalette	getdrivername	getfillpattern
getfillsettings	getgraphmode	getimage	getlinesettings
getmaxcolor	getmaxmode	getmaxx	getmaxy
getmodename	getmoderange	getpalette	getpalettesize
getpixel	gettextsettings	getviewsettings	getx
gety	graphdefaults	grapherrormsg	graphfreemem
graphgetmem	graphresult	imagesize	initgraph
installuserdriver	installuserfont	line	linerel
lineto	moverel	moveto	outtext
outtextxy	pieslice	putimage	putpixel
rectangle	registerbgidriver	registerbgifont	restorecrtmode
sector	setactivepage	setallpalette	setaspectratio
setbkcolor	setfillpattern	setfillstyle	setgraphbufsize
setgraphmode	setlinestyle	setpalette	setrgbpalette
settextjustify	settextstyle	setusercharsize	setviewport
setvisualpage	setwritemode	textheight	textwidth

Ahora veremos algunas de estas funciones de manera mas detallada:

Función arc

```
void far arc(int x, int y, int comienzo_angulo, int final_angulo, int radio);
```

Esta función creará un arco circular. El arco tiene como centro el punto especificado por los argumentos **x** e **y**, y es dibujado con el radio especificado: **radio**. El arco no está rellanado, pero es dibujado usando el color actual. El arco comienza al ángulo especificado por el argumento **comienzo_angulo** y es dibujado en la dirección contraria al de las agujas del reloj hasta llegar al ángulo especificado por el argumento **final_angulo**. La función *arc* usa el este (extendiéndose hacia la derecha puede usarse para establecer el grosor del arco. La del centro del arco en la dirección horizontal) como su punto de 0 grados. La función setlinestyle función *arc*, sin embargo, ignorará el argumento **trama** de la función setlinestyle.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
 int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int radio;
   initgraph ( &gdriver, &gmodo, "C:\\BC5\\BGI" );
   for ( radio = 25; radio < 175; radio += 25)
      arc ( 320, 175, 45, 135, radio );
   getch  ( );    /* Pausa */
   closegraph ( );
```

```
  return 0;
}
```

Función bar

```
void far bar(int izquierda, int superior,
int derecha, int inferior);
```

Esta función dibujará una barra rectangular y rellenada de dos dimensiones. La esquina superior izquierda de la barra rectangular está definida por el argumento izquierdo y superior. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha de la barra.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  int x, y, color, relleno;
  initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
  x = 20;
  y = 20;
  color = 1;
  fill = 1;
  do {
    setfillstyle (fill, color);
    bar (x, y, x+40, 320);
    x += 40;
    y += 10;
```

```
    color = (color+1) % 16;
    fill = (fill+1) % 12;
  } while ( x < 620 );
  getch ( );    /* Pausa */
  closegraph ( );
  return 0;
}
```

Función circle

```
void far circle (int x, int y, int radio);
```

Esta función se usa para dibujar un círculo. Los argumentos **x** e **y** definen el centro del círculo, mientras que el argumento **radio** define el radio del círculo. El círculo no es rellenado pero es dibujado usando el color actual. El grosor de la circunferencia puede ser establecido por la función setlinestyle; sin embargo, el estilo de la línea es ignorado por la función *circle*. La proporción anchura-altura para el modo actual es considerado cuando se calcula el círculo. Por esta razón, alterando los valores de defecto x e y de los factores de anchura-altura afectará el círculo (ya no seguirá siendo redondo).

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  int relleno;
  registerbgidriver (EGAVGA_driver);
  initgraph (&gdriver, &gmodo, "");
  initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
```

```
  relleno = 1;
  setlinestyle (SOLID_LINE, relleno, THICK_WIDTH);
  circle(300, 200, 80);
  getch ( );    /* Pausa */
  closegraph ( );
  return 0;
}
```

Función line

```
void far line (int x1, int y1, int x2, int y2);
```

Esta función es usada para conectar dos puntos con una línea recta. El primer punto es especificado por los argumentos **x1** e **y1**. El segundo punto es especificado por los argumentos **x2** e **y2**. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual. La posición del cursor gráfico no es afectado por la función *line*.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  registerbgidriver (EGAVGA_driver);
  initgraph (&gdriver, &gmodo, "");
  line (20, 40, 350, 100);
  line (400, 30, 50, 250);
  getch ( );
  closegraph ( );
```

```
  return 0;
}
```

Función putpixel

```
void far putpixel (int x, int y, int color);
```

Esta función es usada para asignar el valor del color a un píxel en particular. La posición del píxel en cuestión está especificado por los argumentos **x** e **y**. El argumento **color** especifica el valor del color del píxel.

Existen varios valores para describir colores

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  int t;
  initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
  for (t=0; t<200; t++)
    putpixel (100+t, 50+t, t%16);
  getch ( );
  closegraph ( );
  return 0;
}
```

Función ellipse

```
void far ellipse (int x, int y, int comienzo_angulo,
int final_angulo, int x_radio, int y_radio);
```

Esta función es usada para dibujar un arco elíptico en el color actual. El arco elíptico está centrado en el punto especificado por los argumentos **x** e **y**. Ya que el arco es eilptíco el argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical. El arco elíptico comienza con el ángulo especificado por el argumento **comienzo_angulo** y se extiende en un sentido contrario a las agujas del reloj al ángulo especificado por el argumento **final_angulo**. La función *ellipse* considera este - el eje horizontal a la derecha del centro del elipse - ser 0 grados. El arco elíptico es dibujado con el grosor de línea actual como es establecido por la función setlinestyle. Sin embargo, el estilo de línea es ignorado por la función *ellipse*.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
  ellipse (300, 150, 45, 225, 100, 50);
  getch ( );    /* Pausa */
  closegraph ( );
  return 0;
}
```

Función fillellipse

```
void far fillellipse (int x, int y,
int x_radio, int y_radio);
```

Esta función es usada para dibujar y rellenar una elipse. El centro de la elipse es especificado por los argumentos **x** e **y**. El argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical de la elipse. La elipse es dibujado con el perímetro en el color actual y rellenada con el color de relleno y la trama de relleno actuales.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   trama = SOLID_FILL;
   color = 4;
   setfillstyle (trama, color);
   fillellipse (300, 150, 100, 50);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función linerel

```
void far linerel (int dx, int dy);
```

Esta función es usada para dibujar una línea recta a una distancia y dirección predeterminadas desde la posición actual del cursor gráfico. El argumento **dx** específica el número relativo de píxels para atravesar en la dirección horizontal. El argumento **dy** específica el número relativo de píxels para atravesar en la

dirección vertical. Estos argumentos pueden ser tanto valores positivos como negativos. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual desde la posición actual del cursor gráfico a través de la distancia relativa especificada. Cuando la línea esté terminada, la posición del cursor gráfico es actualizado al último punto de la línea.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   moveto (20, 20);
   linerel (20, 40);
   linerel (40, 30);
   getch ( );
   closegraph ( );
   return 0;
}
```

Función fillpoly

void far fillpoly (int numpuntos, int far *puntos);

Esta función es usada para crear un polígono rellenado. El argumento **numpuntos** es usado para definir el número de puntos en el polígono. Al contrario que la función drawpoly, la función automáticamente cierra el polígono. El argumento ***puntos** apunta a un array de números de longitud **numpuntos** multiplicado por 2. Los dos primeros miembros del array identifica las coordenadas *x* e *y* del primer punto, respectivamente, mientras que los dos siguientes especifican el siguiente

punto, y así sucesivamente. La función *fillpoly* dibuja el perímetro del polígono con el estilo de línea y color actuales. Luego, el polígono es rellenado con la trama de relleno y color de relleno actuales.

Ejemplo;

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;
   int punto s[6] = {300, 50, 500, 300, 100, 300};
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   trama = SLASH_FILL;
   color = 4;
   setfillstyle (trama, color);
   fillpoly (3, puntos);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función rectangle

```
void far rectangle (int izquierda,
int superior, int derecha, int inferior);
```

Esta función dibujará un rectángulo sin rellenar su interior usando el color actual. La esquina superior izquierda del rectángulo está definida por los argumentos **izquierda** y **superior**. Estos argumentos corresponden a los valores *x* e *y* de la

esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha del rectángulo. El perímetro del rectángulo es dibujado usando el estilo y grosor de línea actuales.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   rectangle (20, 20, 400, 300);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Funcion drawpoly

```
void far drawpoly (int numpuntos, int far *puntos);
```

Esta función es usada para crear un polígono con un número especificado de puntos. El argumento **numpuntos** es usado para definir el número de puntos en el polígono. Para la función *drawpoly*, el número de puntos debe ser el número actual de puntos más 1 para poder crear un polígono cerrado. En otras palabras, el primer punto debe ser igual al último punto. El argumento ***puntos** apunta a un array de números de longitud **numpuntos** multiplicado por 2. Los dos primeros miembros del array identifica las coordenadas *x* e *y* del primer punto, respectivamente, mientras que los dos siguientes especifican el siguiente punto, y así sucesivamente. La función *drawpoly* dibuja el perímetro del polígono con el estilo de línea y color actuales, pero no rellena el polígono.

La función *drawpoly* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int puntos [8] = {300, 50, 500, 300, 100, 300, 300, 50};
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   drawpoly (4, puntos);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función sector

```
void far sector (int x, int y,
int comienzo_angulo, int final_angulo, int x_radio, int y_radio);
```

Esta función es usada para dibujar una cuña elíptica. El centro de la cuña elíptica es especificado por los argumentos **x** e **y**. El argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical de la cuña elíptica. La cuña elíptica comienza al ángulo especificado por el argumento **comienzo_angulo** y es dibujado en la dirección contraria al de las agujas del reloj hasta llegar al ángulo especificado por el argumento **final_angulo**. La cuña elíptica es dibujado con el perímetro en el color actual y rellenada con el color de relleno y la trama de relleno actuales.

Valor de retorno:

La función *sector* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setfillstyle (SOLID_FILL, 6);
   sector (300, 150, 45, -45, 150, 50);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

EJERCICIOS:

1. Ubicar en la computadora la ruta del archivo EGAVGA.BGI y si así se desea hacer una copia en nuestro directorio de trabajo.
2. Elaborar un programa que pida los datos de una circunferencia, sacar su área y con la funcion circle dibujar en pantalla.

6.30 MODIFICADORES DE ACCESO

C posee 2 modificadores de tipo que se utilizan para controlar la forma en que se puede acceder a las variables. Estos son const y volatile.

Const

Las variables que se declaran con el modificador Const no pueden cambiar durante la ejecución del programa.

Const float version = 3.20;

Volatile

Es una instrucción para indicar en el programa que las variables declaradas en ella van cambiando sin que esos cambios los indique directamente el programa
Volatile int reloj;

ESPECIFICADORES DE CLASE DE ALMACENAMIENTO

C admite 4 especificadores de clase de almacenamiento:

auto, extern, static, register

Auto

El especificador auto se utiliza para declarar variables locales.

Ejemplo:

```
#include<stdio.h>
#include<conio.h>
int sum a lot
main ( )
{
auto int…
```

```
suma = suma (a,b)
}
Suma (int x, auto int y)
```

Extern

Se utiliza para reindicar o reutilizar las variables declaradas en un principio que van siendo utilizadas en otros archivos, para que no haya confusión con el compilador a la hora de querer saber que variable utilizar de las tantas copias que se pudieran tener sin emplear la palabra extern.

Ejemplo:

```
#include...
```

Archivo 1

```
int x,y;
char car
main ( )
{
.
.
.
}
función 1
{
    x=23;
}
```

Archivo 2

```
extern int x, y;
extern char car;
main ( )
{
x=y/10;
}
función 15
{
 y=10;
 }
```

Static

Son variables permanentes, bien sea dentro de su propia función o dentro del archivo.

Variables locales Static

Se declaran de esa manera para que en una función o procedimiento conserve el valor una vez que haya salido de las instrucciones.

Ejemplo:

```
#include
int serie;
main ( )
{
int i;
for (i=0; i<10; i++)
prinf (“%d ” , serie ( ) );
return 0:
}
serie
{
static int num_serie;
num_serie=num_serie+23;
return (num_serie);
}
```

Variables globales Static

Se declaran para inicializar con cierto valor en la primera llamada, y que las variables tengan de forma permanente los valores que se vayan adquiriendo aún saliendo del procedimiento de la función.

Ejemplo:

```
#include
main (  )
{
int i;
cnst int a=2;
comienzo. Serie (a)
for (i=0; i<10; i++)
printf (“%d “ , serie (  );
return;
}
```

Register

Se declaran para que esas variables se asignen a registros del CPU para emplear de forma más rápida las variables que mandemos llamar.

EJERCICIOS:

1. convierta los siguientes números de decimal a binario
   a) 10
   b) 28
   c) 30
   d) 54

   e) 19
2. cambie los siguientes números de binario a decimal
   a) 10001
   b) 10
   c) 11001
   d) 10101
   e) 111
3. cambie los siguientes números de exadecimal a decimal
   a) A
   b) 12
   c) B14
   d) 1C14
   e) FF
4. cambie los siguientes numeros de binario a exadecimal
   a) 101
   b) 11001
   c) 10011
   d) 1110001
   e) 1100111
5. lea n números y póngalos en orden.
6. dados tres números y determine cual es el mayor.
7. determine si un carácter es número, letra o carácter ascci.
8. dados dos números, preguntar si se quieren sumar o restar y realice la operación.
9. utilice la sentencia if para sumar los primeros n números impares con las etiquetas y goto.
10. Elaborar la progresión geométrica e interpolar n números.
11. Hacer un menú de opciones que dados n números proporciones el promedio, la media y la mediana.
12. Dado n numero preguntar en un menú de opciones que desea hacer:
    a) Serie del seno

b) Serie coseno

c) Serie e^x

13. Elabora un programa para leer una cadena.
14. Obtener la longitud de la cadena por medio de punteros.
15. Localizar algún archivo determinado.
16. Obtener algún dato de un inventario.
17. Obtener información de una base de datos.
18. realice una función en la que se escriban los números del l al 10 y después de ellos 10 asteriscos, del 11 al 20 y después de ellos 10 asteriscos... y así hasta llegar al 100.
19. escriba en pantalla su nombre entre dos líneas.
20. realice una función en la que se pida encontrar el mayor de dos números.
21. utilice la serie del seno que desarrollo en el ejercicio para determinar el ángulo de un triangulo dados dos de sus lados uno de sus ángulos.
22. utilice la serie del coseno que desarrollo en el ejercicio para determinar el angulo de los lados de un triangulo.
23. La famosa sucesión de Fibonacci puede definirse en términos de recurrencia de la siguiente manera:

    (1) Fib (1) = 1 ; Fib (0) = 0

    (2) Fib (n) = Fib (n-1) + Fib (n-2) si n >= 2

¿Cuantas llamadas recursivas se producen para Fib (6)?. Codificar un programa que calcule Fib (n) de forma iterativa.

Nota: no utilizar estructuras de datos, puesto que no queremos almacenar los números de Fibonacci anteriores a n; sí se permiten variables auxiliares.

24. *Dado un array constituido de números enteros y que contiene N elementos siendo N >= 1, escribir una función que devuelva la suma de todos los elementos mayores que el último elemento del array.*

25. *Dado un array constituido de números enteros y que contiene N elementos siendo N >= 1, escribir una función que devuelva cierto si la suma de la primera mitad de los enteros del array es igual a la suma de la segunda mitad de los enteros del array.*
26. *Dados dos arrays A y B de longitud n y m respectivamente, n >= m cuyos elementos estén ordenados y no se repiten, determinar si todos los elementos de B están contenidos en A. Recordar que los elementos están ordenados, de esta manera basta con realizar un único recorrido sobre cada array.*

6.31 MANEJO DE ARCHIVOS

Los archivos son la forma más común de los flujos.

Las funciones para manipular los Archivos se encuentran en el archivo de cabecera <stdio.h> y vale la pena mencionarlos:

Función	Descripción
fopen ( )	Abre un flujo
fclose ( )	Cierra un flujo
Putc	Escribe un carácter en un flujo
getc ( )	Lee un carácter desde un flujo
fputs ( )	Escribe una cadena en un flujo
fgets ( )	Obtiene una cadena de un flujo
fseek ( )	Salta al byte especificado en un flujo
fprintf ( )	Imprime datos con formato en un flujo
fscanf ( )	Lee datos con formato en un flujo
eof ( )	Devuelve verdadero o falso si se halla el fin del archivo.

FLUJOS

Los flujos dan canales de comunicación entre el archivo y el programa. Por ejemplo, el flujo de entrada estándar permite que un programa lea datos desde el teclado, el flujo de salida estándar permite que un programa imprima datos en la pantalla. Un flujo se relaciona con un archivo utilizando una operación de apertura y se desliga del archivo con una operación de cierre.

Existen 2 tipos de flujos: Texto y binarios.

El flujo de Texto se utiliza con conversiones de Caracteres. Es decir, puede no haber correspondencia entre lo que se envía al flujo y lo que se escribe en el archivo.

El flujo Binario tiene una correspondencia directa entre lo que se envía y se escribe en el archivo.

ARCHIVO

Un archivo es el grupo de información que está en una entidad física real que recibe los datos, ya sea CD o diskette.

ABRIR UN ARCHIVO puede ser para varios propósitos: lectura, escritura o ambos, por lo que se utiliza la siguiente función para ello:

FILE fopen (char *f, char *modo); que es lo mismo que fopen (Nombre del Archivo, Modo). Los modos de Abrir al archivo se especifican en la siguiente tabla:

Acceso	Descripción
"r"	Abre un archivo para lectura. El archivo debe de existir.
"w"	Abre para escritura. Si el archivo no existe se crea, pero si existe se borra su información para crearla de nuevo.
"a"	Abre para escribir al final de el. Si el archivo no existe se crea.
"r+"	Abre para Escritura/Lectura. El archivo debe existir.
"w+"	Abre para Escritura/Lectura. Si el archivo no existe se crea, pero si existe su información se destruye para crearla de nuevo.
"a+"	Abre para escribir al final de el y leer. Si el archivo no existe se crea.
"rb"	Abre un archivo binario para lectura. El archivo debe existir.
"wb"	Abre un archivo binario para escritura. Si el archivo no existe se crea, pero si existe su información se destruye para crearla de nuevo.
"ab"	Se abre un archivo binario para escribir al final de el. Si el archivo no existe se crea.
"rb+"	Abre un archivo binario para lectura/Escritura. El archivo debe existir.
"wb+"	Abre un archivo binario para lectura/escritura. Si elarchivo no existe se crea, pero si existe su información se destruye para crearla de nuevo.
"ab+"	Abre un archivo binario para lectura y añadir. Si el archivo no existe se crea.

Para abrir un archivo se debe tener un flujo (apuntador tipo archivo) que apunte a la estructura FILE.

Ejemplo:

FILE *flujo; /* Se declara un flujo */

```
flujo = fopen ("miarch.dat","r");
```

es una buena práctica revisar si un archivo se pudo abrir correctamente

```
if ( (flujo = fopen ("miarch.dat","r")) == NULL )
{
        Printf ("No se pudo abrir %s\n","miarch.dat");
        exit (1);       // ‘exit (1)’ termina el programa con
}                       // un estado de error.
```

http://www.itq.edu.mx/vidatec/maestros/sis/mlopez/Tutorial/archi.htm - f0#f0

CERRAR UN ARCHIVO después de trabajar es algo que debe de hacerse, si no se hace este se cierra automáticamente cuando finaliza el programa. Se cierra de la siguiente manera:

```
fclose (FILE *f)
```

Ejemplo de programa que Escribe y lee carácter por carácter de un archivo las instrucciones usadas son putc y getc. Observe el manejo de fopen y fclose en los siguientes ejemplos.

Int putc (int c, FILE*f); Escribe el carácter c en un archivo apuntado por f

Int getc (FILE *f); Devuelve el carácter leído e incrementa la posición actual del indicador del archivo, si se detecta el fin del archivo se devuelve EOF.

```
# include <stdio.h>
void main(void)
{
  char opcion;
do
{       clrscr();
        printf("Menú de Opciones.\n");
        printf("1.- Escritura de un Archivo. \n");
        printf("2.- Lectura de un Archivo. \n");
        printf("3.- Salida. \n");
        printf("Elija su Opción:  ");
        opcion= getch();
                switch(opcion)
                {
                  case '1':{escritura(); break;}
                  case '2':{ lectura(); break;}
                  case '3':{break;}
                }
}while (opcion!='3');
}

escritura(void)
{  FILE  *apuntador;
 char nom_archivo[13], ch;
 clrscr();
printf("\n Dame el Nombre del Archivo al que va a Escribir:  ");
gets(nom_archivo);
apuntador = fopen(nom_archivo,"w");
while ((ch=getche())!='\r')
        putc(ch, apuntador);getch();
fclose(apuntador);
return 0;
}

lectura(void)
{ FILE *apuntador;
char nom_archivo[13],ch;
clrscr();
printf("\n Deme el Nombre del archivo que va a leer:  ");
gets(nom_archivo);
apuntador=fopen(nom_archivo, "r");
while((ch=getc(apuntador))!=EOF)
        printf("%c",ch);
getch();
fclose(apuntador);
return 0;   }
```

FILE *apuntador establece que "apuntador" es un apuntador a la estructura FILE. La forma de Escribir y leer la información por medio de strings es por medio de las instrucciones fgets y fputs. Int fputs (char *cad, FILE *f); Escribe una cadena de caracteres aun archivo apuntado por f. Int fgets (char *cad, int n, FILE *f); Lee una cadena de Caracteres.

Programa que escribe strings a un archivo.

```
#include<stdio.h>
void main(void)
{ FILE *fptr;
char string [81];
clrscr();
fptr=fopen("PRUEBA.TXT","w");
printf("Este Programa escribe strings en un archivo \n");
printf("Teclear <ENTER> despues de cada frase\n");
while(strlen(gets(string))>0)
{
fputs(string,fptr);
fputs("\n",ftpr);
}
getche();
fclose(fptr);
}
```

Las instrucciones fread y fwrite se utilizan en archivos en el que se utilizan estructuras para definir la longitud de cada record. Size_t fwrite (void *buffer, size_t num_bytes, size_t n, FILE *f); Escribe n registros de longitud num_bytes desde la memoria apuntada por buffer al archivo f. Size_t fread (void *buffer, size_t num_bytes, size_t n, FILE *f); Lee n registros de num_bytes en la memoria apuntada hacia buffer desde el archivo f.

EXISTEN MUCHAS OPERACIONES ASOCIADAS A ARCHIVOS EN C++, LAS MÁS ELEMENTALES SON:

1. Creación de Archivo.- En este proceso se pretende solamente crear un archivo nuevo en disco con su nombre tipo y especialidad de almacenamiento de datos apropiado.
2. Apertura de Archivos.- En este caso se pretende abrir un archivo ya existente en disco para procesarlo ya sea para cargar o grabar estructuras en sus registros o leer algún registro en especial para mandarlo a una variable de cualquier tipo.
3. Creación de Archivos.- No confundir creación con apertura, creación es un proceso que solo se ejecuta una sola vez en la vida de un archivo, mientras que apertura, siempre se esta realizando por los programas especializados en algún proceso.

4. Cierre de archivos.- Es la operación mas importante en cualquier programa que maneje archivos, o se cierra el archivo como ultima instrucción del programa o se vera el anuncio ABORT, RETRY, FAIL.
5. Altas en archivo.- En este proceso se captura una estructura en memoria con sus datos pertinentes y después se graba la estructura al archivo en disco.
6. Lectura de archivo.- En este proceso se abre el archivo, se manda el registro de disco a una estructura en memoria para su procesamiento.
7. Consulta de archivos.- En este proceso se pretende desplegar todos los registros del archivo en disco a la pantalla ya sea consola o mejor aún, a una pagina html.
8. Búsqueda en archivos.- Una de las operaciones más comunes consiste en que el usuario pide toda la información de algún renglón en disco proporcionando la información de algún campo generalmente el campo clave de la estructura.
9. Filtros.- En este proceso el usuario esta interesado en algún conjunto de renglones con características comunes (condición), por ejemplo todos los alumnos de "sistemas" o todos los empleados que ganen mas de 500.00 colones, o todos los clientes que sean de "Heredia", etc.
10. Modificaciones de registros o archivos.- Problema muy común, donde los datos originales ya grabados se tienen que cambiar o actualizar.
11. Bajas de registros.- también muy común este proceso.

CREACIÓN ARCHIVOS SECUENCIAL DISCO C++

En este proceso se pretende solamente crear un archivo secuencial en disco.

```
//ejemplo 1: archivo secuencial
#include <stdio.h>
#include <conio.h>
#include <string.h>
Struct
{
        int carnet;
        char nombre [30];
```

```
        int edad;
} alumno;
void main ( )
{  clrscr ( );
        //creando y cerrando el archivo en disco
        FILE *archdisco;
        archdisco = fopen ("c:\\alumnos.dat","w");
        fclose (archdisco);
        printf ("EL ARCHIVO HA SIDO
CREADO...");
        getchar ( );
}
```

Lo primero que se crea es una variable de tipo puntero a un archivo a disco (instrucción FILE y debe ser en MAYUSCULAS) llamada archdisco que almacenará la dirección física de el archivo en disco. Como segundo paso se abre el archivo con la instrucción fopen ( ), si hubiese error regresaría NULL:

archdisco = fopen ("c:\\alumnos.dat","w");

GRABACIÓN Y LECTURA DISCO C++

Estos dos procesos son los casos más comunes y frecuentes que se pueden realizar con un archivo de disco.

```
//ejemplo 2: archivo secuencial: grabación y lectura
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct
{
```

```
        int carnet;
        char nombre [30];
        int edad;
} alumno;

FILE *archdisco;
void main ( )
{
        clrscr ( );
        // captura de campos
        printf ("Escriba el numero de carnet: ");
        scanf ("%d",alumno.carnet);
        getchar ( );
        printf ("Escriba el nombre del estudiante: ");
        gets (alumno.nombre);
        printf ("Escriba la edad: ");
        scanf ("%d",alumno.edad);
        // grabando a disco
        archdisco = fopen ("c:\\alumnos.dat","at+");
        fwrite (&alumno,sizeof (alumno),1,archdisco);
        fclose (archdisco);

        //avisando al usuario
        printf ("\n Los datos del alumno han sido guardados...");
        getchar ( ); getchar ( );
}
```

La primera observación es que se esta usando el "fopen ( )" en modo "at+" en lugar de modo "w" pero es para lograr dos cosas (revisar los apuntes anteriores de modos de apertura).

La única instrucción nueva es: **fwrite (&persona, sizeof (alumno), 1, archdisco);**

Como se observa ocupa 4 parámetros que son:

- ✓ fwrite ( ) ocupa primero conocer cuales datos va a almacenar en disco, aquí se le esta indicando que es el dato que se tiene en la dirección de memoria donde esta el registro "alumno".

- ✓ fwrite ( ), ocupa conocer cuantos bytes de información debe grabar, para esto se tienen dos opciones o se le da el valor exacto por ejemplo 64 bytes o 39 bytes o más fácil aún se usa sizeof ( ) que regresa el tamaño del dato.

- ✓ fwrite ( ), necesita conocer también cuantas estructuras o registros a la vez debe grabar por lo general es un solo registro, pero más adelante estudiarán que es posible grabar más de un registro a la vez y esto es de mucho provecho, porque por ejemplo si en un sistema se ocupa grabar 1000 registros y usamos fwrite ( ) de uno en uno, quiere decir que habría mil accesos a disco.

- ✓ fwrite ( ) también ocupa conocer exactamente en que cluster, sector y byte exacto del disco duro debe grabar el registro, la primera opción seria desarmar el disco duro y ver donde hay lugar para poner el archivo J o mejor aún usar la variable archdisco que ya tiene esa dirección física del archivo en disco.

LECTURA DE REGISTROS

```
//ejemplo 3: archivo secuencial: lectura de registros

#include <stdio.h>
#include <conio.h>
#include <string.h>
```

```
struct
{
        int carnet;
        char nombre [30];
        int edad;
} alumno;
FILE *archdisco;
void main ( )
{
        clrscr ( );
        // leyendo disco
        archdisco = fopen ("c:\\alumnos.dat","at+");

        // aquí siempre debe empezar el ciclo de lectura
        // y fread ( ) regresa siempre cuantas estructuras leyó

        while (fread (&alumno, sizeof(alumno), 1, archdisco) == 1)
        {
          // desplegando estructuras
          printf ("Carnet..: %d \n",alumno.carnet);
          printf ("Nombre..: %s \n",alumno.nombre);
          printf ("Edad....: %d \n",alumno.edad);
          printf ("--------------------------------\n\n");
        }; // aquí termina while
        // no olvidar cerrar archivo y siempre fuera de while
        fclose (archdisco);
        getchar ( );
}
```

En fopen ( ) se uso modo "a+". En lugar de fwrite ( ), se usa fread ( ) con los mismos cuatro parámetros. También recordar que fread ( ), nos regresa la

cantidad de registros que leyó del disco, por eso el ciclo while se convierte en falso cuando fread ( ) regresa 0 y esto indica que se llego al fin del archivo.

BÚSQUEDA DE ARCHIVOS C++

En este tema se analiza la búsqueda de un registro o renglón determinado.

En este proceso el usuario del programa quiere que se despliegue un y sólo un registro de información proporcionando un dato de búsqueda, generalmente la clave del registro.

```
//ejemplo 4: archivo secuencial: búsqueda de registros

#include <stdio.h>
#include <conio.h>
#include <string.h>
struct
{
        int carnet;
        char nombre[30];
        int edad;
} alumno;
FILE *archdisco;

void main ( )
{
        clrscr ( );
        // cargando clave a buscar
        printf ("Escriba el carnet que desea buscar: ");
        int clave;
        scanf ("%d",&clave);
```

```
        getchar ( );

        //abriendo, leyendo, cargando estructura
        archdisco = fopen ("c:\\alumnos.dat","at+");

        // aquí siempre debe empezar el ciclo de lectura
        // y fread ( ) regresa siempre cuantas estructuras leyó

        int mensaje=1; //se usara para verificar la salida de un mensaje

        while (fread (&alumno, sizeof (alumno), 1, archdisco) == 1)
        {  // desplegando estructura buscada
                if ( clave == alumno.carnet)
                        { printf ("\nCarnet...: %d \n",alumno.carnet);
                          printf ("Nombre...: %s \n",alumno.nombre);
                          printf ("Edad.....: %d \n",alumno.edad);
                          printf ("----------------------------------\n\n");
                          mensaje=0;
                          break; // rompe el ciclo y deja de buscar.
                        } // del if
        }; // del while
        if (mensaje==1)
          { printf ("\nNo se encontraron datos para ese carnet..\n");
                 printf ("---------------------------------------");
          }
                // no olvidar cerrar archivo
        fclose (archdisco);
        getchar ( );
}
```

Recordar que la característica principal de un archivo secuencial de c++ es que no es posible acceder a un registro o renglón específico o determinado sino que se deberá recorrer todos los n-1 renglones anteriores. Esta situación se da porque al construir un registro cualquiera con un montón de campos strings a la hora de almacenar dichos registros, estos registros tendrán tamaños diferentes, esta es la razón principal por la cual al buscar un registro específico se tiene que recorrer y validar todos los registros anteriores.

FILTROS C++

Otro problema similar al anterior es el de filtros, es decir en muchas ocasiones es necesario obtener información acerca de un subconjunto de renglones del archivo. Por ejemplo todos los estudiantes que sean mayores de 17 años, o todos los clientes que sean de Heredia, etc., a esto le llamamos filtros o condiciones.

El programa siguiente es muy similar al de búsquedas, solamente que establece un filtro a través del campo edad. Veamos:

```
//ejemplo 5: archivo secuencial: filtro de registros
#include <stdio.h>
#include <conio.h>
#include <string.h>

struct
{
        int carnet;
        char nombre [30];
        int edad;
} alumno;
FILE *archdisco;
void main ( )
```

```
{       clrscr ( );
        // cargando clave a buscar
        printf ("Buscar los datos de los alumnos cuya edad sea:");
        int xdato;
        scanf ("%d",&xdato);
        getchar ( );

        //abriendo, leyendo,cargando estructura
        archdisco = fopen ("c:\\alumnos.dat","at+");
        // aquí siempre debe empezar el ciclo de lectura
        // y fread ( ) regresa siempre cuantas estructuras leyó

        int mensaje=1; //se usara para verificar la salida de un mensaje

        while (fread (&alumno, sizeof (alumno), 1, archdisco) == 1)
        {  // desplegando estructura buscada

                if ( xdato == alumno.edad)
                        { printf ("\nCarnet...: %d \n",alumno.carnet);
                          printf ("Nombre...: %s \n",alumno.nombre);
                          printf ("Edad.....: %d \n",alumno.edad);
                          printf ("----------------------------------\n\n");
                          mensaje=0;
                        } // del if

        }; // del while
        if (mensaje==1)
          { printf ("\nNo se encontraron datos para ese carnet..\n");
                 printf ("----------------------------------------");
          }
                // no olvidar cerrar archivo
fclose (archdisco);
        getchar ( );
}
```

BAJAS O ELIMINACIONES EN ARCHIVOS CON C++

Eliminación o bajas es el proceso por medio del cual algunos registros del archivo son purgados del archivo, existen dos maneras por las cuales se puede realizar ese proceso: físico y lógico.

En la primera manera de la cual se proporciona el ejemplo correspondiente se usaran dos archivos, el archivo original y un archivo temporal, el procedimiento o algoritmo es muy sencillo, se lee el registro del archivo original y si no es el registro a eliminar entonces se almacena en el archivo temporal, cuando se termina de procesar todo el archivo original, el archivo temporal sólo contendrá todos los registros que no se quisieron eliminar, ya con estos dos archivos se procede a eliminar o borrar usando la instrucción remove de el archivo original y se procede a renombrar usando la instrucción rename de el archivo temporal como nuevo archivo original.

```
//ejemplo 6: archivo secuencial: eliminación de registros
#include <stdio.h>
#include <conio.h>
#include <string.h>

Struct
{       int carnet;
        char nombre [30];
        int edad;
} alumno;

FILE *archdisco;    // tiene TODOS los registros

FILE *archauxiliar; // tendra todos, MENOS los borrados
void main ( )

{       clrscr ( );
```

```
        // cargando clave a buscar
        printf ("Escriba el carnet que desea buscar y borrar: ");
        int clave;
        scanf ("%d",&clave);          getchar ( );

        archdisco = fopen ("c:\\alumnos.dat","r+"); //abre solamente para lectura
        archauxiliar = fopen ("c:\\temporal.dat","at+"); //abre para agregar registros
        int mensaje=1; //se usara para verificar la salida de un mensaje
        while (fread (&alumno, sizeof(alumno), 1, archdisco) == 1)
        {  // desplegando estructura buscada y eliminada

                if ( clave == alumno.carnet)
                        { printf ("\nCarnet...: %d \n",alumno.carnet);
                          printf ("Nombre...: %s \n",alumno.nombre);
                          printf ("Edad.....: %d \n",alumno.edad);
                          printf ("---------------------------------\n\n");
                          mensaje=0;
                        } // del if
          else
                  { //se guarda el registro recién leído en el archivo auxiliar
                          fwrite (&alumno, sizeof (alumno), 1, archauxiliar);
                  }

        }; // del while
        fclose (archdisco); // cerrar los dos archivos
        fclose (archauxiliar);

remove ("c:\\alumnos.dat");  //eliminar el archivo del disco.
        rename ("c:\\temporal.dat","c:\\alumnos.dat"); //renombrar archivo
        if (mensaje==1)
```

```
          { printf ("\nNo se encontraron datos para ese carnet..\n");
                printf ("----------------------------------------");
          }
        getchar ( );
}
```

EL problema con esta manera de eliminar incluso físicamente los registros del archivo es que no hay manera de recuperar esa información posteriormente. Es por eso que otra técnica común de eliminación es la que se denomina lógica, que consiste en incluir un campo de estado, status o bandera en el registro y conforme se va cargando el registro y antes de mandarlo a disco se le agrega a dicho campo el carácter 'A' -->alta, así que cuando se quiera una baja sólo se pondría dicho campo en 'B' y todos los programas de lectura, búsqueda y filtros deberán revisar esta campo de estado antes de hacer algo con el registro.

MODIFICACIONES EN ARCHIVOS CON C++

La edición o modificación de registros en archivos con c++ es un proceso muy similar al anterior, en donde se copian a un archivo temporal todos los registros que, mientras se realiza la búsqueda, no coinciden con el de la clave. Al encontrar tal registro se leen los nuevos datos y se almacenan en el archivo temporal. Se reanuda la lectura y búsqueda en el archivo original. Al concluir, se cierran los archivos, se borra el original y se renombra el temporal.

LECTURA Y ESCRITURA SOBRE UN ARCHIVO

Para leer y escribir en un archivo en modo texto se usan funciones análogas a las de lectura y escritura de la entrada y salida estándar. La diferencia estriba en que siempre deberemos dar un puntero a FILE para indicar sobre que archivo efectuaremos la operación, ya que podemos tener simultáneamente abiertos varios archivos. Las funciones que trabajar con archivos tienen nombres parecidos a las funciones de entrada y salida estándar, pero comienzan con la letra f. Las más habituales son:

int fprintf (FILE *archivo, const char *formato, ...);
/* trabaja igual que printf ( ) sobre el archivo */
int fscanf (FILE *archivo, const char *formato, ...);
/* trabaja igual que scanf ( ) sobre el archivo */
int fputs (const char *s, FILE *archivo);
/* escribe la cadena s en el archivo */
int fputc (int c, FILE *archivo);
/* escribe el carácter c en el archivo */
int fgetc (FILE *archivo);
/* lee un carácter del archivo */
char *fgets (char *s, int n, FILE * archivo);
/* lee una línea del archivo */

Hay una equivalencia entre las funciones de lectura y escritura estándar y las funciones de lectura y escritura de archivos. Normalmente las funciones de lectura y escritura estándar se definen en la cabecea estándar como macros. As¡ la línea:

```
printf ("hola\n");
```

es equivalente a la escritura en el archivo stdout:

```
fprintf (stdout, "hola\n");
```

EJEMPLO DESGLOSADO

imprimir un archivo de texto usando la impresora

El siguiente programa hace uso de las herramientas para manejo de archivos de la librería estándar de c.

```
/* imprime.c */
#include <conio.h>
#include <stdio.h>

void main ( ) {
FILE *pArchivo, *flujo;              /* Se declaran dos flujos, uno */
                                     /* para el archivo de texto y  */
                                     /* otro para la impresora.     */

char archivo [50], c, puerto [6];

printf ("Escribre el nombre del archivo a imprimir, con extensión *.txt");
gets (archivo);

printf ("¿En que puerto esta conectada la impresora (PRN)(LPT1)(LPT2)?");
gets (puerto);

pArchivo = fopen (archivo, "r"); /* Abrimos el archivo de texto.     */
flujo = fopen (puerto, "w");   /* Abrimos el flujo de la impresora  */

/* Leemos cada carácter del archivo de texto y lo mandamos a imprimir */
        while ((c=getc(pArchivo))!=EOF){   /* EOF implica el final del archivo */
        fprintf (flujo,"%c", c);                  /* Salida a la impresora */
}
```

```
/* Termina la impresión */

getch ( );

/* Cerramos los flujos
fclose (pArchivo);
fclose (flujo);}
```

EJERCICIOS SUGERIDOS

1) Crear una aplicación que maneje un menú para manipular datos de los alumnos de una materia determinada y previamente guardados en un archivo llamado “alumnos.txt”. Debe contener: nombre, calificaciones de tres parciales y número de control.

2) Hacer un programa para el manejo de una agenda telefónica, que contenga nombre y teléfono. Hacer uso de un archivo de texto.

3) Desarrollar una aplicación que copie el contenido de un archivo de texto a otro.

4) Crear una base de datos en un archivo de texto que contenga los resultados de todos los partidos del mundial Alemania 2006. Debe tener los equipos que jugaron y sus resultados de la forma ‘equipo1 – marcador – equipo2’.

5) Crear un programa que busque dentro de un archivo de texto la palabra ‘fichero’ y la cambie por ‘archivo’.

6) Desarrollar una aplicación que utilice un archivo de texto para guardar los datos de una muestra de ‘n’ números dados por el usuario.

7) Crear una aplicación que maneje un directorio. Debe contener: nombre, dirección, teléfono y correo electrónico. Hacer un menú de opciones para consulta y modificación.

8) Hacer un programa que busque direcciones de correo electrónico dentro de un archivo de texto, y las imprima en pantalla.

9) Desarrollar una aplicación para el manejo de un calendario. Debe contener: día, mes, año, eventos del día y un menú de opciones para la consulta y edición del calendario.

10) Crear un programa que modifique un archivo de texto, sustituyendo las palabras que deben de tener acento por las correctas.

11) Hacer una aplicación que lea un formulario resuelto de diez preguntas, el cual debe estar en un archivo de texto. Luego evaluar el formulario con un número entre el 1 y el 10. Las preguntas deben de ser bimodales. (Tip: darle un valor de 1 a las respuestas afirmativas y 0 a las negativas).

12) Desarrollar una aplicación que escriba en un archivo de texto los números del 1 al 1000 y luego sustituya los números pares por su forma 2*n.

13) Hacer lo mismo en el archivo de texto del ejercicio anterior pero sustituyendo los impares por su forma 2*n+1.

14) Usar el archivo de texto del ejercicio anterior y reescribir los números pares e impares en su forma natural ‘n’.

15) Hacer una aplicación que imprima con la impresora los resultados de los programas hechos en los ejercicios anteriores.

GRAFICACIÓN EN TURBO C

FUNCIONES DE GRAFICACIÓN EN TURBO C

arc	bar	bar3d	circle
cleardevice	clearviewport	closegraph	detectgraph
drawpoly	ellipse	fillellipse	fillpoly
floodfill	getarccoords	getaspectratio	getbkcolor
getcolor	getdefaultpalette	getdrivername	getfillpattern
getfillsettings	getgraphmode	getimage	getlinesettings
getmaxcolor	getmaxmode	getmaxx	getmaxy
getmodename	getmoderange	getpalette	getpalettesize
getpixel	gettextsettings	getviewsettings	getx
gety	graphdefaults	grapherrormsg	graphfreemem
graphgetmem	graphresult	imagesize	initgraph
installuserdriver	installuserfont	line	linerel
lineto	moverel	moveto	outtext
outtextxy	pieslice	putimage	putpixel
rectangle	registerbgidriver	registerbgifont	restorecrtmode
sector	setactivepage	setallpalette	setaspectratio
setbkcolor	setfillpattern	setfillstyle	setgraphbufsize
setgraphmode	setlinestyle	setpalette	setrgbpalette
settextjustify	settextstyle	setusercharsize	setviewport
setvisualpage	setwritemode	textheight	textwidth

MACROS

colores	drivers	enlazar	errores
fuentes	linea	modos	put_op
trama			

Función arc

```
void far arc (int x, int y,
  int comienzo_angulo, int final_angulo, int radio);
```

Esta función creará un arco circular. El arco tiene como centro el punto especificado por los argumentos **x** e **y**, y es dibujado con el radio especificado: **radio**. El arco no está rellanado, pero es dibujado usando el color actual. El arco comienza al ángulo especificado por el argumento **comienzo_angulo** y es dibujado en la dirección contraria al de las agujas del reloj hasta llegar al ángulo especificado por el argumento **final_angulo**. La función *arc* usa el este (extendiéndose hacia la derecha del centro del arco en la dirección horizontal) como su punto de 0 grados. La función setlinestyle puede usarse para establecer el grosor del arco. La función *arc*, sin embargo, ignorará el argumento **trama** de la función setlinestyle.

Valor de retorno

La función *arc* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
 int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int radio;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   Registerbgidriver (EGAVGA_driver);
```

```
   Initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   for (radio = 25; radio < 175; radio += 25)
      arc (320, 175, 45, 135, radio);
   getch ( );    /* Pausa */
   closegraph ( );
return 0; }
```

Función bar

```
void far bar(int izquierda, int superior,
   int derecha, int inferior);
```

Esta función dibujará una barra rectangular y rellenada de dos dimensiones. La esquina superior izquierda de la barra rectangular está definida por los argumentos **izquierda** y **superior**. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha de la barra. La barra no tiene borde, pero es rellenada con la trama de relleno actual y el color de relleno como es establecido por la función setlinestyle.

Valor de retorno

La función *bar* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int mai n( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y, color, relleno;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   x = 20;
   y = 20;
   color = 1;
   fill = 1;
   do {
      setfillstyle (fill, color);
      bar (x, y, x+40, 320);
      x += 40;
      y += 10;
      color = (color+1) % 16;
      fill = (fill+1) % 12;
   } while (x < 620);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función bar3d

```
void far bar3d (int izquierda, int superior,
  int derecha, int inferior, int profundidad, int banderin_tapa);
```

Esta función creará una barra rectangular y rellenada de tres dimensiones. La esquina superior izquierda de la barra rectangular más frontal está definida por el argumento izquierdo y superior. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda del rectángulo más frontal. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha del rectángulo más frontal. La barra tiene borde, en todas las tres dimensiones, rellenada con el color y estilo de línea actuales. El rectángulo más frontal es rellenado usando la trama de relleno actual y el color de relleno como es establecido por la función setlinestyle. El argumento **banderin_tapa** es usado para especificar si es o no es posible apilar varias barras encima de cada una. Si **banderin_tapa** tiene un valor distinto a cero, entonces la barra está "tapada". Si **banderin_tapa** tiene un valor de cero, entonces la barra no está "tapada", permitiendo otras barras ser apiladas encima de ésta.

Valor de retorno

La función *bar3d* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  int color, relleno;
```

```
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   color = 10;
   relleno = 11;
   setfillstyle (relleno, color);
   bar3d (100, 50, 300, 150, 25, 1);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función circle

```
void far circle (int x, int y, int radio);
```

Esta función se usas para dibujar un círculo. Los argumentos **x** e **y** definen el centro del círculo, mientras que el argumento **radio** define el radio del círculo. El círculo no es rellenado pero es dibujado usando el color actual. El grosor de la circunferencia puede ser establecido por la función setlinestyle; sin embargo, el estilo de la línea es ignorado por la función *circle*. La proporción anchura-altura para el modo actual es considerado cuando se calcula el círculo. Por esta razón, alterando los valores de defecto x e y de los factores de anchura-altura afectará el círculo (ya no seguirá siendo redondo).

Valor de retorno

La función *circle* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int relleno;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   relleno = 1;
   setlinestyle (SOLID_LINE, relleno, THICK_WIDTH);
   circle (300, 200, 80);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función cleardevice

```
void far cleardevice (void);
```

Esta función es usada para despejar una pantalla gráfica. La función *cleardevice* usa el color de fondo actual, como es establecido por la función setbkcolor, para rellenar la pantalla. La posición del cursor gráfico es la esquina superior izquierda de la pantalla - posición (0,0) - después de que la pantalla haya sido borrado.

Valor de retorno

La función *cleardevice* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int relleno, color;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
  initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   relleno = 1;
   color = 1;

   setlinestyle (SOLID_LINE, relleno, THICK_WIDTH);
   circle (300, 200, 80);
   getch ( );    /* Pausa */
   setbkcolor (color);
   cleardevice ( );
```

```
   setlinestyle (SOLID_LINE, relleno, THICK_WIDTH);
   circle (400, 200, 20);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función clearviewport

```
void far clearviewport (void);
```

Esta función es usada para rellenar la pantalla actual del usuario con el color de fondo actual. El color de fondo puede ser establecido con la función setbkcolor. La posición del cursor gráfico es la esquina superior izquierda de la pantalla actual del usuario. Esta posición es (0,0) según la pantalla actual del usuario.

Valor de retorno

La función *clearviewport* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
```

```
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setviewport (150, 150, 350, 350, 0);
   for (color = 0; color<16; color++) {
      circle(100, 100, 60);
      getch ( );
      setbkcolor (color);
      clearviewport  ( );
   }
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función closegraph

```
void far closegraph (void);
```

Esta función es usada para cerrar el sistema gráfico como es iniciada por la función initgraph. La función *closegraph* libera toda la memoria usada por el sistema gráfico y luego restaura el modo de vídeo al modo de texto que estaba en uso anteriormente a la llamada a la función initgraph.

Valor de retorno

La función *closegraph* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   circle (300, 200, 80);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función detectgraph

```
void far detectgraph (int far *driver, int far *modo);
```

Esta función es usada para detectar el adaptador gráfico y el modo óptimo para usar con el sistema en uso. Si la función *detectgraph* no puede detectar ningún dispositivo gráfico, el argumento ***driver** es asignado grNotDetected (-2). Una llamada a graphresult resultará en un valor de retorno de -2, o grNotDetected.

Existen varios valores que indican los diferentes dispositivos gráficos que pueden ser usados por el argumento ***driver**. Un valor de 0, o DETECT, inicia la funcionalidad de autodetección, el cual determina el driver óptimo a usar.

Para cada dispositivo existen varios valores que indican los diferentes modos gráficos que pueden ser usados por el argumento ***modo**. Sin embargo, si el argumento ***driver** es asignado el valor de 0, o DETECT, el argumento ***modo** es automáticamente establecido al modo de resolución mas alto para el driver.

Valor de retorno

La función *detectgraph* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver, gmodo;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   detectgraph (&gdriver, &gmodo,);
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   circle (300, 200, 80);
   getch ( );    /* Pausa */
   closegraph ( );
   printf ("Driver: %d\tModo: %d\n\n", gdriver, gmodo);
   return 0;
```

```
}
```

Función drawpoly

```
void far drawpoly (int numpuntos, int far *puntos);
```

Esta función es usada para crear un polígono con un número especificado de puntos. El argumento **numpuntos** es usado para definir el número de puntos en el polígono. Para la función *drawpoly*, el número de puntos debe ser el número actual de puntos más 1 para poder crear un polígono cerrado. En otras palabras, el primer punto debe ser igual al último punto. El argumento ***puntos** apunta a un array de números de longitud **numpuntos** multiplicado por 2. Los dos primeros miembros del array identifica las coordenadas *x* e *y* del primer punto, respectivamente, mientras que los dos siguientes especifican el siguiente punto, y así sucesivamente. La función *drawpoly* dibuja el perímetro del polígono con el estilo de línea y color actuales, pero no rellena el polígono.

Valor de retorno

La función *drawpoly* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int puntos [8] = { 300, 50, 500, 300, 100, 300, 300, 50 };
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   drawpoly (4, puntos);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función ellipse

```
void far ellipse (int x, int y, int comienzo_angulo,
   int final_angulo, int x_radio, int y_radio);
```

Esta función es usada para dibujar un arco elíptico en el color actual. El arco elíptico está centrado en el punto especificado por los argumentos **x** e **y**. Ya que el arco es eilptíco el argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical. El arco elíptico comienza con el ángulo

especificado por el argumento **comienzo_angulo** y se extiende en un sentido contrario a las agujas del reloj al ángulo especificado por el argumento **final_angulo**. La función *ellipse* considera este - el eje horizontal a la derecha del centro del elipse - ser 0 grados. El arco elíptico es dibujado con el grosor de línea actual como es establecido por la función setlinestyle. Sin embargo, el estilo de línea es ignorado por la función *ellipse*.

Valor de retorno

La función *ellipse* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   ellipse (300, 150, 45, 225, 100, 50);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función fillellipse

```
void far fillellipse (int x, int y,
   int x_radio, int y_radio);
```

Esta función es usada para dibujar y rellenar una elipse. El centro de la elipse es especificado por los argumentos **x** e **y**. El argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical de la elipse. La elipse es dibujado con el perímetro en el color actual y rellenada con el color de relleno y la trama de relleno actuales.

Valor de retorno

La función *fillellipse* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
```

```
   trama = SOLID_FILL;
   color = 4;
   setfillstyle (trama, color);
   fillellipse (300, 150, 100, 50);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función fillpoly

```
void far fillpoly (int numpuntos, int far *puntos);
```

Esta función es usada para crear un polígono rellenado. El argumento **numpuntos** es usado para definir el número de puntos en el polígono. Al contrario que la función drawpoly, la función automáticamente cierra el polígono. El argumento ***puntos** apunta a un array de números de longitud **numpuntos** multiplicado por 2. Los dos primeros miembros del array identifica las coordenadas *x* e *y* del primer punto, respectivamente, mientras que los dos siguientes especifican el siguiente punto, y así sucesivamente. La función *fillpoly* dibuja el perímetro del polígono con el estilo de línea y color actuales. Luego, el polígono es rellenado con la trama de relleno y color de relleno actuales.

Valor de retorno

La función *fillpoly* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
```

```
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;
   int puntos [6] = { 300, 50, 500, 300, 100, 300 };
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   trama = SLASH_FILL;
   color = 4;
   setfillstyle (trama, color);
   fillpoly (3, puntos);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función floodfill

```
void far floodfill (int x, int y, int borde);
```

Esta función es usada para rellenar un área cerrado con el color de relleno y trama de relleno actuales. Los argumentos **x** e **y** especifican el punto de comienzo para el algoritmo de relleno. El argumento **borde** especifica el valor del color del borde del área. Para que la función *fillpoly* funcione como es esperado, el área a ser

rellenado debe estar rodeada por el color especificado por el argumento **borde.** Cuando el punto especificado por los argumentos **x** e **y** se encuentra dentro del área a ser rellenada, el interior será rellenado. Si se encuentra fuera del área, el exterior será rellenada.

Nota: Esta función no funciona con el driver IBM-8514.

Valor de retorno

La función *floodfill* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;
   int puntos [8] = { 300, 50, 500, 300, 100, 300, 300, 50 };
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setcolor (10);
   drawpoly (4, puntos);
   trama = SLASH_FILL;
```

```
   color = 4;
   setfillstyle (trama, color);
   floodfill (400, 250, 10);
   getch ( );    /* Pausa */
   closegrap h ( );
   return 0;
}
```

Función getarccoords

```
void far getarccoords (struct arccoordstype far *coordenadas_arco);
```

Esta función es usada para recoger las coordenadas del centro, y los puntos del comienzo y final de la última llamada con éxito a la función arc. El argumento ***coordenadas_arco** apunta a la estructura de tipo **arccoordstype** que guarda la información recogida. La síntaxis de la estructura arccoordstype es:

```
struct arccoordstype {
   int x, y;
   int xstart, ystart;
   int xend, yend;
};
```

Los miembros **x** e **y** definen el centro del arco. Los miembros **xstart** e **ystart** definen las coordenadas *x* e *y* del punto de comienzo del arco. Similarmente, los miembros **xend** e **yend** definen las coordenadas *x* e *y* del punto de final del arco.

Valor de retorno

La función *getarccoords* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int radio;
   struct arccoordstype info_arco;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   for (radio=25; radio<=100; radio+=25) {
      arc (300, 150, 45, 315, radio);
      getarccoords (&info_arco);
      moveto (info_arco.xstart, info_arco.ystart);
      lineto (info_arco.xend, info_arco.yend);
   }
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función getaspectratio

```
void far getaspectratio (int far *x_proporcion,
  int far *y_proporcion);
```

Esta función es usada para obtener la proporción anchura-altura del modo gráfico actual. La proporción anchura-altura puede definirse como la proporción de la anchura del píxel del modo gráfico y la altura del píxel. Esta proporción, usando los modos gráficos existentes, es siempre menor o igual que 1. El valor para determinar la proporción anchura-altura con respecto al eje horizontal es retornado en el argumento ***x_proporcion**. Similarmente, el valor para el eje vertical es retornado en el argumento ***y_proporcion**. El argumento ***y_proporcion** es asignado 10000, el cual es retornado cuando se llama a la función *getaspectratio*. El argumento ***x_proporcion** es casi siempre menor que el valor de ***y_proporcion**. Esto es debido al hecho de que la mayoría de los modos gráficos tiene píxels más altos que anchos. La única excepción es en los modos de VGA que produce píxels cuadrados; es decir, **x_proporcion** = **y_proporcion**.

Valor de retorno

La función *getaspectratio* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  int x_proporcion, y_proporcion;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
```

```
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getaspectratio (&x_proporcion, &y_proporcion);
   circle (300, 150, 50);
   getch ( );    /* Pausa */
   closegraph ( );
   printf ("Proporción anchura-altura.\nFactor x: %d\tFactor y: %d\n",
      x_proporcion, y_proporcion);
   return 0;
}
```

Función getbkcolor

```
int far getbkcolor (void);
```

Esta función es usada para obtener el valor del color de fondo actual. El color de fondo, por defecto, es el color 0. Sin embargo, este valor puede cambiar con una llamada a la función setbkcolor.

Existen varios valores para ciertos colores de fondo.

Valor de retorno

La función *getbkcolor* retorna el valor del color de fondo actual.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setbkcolor (4);
   circle (300, 150, 50);
   color = getbkcolor ( );
   getch();    /* Pausa */
   closegraph ( );
   printf ("Color de fondo: %d\n", color);
   return 0; }
```

Función getcolor

```
int far getcolor (void);
```

Esta función obtiene el valor del color actual. El color actual es el color usado para dibujar líneas, arcos, etc. Este color no es el mismo que el color de relleno. El valor del color obtenido es interpretado según el modo que esté en uso.

Existen varios valores para ciertos colores de fondo.

Valor de retorno

La función *getcolor* retorna el valor del color actual.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");

   setcolor (4);
   circle (300, 150, 50);
   color = getcolor ( );
   getch ( );    /* Pausa */
   closegraph ( );
   printf ("Color del perímetro: %d\n", color);
   return 0; }
```

Función getdefaultpalette

```
struct palettetype far *getdefaultpalette (void);
```

Esta función es usada para obtener una estructura que define la paleta según el dispositivo en la inicialización - esto es, cuando se llama a initgraph. La estructura palettetype se define de la siguiente manera:

```
#define MAXCOLORS 15
struct palettetype {
  unsigned char size;
  signed char colors [MAXCOLORS+1];
}
```

El campo **size** indica el tamaño de la paleta. El campo **colors** contiene los valores numéricos que representan los colores que ofrece el dispositivo en su paleta de colores.

La función *getdefaultpalette* retorna un puntero a una estructura del tipo **palettetype**.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h&t;
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  struct palettetype *palette = NULL;
```

```
   int i;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   palette = getpalettetype ( );
   circle (300, 150, 50);
   getch ( );    /* Pausa */
   closegraph ( );
   printf ("Paleta\n\nTamaño: %d\nColores: %d",
      palette->size, palette->colors[0]);
   for (i=1; i<palette->size; i++)
      printf (", %d", palette->colors[i]);
   printf ("\n");
   return 0;
}
```

Función getdrivername

```
char *far getdrivername (void);
```

Esta función es usada para obtener una cadena de caracteres que contiene el nombre del dispositivo gráfico actual. Este función debería ser llamada después de que un dispositivo haya sido definido e inicializado - esto es, después de llamar a initgraph.

Valor de retorno

La función *getdrivername* retorna una cadena de caracteres conteniendo el nombre del dispositivo gráfico.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
 int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char *nombre;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   strcpy (nombre, getdrivername ( ) );
   circle (300, 150, 50);
   getch ( );    /* Pausa */
   closegraph ( );
   printf ("Nombre del dispositivo gráfico: %s\n", nombre);
   return 0;
}
```

Función getfillpattern

```
void far getfillpattern (char far *trama);
```

Esta función es usada para obtener una trama de relleno definido por el usuario, como es definida por la función setfillpattern, y la guarda en memoria. El argumento ***trama** es un puntero a una serie de ocho bytes que representa una trama de relleno de bits de 8 x 8. Cada byte representa una fila de ocho bits, donde cada bit está encendido o no (1 ó 0). Un bit de 0 indica que el píxel correspondiente será asignado el color de relleno actual. Un bit de 0 indica que el píxel correspondiente no será alterado.

Valor de retorno

La función *getfillpattern* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char trama 1 [8] = { 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE };
   char trama 2 [8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
```

```
  Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
  getfillpattern (trama 2);
  bar (50, 50, 150, 150);
  setfillpattern (trama 1, 9);
  bar (160, 50, 260, 150);
  setfillpattern (trama 2, 4);
  bar (105, 160, 205, 260);
  getch ( );    /* Pausa */
  closegraph ( );
  return 0;
}
```

Función getfillsettings

```
void far getfillsettings (struct fillsettingstype far *info);
```

Esta función es usada para obtener la información de tramas de relleno. El argumento ***info** apunta a una estructura de tipo fillsettingstype, el cual es actualizado cuando se llama a la función *getfillsettings*. La estructura es:

```
struct fillsettingstype {
  int pattern;
  int color;
};
```

El campo **pattern** es la trama y el campo **color** es el color de relleno de la trama.

Existen trece valores ya definidos para tramas.

Valor de retorno

La función *getfillsettings* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct fillsettingstype info;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getfillsettings (&info);
   bar (50, 50, 350, 300);
   getch  ();    /* Pausa */
   closegrap h( );
   printf ("Trama de relleno: %d\tColor de relleno: %d\n",
      info.pattern, info.color);
   return 0;
}
```

Función getgraphmode

```
int far getgraphmode (void);
```

Esta función es usada para obtener el valor del modo gráfico actual. El dispositivo actual debe ser considerado cuando se interprete el valor de retorno. Esta función sólo debería ser llamada después de que el sistema gráfico haya sido inicialiazado con la función initgraph.

Existen varios valores para los modos de cada dispositivo.

Valor de retorno

La función *getgraphmode* retorna el modo gráfico como es establecido por initgraph o setgraphmode.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int modo;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
```

```
  Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
  modo = getgraphmode ( );
  bar (50, 50, 350, 300);
  getch ( );    /* Pausa */
 closegraph ( );
  printf ("Modo gráfico: %d\n", modo);
  return 0;
}
```

Función getimage

```
void far getimage (int izquierda, int superior,
  int derecha, int inferior, void far *imagen);
```

Esta función es usada para guardar una porción rectangular de la pantalla para un uso posterior. La esquina superior izquierda del área rectangular que ha de ser guardada está definida por el argumento izquierdo y superior. Estos argumentos representan las coordenades *x* e *y* de la esquina superior izquierda, respectivamente. Los argumentos **derecha** e **inferior** definen la esquina inferior derecha de la imagen rectangular. Estos argumentos definen las coordenades *x* e *y* de la esquina inferior derecha, respectivamente. El argumento ***image** apunta al búfer de memoria donde la imagen está guardada.

Valor de retorno

La función *getimage* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
```

```
#include <stdlib.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   void *imagen;
   int imagentam;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   Setfillstyle (SLASH_FILL, 5);
   bar (50, 50, 350, 300);
   imagentam = imagesize (50, 50, 100, 100);
   imagen = malloc (imagentam);
   getimage (50, 50, 100, 100, imagen);
   putimage (400, 50, imagen, COPY_PUT);
   putimage (400, 110, imagen, COPY_PUT);
   getch ( );    /* Pausa */
   closegraph ( );
   free (imagen);
   return 0;
}
```

Función getlinesettings

```
void far getlinesettings (struct linesettingstype far *info);
```

Esta función obtiene la información actual para las líneas. Esta información es guardada en una estructura de tipo linesettingstype que es apuntado por el argumento ***info**. El estilo de línea, trama, y grosor actuales son guardados en esta estructura. La síntaxis para la estructura **linesettingstype**.

```
struct linesettingstype {
   int linestyle;
   unsigned upattern;
   int thickness;
}
```

El campo **linestyle** es el estilo de la línea recta. El campo **upattern** es la trama de la línea del usuario solamente cuando el campo **linestyle** es igual a **USERBIT_LINE**, ó 4. Cuando esto sea el caso, el miembro **upattern** contiene una trama de línea definido por el usuario de 16 bits. Un bit 1 en esta trama indica que el píxel correspondiente será asignado el color actual. Un bit 0 indica que el píxel correspondiente no será alterado. El campo **thickness** es el grosor de la línea.
Existen varios valores para los diferentes estilos y grosores de líneas rectas.

Valor de retorno

La función *getlinesettings* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
```

```
   struct linesettingstype info;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setlinestyle (DOTTED_LINE, 0xFF33, THICK_WIDTH);
   circle (350, 250, 50);
   getlinesettings (&info);
   getch ( );    /* Pausa */
   closegraph ( );
   printf ("Líneas rectas.\nEstilo: %d\tTrama: %X\tGrosor: %d\n",
      info.linestyle, info.upattern, info.thickness );
   return 0;
}
```

Función getmaxcolor

```
int far getmaxcolor (void);
```

Esta función es usada para obtener el valor más alto de color en la paleta actual. La paleta en uso depende del dispositivo y modo inicializados. Para los modos de 16 colores, el valor de retorno es 15. Similarmente, para los modos de dos colores, el valor de retorno es 1.

Valor de retorno

La función *getmaxcolor* retorna el valor máximo del color en la paleta en uso.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color_max;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   color_max = getmaxcolor ( );
   closegraph ( );

   printf ("Color máximo: %d\n", color_max);
   return 0;
}
```

Función getmaxmode

```
int far getmaxmode (void);
```

Esta función es usada para determinar el valor más alto del modo gráfico para el dispositivo actual. El mayor valor del modo generalmente representa el modo con la resolución más alta.

Valor de retorno

La función *getmaxmode* retorna el valor máximo del modo gráfico para el dispositivo actual.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int modo_max;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   modo_max = getmaxmode ( );
   closegraph ( );
   printf ("Modo máximo: %d\n", modo_max);
   return 0;
}
```

Función getmaxx

```
int far getmaxx (void);
```

Esta función es usada para obtener la coordenada máxima de la pantalla en la dirección horizontal. Este valor suele ser la resolución horizontal máxima menos 1.

Valor de retorno

La función *getmaxx* retorna la coordenada máxima de la pantalla en la dirección horizontal.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x_max;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   x_max = getmaxx ( );
   closegraph ();
   printf ("X máxima: %d\n", x_max);
   return 0;
}
```

Función getmaxy

```
int far getmaxy (void);
```

Esta función es usada para obtener la coordenada máxima de la pantalla en la dirección vertictal. Este valor suele ser la resolución vertictal máxima menos 1.

Valor de retorno

La función *getmaxy* retorna la coordenada máxima de la pantalla en la dirección vertictal.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x_max, y_max;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   x_max = getmaxx ( );
   y_max = getmaxy ( );
   closegraph ( );
```

```
   printf ("X máxima: %d\tY máxima: %d\n", x_max, y_max);
   return 0;
}
```

Función getmodename

```
char *far getmodename (int num_modo);
```

Esta función es usada para obtener el nombre del modo gráfico especificado por el argumento **num_modo**.

Valor de retorno

La función *getmodename* retorna el nombre del modo gráfico que está contenido en todos los dispositivos gráficos.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char *nombre;
   int num_modo;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
```

```
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   num_modo = getgraphmode ( );
   strcpy (nombre, getmodename (num_modo ) );
   closegraph ( );
   printf ("X máxima: %d\tY máxima: %d\n", x_max, y_max);
   return 0; }
```

Función getmoderange

```
void far getmoderange (int driver,
   int far *modo_bajo, int far *modo_alto);
```

Esta función es usada para obtener los valores altos y bajos del modo gráfico del dispositivo especificado por el argumento **driver**. El valor más bajo del modo es retornado en ***modo_bajo**, y el valor más alto del modo es retornado en ***modo_alto**. Si el dispositivo especificado es inválido, el valor de -1 es retornado en ambos argumentos: ***modo_bajo** y ***modo_alto**. Sin embargo, si el argumento **driver** es asignado -1, los modos alto y bajo del dispositivo actual son retornados.

Valor de retorno

La función *getmoderange* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
```

```
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int modo_bajo, modo_alto;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
  registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getmoderange (gdriver, &modo_bajo, &modo_alto);
   closegraph ( );
   printf ("Alcance de modos, de %d á %d\n", modo_bajo, modo_alto);
   return 0; }
```

Función getpalette

```
void far getpalette (struct palettetype far *paleta);
```

Esta función es usada para obtener la información de la paleta actual. El argumento ***paleta** apunta a una estructura del tipo palettetype donde la información de la paleta es guardada. La estructura **palettetype** se define de la siguiente manera:

```
#define MAXCOLORS 15
struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS+1];
}
```

El campo **size** indica el número de colores en la paleta. El campo **colors** contiene los valores numéricos que representan los colores que ofrece el dispositivo en su paleta de colores.

Valor de retorno

La función *getpalette* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int i;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getpalette (&palette);
   closegraph ( );
   printf ("Paleta\n\nTamaño: %d\nColores: %d", palette.size, palette.colors[0]);
   for (i=1; i<palette.size; i++)
      printf (", %d", palette.colors[i]);
   printf ("\n");
```

```
   return 0;
}
```

Función getpalettesize

```
int far getpalettesize (void);
```

Esta función es usada para obtener el número de entradas de paleta válidas para la paleta actual, considerando el modo gráfico en uso.

Valor de retorno

La función *getpalettesize* retorna el número de colores en la paleta actual. Para modos de 16 colores, la función *getpalettesize* retorna 16.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int num_colores;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
```

```
   num_colores = getpalettesize ( );
   closegraph ( );
   printf ("Paleta\n\nNúmero de colores: %d\n", num_colores);
   return 0;
}
```

Función getpixel

```
unsigned far getpixel (int x, int y);
```

Esta función es usada para obtener el valor del color del píxel especificado por los argumentos **x** e **y**. Estos argumentos especifican las coordenadas de la pantalla del píxel a ser evaluado. Cuando se evalúa el valor del color retornado, el modo gráfico en uso debe ser considerado.

Existen varios valores para describir colores.

Valor de retorno

La función *getpixel* retorna el número del color del píxel especificado.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y, color;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
```

```
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   x = 300;
   y = 100;
   setfillstyle (SOLID_FILL, 2);
   fillellipse (300, 160, 50, 150);
   color = getpixel ( x, y );
   getch ( );
   closegraph ( );
   printf ("Colores\n\nEl color del píxel (%d,%d): %d\n", x, y, color);
   return 0;
```

Función gettextsettings

```
void far gettextsettings (struct textsettingstype far *info);
```

Esta función es usada para obtener información acerca de la fuente gráfica actual. Esta información es guardada en una estructura de tipo textsettingstype, la cual es apuntada por el argumento ***info**. Esta estructura contiene información de la fuente actual en uso, la orientación del texto, el tamaño del carácter, y la justificación horizontal y vertical. La síntaxis de la estructura **textsettingstype** es la siguiente:

```
struct textsettingstype {
   int font;
   int direction;
   int charsize;
   int horiz;
   int vert;
```

```
};
```

Existen varios valores para describir el tipo, la orientación, y justificación de fuentes.

Valor de retorno

La función *gettextsettings* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct textsettingstype info;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   gettextsettings (&info);
   closegraph ( );
   printf ("Texto\n\nFuente: %d\tSentido: %d\tTamaño: %d\n"
           "Justificación:\nHorizontal: %d, Vertical: %d\n",
           info.font, info.direction, info.charsize, info.horiz, info.vert);
   return 0;
}
```

Función getviewsettings

```
void far getviewsettings (struct viewporttype far *info);
```

Esta función es usada para obtener información acerca del área gráfica actual. Esta información es guardada en una estructura de tipo viewporttype, la cual es apuntada por el argumento ***info**. Esta estructura contiene información acerca de las esquinas superior izquierda e inferior derecha, también como el banderín de recorte del área gráfica. La síntaxis de la estructura **viewporttype** es la siguiente:

```
struct viewporttype {
   int left, top;
   int right, bottom;
   int clip;
};
```

Valor de retorno

La función *getviewsettings* no retorna ningún valor, directamente.

EJEMPLO

```
#include <graphics.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct viewporttype info;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
```

```
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getviewsettings (&info);
   closegraph ( );
   printf ("Pantalla\n\nIzquierda: %d\tSuperior: %d\tDerecha: %d\t"
             "Inferior: %d\tBanderín: %d\n",
          info.left, info.top, info.right, info.bottom, info.clip);
   return 0; }
```

Función getx

```
int far getx (void);
```

Esta función es usada para obtener la posición, en la dirección horizontal, del cursor gráfico. El valor retornado especifica el lugar del píxel horizontal del cursor gráfico (la coordenada *x*), relativo a la pantalla del usuario actual.

Valor de retorno

La función *getx* retorna la coordenada *x* del cursor gráfico.

EJEMPLO

```
#include <graphics.h>
#include <stdio.h>
int main ( ) {
```

```
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   moveto (300, 150);
   x = getx ( );
   y = gety ( );
   closegraph ( );
   printf ("Cursor gráfico\n\nX: %d\tY: %d\n", x, y);
   return 0;
}
```

Función gety

```
int far gety (void);
```

Esta función es usada para obtener la posición, en la dirección vertical, del cursor gráfico. El valor retornado especifica el lugar del píxel vertical del cursor gráfico (la coordenada *y*), relativo a la pantalla del usuario actual.

Valor de retorno

La función *gety* retorna la coordenada *y* del cursor gráfico.

EJEMPLO

```
#include <graphics.h>
#include <stdio.h>
int main  () {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   moveto (300, 150);
   x = getx ( );
   y = gety ( );
   closegraph ( );
   printf ("Cursor gráfico\n\nX: %d\tY: %d\n", x, y);
   return 0;
}
```

Función graphdefaults

```
void far graphdefaults (void);
```

Esta función es usada para reiniciar todos los datos gráficos a sus valores originales, o por defecto. La función *graphdefaults* reinicia la pantalla del usuario para que cubra la pantalla entera, mueve el cursor a la posición (0, 0), y reinicia la paleta actual a sus colores por defecto. También reinicia el color de fondo y el

actual a sus valores por defecto, reinicia el estilo y trama de relleno a sus valores por defecto, y reinicia la fuente y justificación de texto.

Valor de retorno

La función *graphdefaults* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setcolor (4);
   setviewport (250, 150, 350, 250, 1);
   graphdefaults ( );
   circle (300, 200, 50);
   getch ( );
   closegraph ( );
  return 0; }
```

Función grapherrormsg

```
char *far grapherrormsg (int codigo_error);
```

Esta función es usada para obtener una cadena de caracteres conteniendo el mensaje de error para un código de error especificado. El argumento **codigo_error** especifica el valor del código de error. La función graphresult debe ser usada para obtener el código de error usado para el argumento **codigo_error**.

Valor de retorno

La función *grapherrormsg* retorna una cadena de caracteres.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int codigo_error;
   char *mensaje_error;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver ( EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setgraphmode (40);    /* Creamos un ERROR */
   codigo_error = graphresult ( );
```

```
  strcpy (mensaje_error, grapherrormsg (codigo_error) );
  closegraph ( );
  printf ("ERROR: \"%s\" (%d)\n", mensaje_error, codigo_error);
return 0; }
```

Función _graphfreemem

```
void far _graphfreemem (void far *ptr, unsigned tamanyo);
```

Esta función es usada por la librería gráfica para desadjudicar memoria previamente reservada mediante una llamada a la función _graphgetmem. Esta función es llamada por la librería gráfica cuando se quiere liberar memoria. Por defecto, la función simplemente llama a *free*, pero se puede controlar la administración de memoria de la librería gráfica. La forma de hacer esto es simplemente creando la definición de la función, con el mismo prototipo mostrado aquí.

Valor de retorno

La función *_graphfreemem* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
void far _graphfreemem (void far *ptr, unsigned tamanyo) {
  printf ("__graphfreemem ha sido llamado para "
    "desadjudicar %d bytes en memoria\n");
  printf ("para el montón (heap) interno\n", tamanyo);
```

```
   printf ("Pulse cualquier tecla...\n\n" );
   getch ( );
   free (ptr);
}
void far * far _graphgetmem (unsigned tamanyo) {
   printf ("__graphgetmem ha sido llamado para "
      "adjudicar %d bytes en memoria\n");
   printf ("para el montón (heap) interno\n", tamanyo);
   printf ("Pulse cualquier tecla...\n\n");
   getch ( );
   return malloc (tamanyo);
}
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   circle (200, 100, 50);
   getch ( );
   closegraph ( );
   return 0; }
```

Función _graphgetmem

```
void far * far _graphgetmem (unsigned tamanyo);
```

Esta función es usada por la librería gráfica para adjudicar memoria gráfica para búfers internos, dispositivos gráficos, y fuentes. Esta función tiene como intención ser llamada por la librería gráfica cuando se quiere adjudicar memoria. Por defecto, la función simplemente llama a *malloc*, pero se puede controlar la administración de memoria de la librería gráfica. La forma de hacer esto es simplemente creando la definición de la función, con el mismo prototipo mostrado aquí.

Valor de retorno

La función ***_graphgetmem*** no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
void far _graphfreemem (void far *ptr, unsigned tamanyo) {
   printf ("__graphfreemem ha sido llamado para "
      "desadjudicar %d bytes en memoria\n");
   printf ("para el montón (heap) interno\n", tamanyo);
   printf ("Pulse cualquier tecla...\n\n");
   getch ( );
   free (ptr);
}
void far * far _graphgetmem (unsigned tamanyo) {
   printf ("__graphgetmem ha sido llamado para "
      "adjudicar %d bytes en memoria\n");
   printf ("para el montón (heap) interno\n", tamanyo);
   printf ("Pulse cualquier tecla...\n\n");
   getch ( );
   return malloc (tamanyo);
}
```

```
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   circle (200, 100, 50);
   getch ( );
   closegraph ( );
   return 0;
}
```

Función graphresult

```
int far graphresult (void);
```

Esta función obtiene y retorna el código de error para la última llamada sin éxito. Además, reinicia el nivel de error a 0, o **grOk**.

Existen varios valores de códigos de error.

Valor de retorno

La función *graphresult* retorna el código de error de la última llamada gráfica sin éxito.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int codigo_error;
   char *mensaje_error;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setgraphmode (40);    /* Creamos un ERROR */
   codigo_error = graphresult ( );
   strcpy (mensaje_error, grapherrormsg (codigo_error ) );
   closegraph ( );
   printf ("ERROR: \"%s\" (%d)\n", mensaje_error, codigo_error);
   return 0;
}
```

Función imagesize

```
unsigned far imagesize (int izquierda, int superior,
   int derecha, int inferior);
```

Esta función es usada para determinar el tamaño del búfer necesitado para almacenar una imagen con la función getimage. Los argumentos **izquierda** y

superior definen las coordenadas *x* e *y* de la esquina superior izquierda de la imagen rectangular. Similarmente, los argumentos **derecha** y **inferior** definen las coordenadas *x* e *y* de la esquina inferior derecha de la imagen rectangular.

Valor de retorno

La función *imagesize* retorna el número actual de bytes necesarios si el tamaño requerido es menor que 64 Kb menos 1 byte. Si esto no es el caso, el valor retornado es 0xFFFF, ó -1.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   void *imagen;
   int imagentam;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setfillstyle (SLASH_FILL, 5);
   bar (50, 50, 350, 300);
   imagentam = imagesize (50, 50, 100, 100);
   imagen = malloc (imagentam);
   getimage (50, 50, 100, 100, imagen);
```

```
  putimage (400, 50, imagen, COPY_PUT);
  putimage (400, 110, imagen, COPY_PUT);
  getch ( );    /* Pausa */
  closegraph ( );
free (imagen); return 0;}
```

Función initgraph

```
void far initgraph (int far *driver,
  int far *modo, int far *path);
```

Esta función es usada para cargar o validar un dispositivo gráfico y cambiar el sistema de vídeo a modo gráfico. La función *initgraph* debe ser llamada antes de cualesquier funciones que generan una salida gráfica sean usadas.

Existen varios valores a ser usados para el argumento ***driver**. Si ***driver** es asignado a DETECT, ó 0, la función detectgraph es llamada, y un dispositivo y modo gráfico apropiados son seleccionados. Asignando a ***driver** cualquier otro valor predefinido inicia la carga del dispositivo gráfico correspondiente.

Existen varios valores a ser usados para el argumento ***modo**. Estos valores deberían corresponder al dispositivo especificado en el argumento ***driver**.

El argumento ***path** especificad el directorio donde los dispositivos gráficos están localizados. La función *initgraph* buscará el dispositivo primeramente en este directorio. Si no es encontrado, la función buscará en el directorio de inicio. Cuando el argumento ***path** es NULL, solamente el directorio de inicio es buscado. Otra forma para evitar cargando el dispositivo desde el disco cada vez que el programa es ejecutado es ligarlo o enlazarlo al dispositivo apropiado en un programa ejecutable.

Valor de retorno

La función *initgraph* no retorna ningún valor. Sin embargo, cuando la función *initgraph* es llamada, el código de error interno es activado. Si la función *initgraph* termina con éxito, el código es asignado un 0. Si no, el código es asignado así:

-2 grNotDetected La tarjeta gráfica no se encontró

-3 grFileNotFound El fichero del dispositivo no se encontró

-4 grInvalidDriver El fichero del dispositivo es inválido

-5 grNoLoadMem No hay suficiente memoria para cargar el dispositivo

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   circle (300, 200, 80);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función installuserdriver

```
int far installuserdriver (char far *nombre,
  int huge (*detectar) (void));
```

Esta función permite al usuario añadir dispositivos adicionales de otras compañías o grupos a la tabla interna BGI de los dispositivos. El argumento ***nombre** define el nombre del fichero nuevo del dispositivo .BGI. El parámetro ***detectar** es un puntero a una función opcional para autodetectar que puede ser o no ser provisto con el dispositivo nuevo. La función de autodetectación espera no recibir ningún parámetro y retorna un valor entero.

Valor de retorno

La función *installuserdriver* retorna el parámetro del número del dispositivo que hubiese sido pasado a la función initgraph para seleccionar un dispositivo nuevo.

EJEMPLO

```
/* Este programa no funcionará, ya que se
** necesitaría otra tarjeta gráfica
** desconocida por las librerías gráficas de BGI.
** Esto sólo es para poner un ejemplo.
*/
#include <graphics.h>
int huge detectarSMGGA (void) {
  int driver, modo, modo_sugerirdo=0;
  detectgraph (&driver, &modo);
  if (SMGGA == driver)   return modo_sugerido;
  return grError;
}
int main ( ) {
```

```
   int gdriver, gmodo;
/* Intentamos instalar nuestra tarjeta gráfica:
** Súper Mega Guay Graphics Array (SMGGA)
** Ya sé que suena muy cursi, pero esto sólo es un ejemplo:)
*/
   gdriver = installuserdriver ("SMGGA", detectarSMGGA);
/* Forzamos a que use nuestra función para autodetectar */
   gdriver = DETECT;
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   closegraph ( );
   return 0;
}
```

Función installuserfont

```
int far installuserfont (char far *nombre);
```

Esta función carga un fichero de fuente escalable que no está provisto con el sistema BGI. El parámetro ***nombre** especifica el nombre del fichero fuente a cargar, en el directorio de inicio. El sistema gráfico puede tener hasta veinte fuentes instaladas a la vez.

Valor de retorno

La función *installuserfont* retorna el número de identificación de la fuente que es usada para seleccionar la fuente nueva a través de la función settextstyle. Si la tabla interna de fuentes está llena, un valor de -11 (grError) es retornado, indicando un error.

EJEMPLO

```
/* Este programa no funcionará, ya que se
** necesitaría tener una fuente nueva
** y desconocida por las librerías gráficas de BGI.
** Esto sólo es para poner un ejemplo.
*/
#include <graphics.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int fuente_SMGF;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
/* Intentamos instalar nuestra fuente nueva:
** Súper Mega Chula Fuente (SMGF)
** Ya sé que suena muy cursi, pero esto sólo es un ejemplo:)
*/
   if ( (fuente_SMGF = installuserfont ("SMGF.CHR" )) != grError)
      settextstyle (fuente_SMGF, HORIZ_DIR, 4);
   else
      settextstyle( DEFAULT_FONT, HORIZ_DIR, 4 );
   closegraph ( );
   return 0;
}
```

Función line

```
void far line (int x1, int y1, int x2, int y2);
```

Esta función es usada para conectar dos puntos con una línea recta. El primer punto es especificado por los argumentos **x1** e **y1**. El segundo punto es especificado por los argumentos **x2** e **y2**. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual. La posición del cursor gráfico no es afectado por la función *line*.

Valor de retorno

La función *line* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   line (20, 40, 350, 100);
   line ( 400, 30, 50, 250);
   getch ( );
```

```
   closegraph ( );
   return 0;
}
```

Función linerel

```
void far linerel (int dx, int dy);
```

Esta función es usada para dibujar una línea recta a una distancia y dirección predeterminadas desde la posición actual del cursor gráfico. El argumento **dx** específica el número relativo de píxels para atravesar en la dirección horizontal. El argumento **dy** específica el número relativo de píxels para atravesar en la dirección vertical. Estos argumentos pueden ser tanto valores positivos como negativos. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual desde la posición actual del cursor gráfico a través de la distancia relativa especificada. Cuando la línea esté terminada, la posición del cursor gráfico es actualizado al último punto de la línea.

Valor de retorno

La función *linerel* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
```

```
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   moveto (20, 20);
   linerel (20, 40);
   linerel (40, 30);
   getch ( );closegraph ( );return 0; }
```

Función lineto

```
void far lineto (int x, int y);
```

Esta función es usada para dibujar una línea recta desde la posición actual del cursor gráfico hasta el punto especificado por los argumentos **x** e **y**. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual. Después de que la línea recta haya sido dibujado, la posición del cursor gráfico es actualizado a la posición especificado por los argumentos **x** e **y** (el punto final de la línea).

Valor de retorno

La función *lineto* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
```

```
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   moveto (20, 20);
   lineto (40, 60);
   lineto (80, 90);
   getch ( );
   closegraph ( );
   return 0;
}
```

Función moverel

```
void far moverel (int dx, int dy);
```

Esta función es usada para mover la posición del cursor gráfico a una distancia relativa como los argumentos **dx** y **dy**. El argumento **dx** define la distancia relativa a moverse en la dirección horizontal. El argumento **dy** define la distancia relativa a moverse en la dirección vertical. Estos valores pueden ser positivos o negativos. No se dibuja ya que el cursor es mudado.

Valor de retorno

La función *moverel* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   moveto (20, 20);
   linerel (20, 40);
   moverel (50, 50);
   linerel (40, 30);
  getch ( );
   closegraph ( );
   return 0;
}
```

Función moveto

```
void far moveto (int x, int y);
```

Esta función es usada para colocar el cursor gráfico al punto especificado por los argumentos **x** e **y**. Ya que el cursor es movido desde su posición anterior al punto especificado por los argumentos **x** e **y**, no hay dibujo alguno.

Valor de retorno

La función *moveto* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   moveto (20, 20);
   lineto (40, 60);
   lineto (80, 90);
   getch ( );
   closegraph ( );
   return 0;
}
```

Función outtext

```
void far outtext (char far *cadena_texto);
```

Esta función es usada para mostar una cadena de caracteres. El argumento ***cadena_texto** define la cadena de texto a ser mostrado. La cadena es mostrado donde está el cursor gráfico actualmente usando el color actual y fuente, dirección, valores, y justificaciones de texto. La posición del cursor permanece sin ser cambiado al menos que la justificación horizontal actual es LEFT_TEXT y la orientación del texto es HORIZ_DIR. Cuando esto sea el caso, la posición del cursor es colocada horizontalmente a la anchura del píxel de la cadena de texto. Además, cuando se use la fuente por defecto, cualquier texto que se extiende a fuera del área gráfica actual es truncado.

Aunque la función *outtext* está diseñada para texto sin formato, texto con formato puede ser mostrada a través del uso de un búfer de caracteres y la función sprintf.

Valor de retorno

La función *outtext* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char mensaje [40];
   char nombre [25];
  printf ("Escribe tu nombre: ");
   scanf ("%s", nombre);
   sprintf (mensaje, "Hola %s!", nombre);
```

```
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   outtext (mensaje);
   outtextxy (300, 150, mensaje);
   getch ( );
   closegraph ( );
   return 0;
}
```

Función outtextxy

```
void far outtextxy (int x, int y, char far *cadena_texto);
```

Esta función es usada para mostar una cadena de caracteres. El argumento ***cadena_texto** define la cadena de texto a ser mostrado. La cadena es mostrada en la posición descrita por los argumentos **x** e **y** usando el color actual y fuente, dirección, valores, y justificaciones de texto. Cuando se use la fuente por defecto, cualquier texto que se extiende fuera del área gráfica actual es truncado.

Aunque la función *outtextxy* está diseñada para texto sin formato, texto con formato puede ser mostrada a través del uso de un búfer de caracteres y la función sprintf.

Valor de retorno

La función *outtextxy* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char mensaje [40];
   char nombre [25];
   printf ("Escribe tu nombre: ");
   scanf ("%s", nombre);
   sprintf (mensaje, "Hola %s!", nombre);
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
  registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   outtext (mensaje);
   outtextxy (300, 150, mensaje);
   getch ( );
   closegraph ( );
   return 0;
}
```

Función pieslice

```
void far pieslice (int x, int y,
```

```
   int comienzo_angulo, int final_angulo, int radio);
```

Esta función es usada para dibujar y rellenar un una cuña circular. La cuña circular está centrada en el punto especificado por los argumentos **x** e **y**. La porción circular de la cuña comienza con el ángulo especificado por el argumento **comienzo_angulo** y se extiende en un sentido contrario a las agujas del reloj al ángulo especificado por el argumento **final_angulo**. La función *pieslice* considera este - el eje horizontal a la derecha del centro - como su punto de referencia de 0 grados. El perímetro de la cuña es dibujado con el color actual y es rellenado con la trama y color de relleno actual.

Valor de retorno

La función *pieslice* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   pieslice (300, 150, 45, 225, 50);
```

```
  getch ( );    /* Pausa */
  closegraph ( );
  return 0;
}
```

Función putimage

```
void far putimage (int izquierda, int superior,
  void far *imagen, int accion);
```

Esta función coloca una imagen que fue previamente guardada con la función getimage en la pantalla. La esquina superior izquierda donde será colocada la imagen está definida por el argumento izquierdo y superior. Estos argumentos representan las coordenades *x* e *y* de la esquina superior izquierda, respectivamente. El argumento ***image** apunta al búfer de memoria donde la imagen está guardada. La imagen se coloca en la pantalla con la acción defindia en el argumento **accion**. Los valores y consonantes usados por el argumento **accion** se describen a continuación:

Constante	Valor	Significado
COPY_PUT	0	Sobrescribir los píxels existentes
XOR_PUT	1	Operación OR Exclusivo con los píxels
OR_PUT	2	Operación OR Inclusivo con los píxels
AND_PUT	3	Operación AND con los píxels
NOT_PUT	4	Invertir la imagen

Valor de retorno

La función *putimage* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   void *imagen;
   int imagentam;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setfillstyle (SLASH_FILL, 5);
   bar (50, 50, 350, 300);
   imagentam = imagesize (50, 50, 100, 100);
   imagen = malloc (imagentam);
   getimage (50, 50, 100, 100, imagen);
   putimage (400, 50, imagen, COPY_PUT);
   putimage (400, 110, imagen, COPY_PUT);
   getch ( );    /* Pausa */
   closegraph ( );
   free (imagen);
   return 0; }
```

Función putpixel

```
void far putpixel (int x, int y, int color);
```

Esta función es usada para asignar el valor del color a un píxel en particular. La posición del píxel en cuestión está especificado por los argumentos **x** e **y**. El argumento **color** específica el valor del color del píxel.

Existen varios valores para describir colores.

Valor de retorno

La función *putpixel* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int t;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
```

```
  for (t=0; t<200; t++)
    putpixel (100+t, 50+t, t%16);
  getch ( );
  closegraph ( );
  return 0;
}
```

Función rectangle

```
void far rectangle (int izquierda,
  int superior, int derecha, int inferior);
```

Esta función dibujará un rectángulo sin rellenar su interior usando el color actual. La esquina superior izquierda del rectángulo está definida por el argumento izquierdo y superior. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha del rectángulo. El perímetro del rectángulo es dibujado usando el estilo y grosor de línea actuales.

Valor de retorno

La función *rectangle* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
```

```
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   rectangle (20, 20, 400, 300);
  getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función registerbgidriver

```
int registerbgidriver (void (*driver) (void));
```

Esta función es usada para cargar y registrar un dispositivo gráfico. El argumento ***driver** apunta a un dispositivo. Un fichero de dispositivo registrado puede ser tanto cargado desde el disco o convertido en un formato .OBJ y ligado (o enlazado) dentro del programa. Registrando el dispositivo de esta manera, el fichero .EXE no depende de un fichero externo de dispositivo para poder ejecutarse.

Valor de retorno

La función *registerbgidriver* retorna número del dispositivo cuando tiene éxito. Un código de error, un número negativo, es retornado si el dispositivo especificado es inválido.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
*/
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
*/
   rectangle (20, 20, 400, 300);
   getch();    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función registerbgifont

```
int registerbgifont (void (*fuente) (void));
```

Esta función es usada para informar al sistema que la fuente apuntada por el argumento ***fuente** fue incluida durante el enlazo. Un fichero de fuente registrado ha de ser convertido en un fichero objeto .OBJ y ligado (o enlazado) dentro del programa. Registrando la fuente de esta manera, el fichero .EXE no depende de un fichero externo de fuentes para poder ejecutarse.

Nota: La fuente de defecto es la única que está disponible en el programa, ya que forma parte del sistema gráfico; no es necesario ligarlo al programa.

Valor de retorno

La función *registerbgifont* retorna número del dispositivo cuando tiene éxito. Un código de error, un número negativo, es retornado si el dispositivo especificado es inválido.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:
*/
   registerbgidriver (EGAVGA_driver);
   registerbgifont (sansserif_font);
   initgraph (&gdriver, &gmodo, "");
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
*/
   outtext ("Esto es una prueba con la fuente \"Sans Serif\"");
   getch ( );    /* Pausa */
   closegraph ( );
  return 0; }
```

Función restorecrtmode

```
void far restorecrtmode (void);
```

Esta función es usada para reiniciar el modo gráfico del vídeo al modo en uso anterior a la incialización del sistema gráfico. Esta función suele ser usada en conjunción con la función setgraphmode para cambiar entre ambos modos de texto y de gráficos.

Valor de retorno

La función *restorecrtmode* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   outtext ("Esto es una prueba para cambiar entre modo gráfico...");
```

```
  getch ( );
  restorecrtmode ( );
  printf ("...y en modo texto.\nPulsa una tecla para volver\n");
  getch ( );
  setgraphmode (gmodo);
  rectangle (200, 100, 400, 250);
    getch ( );    /* Pausa */
  closegraph ( ); return 0; }
```

Función sector

```
void far sector (int x, int y,
  int comienzo_angulo, int final_angulo, int x_radio, int y_radio);
```

Esta función es usada para dibujar una cuña elíptica. El centro de la cuña elíptica es especificado por los argumentos **x** e **y**. El argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical de la cuña elíptica. La cuña elíptica comienza al ángulo especificado por el argumento **comienzo_angulo** y es dibujado en la dirección contraria al de las agujas del reloj hasta llegar al ángulo especificado por el argumento **final_angulo**. La cuña elíptica es dibujado con el perímetro en el color actual y rellenada con el color de relleno y la trama de relleno actuales.

Valor de retorno

La función *sector* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
```

```
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setfillstyle (SOLID_FILL, 6);
   sector (300, 150, 45, -45, 150, 50);
   getch ( );    /* Pausa */ closegraph ( );return 0; }
```

Función setactivepage

```
void far setactivepage (int pagina);
```

Esta función es usada para especificar un número de página que representa una sección de memoria del vídeo donde todos los datos gráficos para mostrar son enviados. Está sección de memoria se denomina una página activa. El argumento **pagina** específica el número de la página activa. Para usar esta función con eficacia, el adaptador de vídeo usado debe ser EGA o VGA y tener suficiente memoria para soportar múltiples páginas para gráficos. Esta función es usada con la función setvisualpage para dibujar páginas no visuales y para crear animación.

Valor de retorno

La función *setactivepage* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int visual=1;
   printf ("Instrucciones:\nPulsa el espacio para cambiar de "
      "página, cualquier otra tecla para salir\n");
   printf ("(Pulsa cualquier tecla para entrar en modo gráfico)\n");
   getch ( );
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setactivepage (0);
   setfillstyle (SOLID_FILL, 6);
   sector (300, 150, 45, 315, 150, 50);
   setactivepage (1);
   setfillstyle (SOLID_FILL, 6);
   sector (300, 150, 90, 270, 150, 50);
   while (getch ( ) == ' ' ) {
      setvisualpage (visual);
      visual = 0==visual ? 1 : 0;
   }
   closegraph ( );
```

```
  return 0;
}
```

Función setallpalette

```
void far setallpalette (struct palettetype far *paleta);
```

Esta función es usada para asignar la paleta actual a la paleta definida en la estructura del tipo **palettetype** que es apuntado por el argumento ***paleta**. Todos los colores de la paleta actual son asignados a aquéllos definidos en la estructura palettetype. La síntaxis de la estructura **palettetype** es:

```
#define MAXCOLORS 15
struct palettetype {
  unsigned char size;
  signed char colors [MAXCOLORS+1];
}
```

El campo **size** indica el número de colores de la paleta actual. El campo **colors** es un array que contiene los valores numéricos que representan los colores que ofrece el dispositivo en su paleta de colores. Si la entrada de cualquier elemento del array es -1, el valor del color de ese elemento no cambiará.

Nota: Recuerda que todos los cambios hechos a la paleta tiene un efecto visual inmediato y que la función setallpalette no debería usarse con el dispositivo IBM-8514.

Valor de retorno

La función setallpalette no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y la paleta no es alterada.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int size, temp, i, y=0;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getpalette (&palette);
   size = palette.size;
   for (i=0; i<size; i++) {
      y += 30;
      setcolor (palette.colors [i]);
      line (20, y, 520, y);
   }
   Getch ( );    /* Pausa */
   for (i=0; i<size/2; i++) {
      temp = palette.colors[i];
      palette.colors [i] = palette.colors [size-1-i];
      palette.colors [size-1-i] = temp;
```

```
  }
  setallpalette (&palette);
  getch ( );    /* Pausa */
  closegraph ( );
  return 0;
}
```

Función setaspectratio

```
void far setaspectratio (int x_proporcion, int y_proporcion);
```

Esta función es usada para modificar la proporción anchura-altura del modo gráfico actual. La proporción anchura-altura puede definirse como la proporción de la anchura del píxel del modo gráfico y la altura del píxel. Esta proporción es usada por el sistema gráfico para calcular círculos y arcos. Por ello, alterando la proporción anchura-altura afectará la visualización de estas funciones. La función getaspectratio puede ser usada para obtener las opciones por defecto del modo actual anteriormente a ser modificados.

Valor de retorno

La función *setaspectratio* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  int x_proporcion, y_proporcion;
```

```
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getaspectratio (&x_proporcion, &y_proporcion);
   circle (300, 150, 50);
   getch ( );    /* Pausa */
   setaspectratio (2*x_proporcion, y_proporcion);
   circle (300, 150, 50);
   getch ( );    /* Pausa */
   closegraph ( );   return 0; }
```

Función setbkcolor

```
void far setbkcolor (int color);
```

Esta función es usada para asignar el color de fondo al valor del color de fondo especificado por el argumento **color**.

Existen varios valores para ciertos colores de fondo.

Valor de retorno

La función *setbkcolor* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
```

```
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setbkcolor (4);
   circle (300, 150, 50);
   getch();    /* Pausa */
   closegraph ( );
  return 0; }
```

Función setfillpattern

```
void far setfillpattern (char far *trama, int color);
```

Esta función es usada para seleccionar una trama de relleno definido por el usuario. El argumento ***trama*** apunta a una serie de ocho bytes que representa una trama de relleno de bits de 8 x 8. Cada byte representa una fila de ocho bits, donde cada bit está encendido o no (1 ó 0). Un bit de 0 indica que el píxel correspondiente será asignado el color de relleno actual. Un bit de 0 indica que el píxel correspondiente no será alterado. El argumento **color** especifica el color de relleno que será usado para la trama.

Valor de retorno

La función *setfillpattern* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char trama1 [8] = { 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE };
   char trama2 [8] = { 0x0A, 0xF0, 0xF0, 0x0A, 0x0A, 0xF0, 0xF0, 0x0A };
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   bar (50, 50, 150, 150);
   setfillpattern (trama1, 9);
   bar (160, 50, 260, 150);
   setfillpattern (trama2, 4);
   bar (105, 160, 205, 260);
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función setfillstyle

```
void far setfillstyle (int trama, int color);
```

Esta función es usada para seleccionar una trama predefinida y un color de relleno. El argumento **trama** específica la trama predefinida, mientras que el argumento **color** especifica el color de relleno.

Existen trece valores ya definidos para tramas. Sin embargo, la trama USER_FILL (valor 12) no debería usarse para asignar unla trama definida por el usuario. En su lugar, se debería usar la función setfillpattern.

Valor de retorno

La función *setfillstyle* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setfillstyle (LTSLASH_FILL, 6);
   bar (50, 50, 350, 300);
   getch ( );    /* Pausa */
   closegraph ( );
```

```
  return 0; }
```

Función setgraphbufsize

```
unsigned far setgraphbufsize (unsigned bufer_tam);
```

Esta función es usada para cambiar el tamaño del búfer gráfico interno como es asignado por la función initgraph cuando el sistema gráfico es inicializado. El búfer gráfico es usado por varias funciones gráficos; por ello, se debería tener un mayor cuidado cuando se altera este búfer del tamaño por defecto de 4096. La función *setgraphbufsize* se debería llamar antes de llamar a la función initgraph.

Valor de retorno

La función *setgraphbufsize* retorna el tamaño anterior del búfer gráfico interno.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
  int buf_inicial, buf_nuevo=10000;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
  registerbgidriver (EGAVGA_driver);
  initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
```

```
  Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
  buf_inicial = setgraphbufsize (buf_nuevo);
  closegraph ( );
  printf ("Búfer inicial: %d\tBúfer nuevo: %d\n", buf_inicial, buf_nuevo);
 return 0; }
```

Función setgraphmode

```
void far setgraphmode (int modo);
```

Esta función es usada para seleccionar el modo gráfico actual pero solamente cuando el sistema gráfico haya sido inicializado con la función initgraph. El argumento **modo** define el modo a usar según el dispositivo actual. Además de seleccionar un nuevo modo, la función *setgraphmode* despeja la pantalla y reinicia todas las opciones gráficas a sus valores por defecto. Esta función suele usarse conjuntamente con restorecrtmode para cambiar entre modos gráficos y de texto.

Valor de retorno

La función *setgraphmode* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:
```

```
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   outtext ("Esto es una prueba para cambiar entre modo gráfico...");
   getch ( );
   restorecrtmode ( );
   printf ("...y en modo texto.\nPulsa una tecla para volver\n");
   getch ( );
   setgraphmode (gmodo);
   rectangle( 200, 100, 400, 250 );
      getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función setlinestyle

```
void far setlinestyle (int estilo,
   unsigned trama, int grosor);
```

Esta función es usada para definir las características de líneas para líneas rectas. El argumento **estilo** específica la trama de línea predefinida para su uso. El argumento **trama** es una trama de 16 bits que describe el estilo de línea cuando el argumento **estilo** es USERBIT_LINE, ó 4. Un bit 1 en esta trama indica que el píxel correspondiente será asignado el color actual. Un bit 0 indica que el píxel correspondiente no será alterado. El argumento **grosor** define el grosor de la línea.

Existen varios valores para los diferentes estilos y grosores de líneas rectas.

Valor de retorno

La función *setlinestyle* no retorna ningún valor; sin embargo, si un argumento es inválido, entonces la función graphresult retorna grError (11).

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setlinestyle (DOTTED_LINE, 0, THICK_WIDTH);
   line (200, 300, 400, 50);
      getch ( );    /* Pausa */
   closegraph();

   return 0;
}
```

Función setpalette

```
void far setpalette (int num_paleta, int color);
```

Esta función es usada para modificar una sola entrada en la paleta actual. El argumento **num_paleta** específica el miembro de la paleta a cambiar. El argumento **color** específica el nuevo valor de color para el miembro de la paleta.

Existen varios valores para los colores dependiendo del dispositivo.

Nota: Recuerda que todos los cambios hechos a la paleta tiene un efecto visual inmediato y que la función setpalette no debería usarse con el dispositivo IBM-8514.

Valor de retorno

La función setpalette no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y la paleta no es alterada.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int size, temp, i, y=0;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
```

```
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getpalette (&palette);
   size = palette.size;
   for (i=0; i<size; i++) {
      y += 30;
      setcolor (palette.colors [i]);
      line (20, y, 520, y);
   }
  getch ( );    /* Pausa */
   for (i=0; i<size/2; i++) {
      temp = palette.colors [i];
      setpalette (i, palette.colors[size-1-i]);
      setpalette (size-1-i, temp);
   }
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función setrgbpalette

```
void far setrgbpalette (int num_paleta,
   int rojo, int verde, int azul);
```

Esta función es para usarse con los dispositivos de IBM 8514 y VGA. El argumento **num_paleta** especifica el miembro de la paleta a ser modificado. Para

la IBM 8514 (y para el modo de 256K de la VGA), el intervalo de la paleta es de 0 á 255. Para los modos de VGA, el intervalo es de 0 á 15. Los argumentos **rojo**, **verde**, y **azul** especifican la intensidad del color para el miembro de la paleta. De cada byte (de cada argumento) sólo los seis bits más signifcativos son cargados en la paleta.

Por razones de compatibilidad con otros adaptadores gráficos de IBM, el dispositivo BGI define las primeras dieciséis entradas a la paleta de la IBM 8514 a los colores por defecto de la EGA/VGA.

Nota: Recuerda que todos los cambios hechos a la paleta tiene un efecto visual inmediato y que la función setrgbpalette no debería usarse con el dispositivo IBM-8514.

Valor de retorno

La función setrgbpalette no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int size, i, y=0;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
```

```
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   getpalette (&palette);
   size = palette.size;
   for (i=0; i<size; i++) {
      y += 30;
      setcolor (palette.colors [i]);
      line (20, y, 520, y);
   }
   getch ( );    /* Pausa */
   for (i=0; i<size; i++)
      setrgbpalette (i, 2*i+33, 42, 63-4*i);
 /* Tonos de naranja y azul */
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función settextjustify

```
void far settextjustify (int horizontal, int vertical);
```

Esta función es usada para especificar el método en el cual el texto es colocado en la pantalla con relación a la posición del cursor. El argumento **horizontal** define la justificación horizontal, mientras que el argumento **vertical** indica la justificación vertical.

Existen varios valores y constantes para las justificaciones.

Valor de retorno

La función *settextjustify* no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y la paleta no es alterada.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
  settextjustify (RIGHT_TEXT, BOTTOM_TEXT);
   moveto (300, 200);
   outtext ("(RIGHT_TEXT, BOTTOM_TEXT)");
   settextjustify (RIGHT_TEXT, TOP_TEXT);
   moveto (300, 200);
   outtext ("(RIGHT_TEXT, TOP_TEXT)");
   settextjustify (LEFT_TEXT, BOTTOM_TEXT);
   moveto (300, 200);
   outtext ("(LEFT_TEXT, BOTTOM_TEXT)");
   settextjustify (LEFT_TEXT, TOP_TEXT);
   moveto (300, 200);
   outtext ("(LEFT_TEXT, TOP_TEXT)");
   setcolor (1);
```

```
  line (300, 200, 300, 100);
  setcolor (2);
  line (300, 200, 300, 300);
  setcolor (3);
  line (300, 200, 100, 200);
  setcolor (4);
  line (300, 200, 500, 200);
  getch ( );
  closegraph ( );
  return 0;
}
```

Función settextstyle

```
void far settextstyle (int fuente,
  int orientacion, int tam_caracter);
```

Esta función es usada para especificar las características para la salida de texto con fuente. El argumento **fuente** especifica la fuente registrada a usar. La fuente ha de estar registrada para resultados predecibles; es decir, usa registerbgifont antes de usar esta función. El argumento **orientacion** especifica la orientación en que el texto ha de ser mostrado. La orientación por defecto es HORIZ_DIR. El argumento **tam_caracter** define el factor por el cual la fuente actual será multiplicada. Un valor distinto a 0 para el argumento **tam_caracter** puede ser usado con fuentes escalables o de bitmap. Sin embargo, un valor distinto a 0 para el argumento **tam_caracter**, el cual selecciona el tamaño del carácter definido por el usuario usando la función setusercharsize, solamente funciona con fuentes escalables. El argumento **tam_caracter** puede agrandar el tamaño de la fuente hasta 10 veces su tamaño normal.

Existen varios valores y constantes para las justificaciones.

Valor de retorno

La función *settextstyle* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char mensaje [40];
   char nombre [25];
   printf ("Escribe tu nombre: ");
   scanf ("%s", nombre);
   sprintf (mensaje, "Hola %s!", nombre);
/* Esta fuente ha de ser enlazada antes de poder registrarla
   registerbgifont (sansserif_font);
*/
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   settextstyle (DEFAULT_FONT, 0, 2);
   outtextxy (100, 50, mensaje);
   settextstyle (DEFAULT_FONT, 1, 2);
```

```
  outtextxy (200, 125, mensaje);
  settextstyle (SANS_SERIF_FONT, 1, 3);
  outtextxy (400, 150, mensaje);
  getch ( );
  closegraph ( );
  return 0;
}
```

Función setusercharsize

```
void far setusercharsize (int x_dividendo, int x_divisor,
  int y_dividendo, int y_divisor);
```

Esta función establece las características de fuentes escalables. Para que esta función afecte el tamaño del carácter, el argumento **tam_caracter** de la función settextstyle debe ser 0. La anchura del carácter se establece con los argumentos **x_dividendo** y **x_divisor** que representan la proporción. Similarmente, los argumentos **y_dividendo** e **y_divisor** especifican la altura del carácter.

Valor de retorno

La función *setusercharsize* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
  int gdriver = EGA;
  int gmodo = EGAHI;
```

```
   char mensaje [40];
   char nombre [25];
   printf ("Escribe tu nombre: ");
   scanf ("%s", nombre);
   sprintf (mensaje, "Hola %s!", nombre);
/* Esta fuente ha de ser enlazada antes de poder registrarla */
   registerbgifont (sansserif_font);
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   settextstyle (SANS_SERIF_FONT, 0, 0);
   setusercharsize (1, 4, 1, 2);    /* 25% de ancho; 50% de alto */
   outtextxy (100, 50, mensaje);
   settextstyle (SANS_SERIF_FONT, 0, 1);
   outtextxy (100, 125, mensaje);
   settextstyle (SANS_SERIF_FONT, 1, 0);
   setusercharsize (1, 2, 3, 4);    /* 50% de ancho; 75% de alto */
   outtextxy (400, 150, mensaje);
   getch ( );
   closegraph ( );
 return 0; }
```

Función setviewport

```
void far setviewport (int izquierda, int superior,
   int derecha, int inferior, int recorte_banderin);
```

Esta función es usada para definir el área gráfico. La esquina superior izquierda del área gráfica está definida por los argumentos **izquierda** y **superior**. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha del área gráfica. El argumento **recorte_banderin** define si los datos para la salida gráfica serán recortados por el borde del área gráfico. Un valor de 0 para **recorte_banderin** indica que los datos de salida no serán recortados, mientras que un valor distinto a 0 indica que los datos serán recortados. Cuando el área gráfica es incializada, la posición del cursor será mudado a la posición (0,0) (la esquina superior izquierda). Todos los datos de salida después de que el área gráfica haya sido inicializada serán con relación a este punto. El área gráfica por defecto cubre la pantalla entera.

Valor de retorno

La función *setviewport* no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y el área gráfica no será alterada.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
```

```
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   lineto (100, 100);
   outtextxy (15, 5, "Inicial");
   getch ( );
   setviewport (250, 200, 450, 300, 0);
   setcolor (9);
   lineto (100, 100);
   outtextxy (15, 5, "Nueva");
   moveto (0, 0);
   lineto (-50, -20); /* Fuera del área */
   getch ( );
   setviewport (250, 200, 450, 300, 1);
   setcolor (4);
   moveto (120, 40);
   lineto (150, -20);  /* Fuera del área */
   outtextxy (25, 15, "Con recorte");
   getch ( );    /* Pausa */
   closegraph ( );
   return 0;
}
```

Función setvisualpage

```
void far setvisualpage (int pagina);
```

Esta función es usada para establecer la página visual como es especificado por el argumento **pagina**. Una página es una sección de memoria donde se guarda la información del vídeo. Cuando se usa con un sistema (EGA o VGA) con suficiente

memoria de vídeo para soportar múltiples páginas de gráficos, la función *setvisualpage* (junto con la función setactivepage) perimte al programador crear gráficos en páginas escondidas y pasar de página entre las que se han definido con información gráfica. Esto es la base para crear animación.

Valor de retorno

La función *setvisualpage* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int visual=1;
   printf ("Instrucciones:\nPulsa el espacio para cambiar de página, cualquier otra tecla para salir\n");
   printf ("(Pulsa cualquier tecla para entrar en modo gráfico)\n");
   getch ( );
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setactivepage (0);
```

```
  setfillstyle (SOLID_FILL, 6);
  sector (300, 150, 45, 315, 150, 50);
  setactivepage (1);
  setfillstyle (SOLID_FILL, 6);
  sector (300, 150, 90, 270, 150, 50);
  while (getch ( ) == ' ' ) {
    setvisualpage( visual );
    visual = 0==visual ? 1 : 0;
  }
  closegraph ( );
  return 0;
}
```

Función setwritemode

```
void far setwritemode (int modo);
```

Esta función es usada para establecer el modo lógico de escritura para líneas rectas. El argumento **modo** especifica el modo de escritura, el cual determina la interacción entre valores de píxels existentes y los valores de píxels en la línea.

Existen dos valores para los modos de escritura.

Valor de retorno

La función *setwritemode* no retorna ningún valor.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
```

```
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   setfillstyle (SOLID_FILL, 1);
   bar (50, 50, 500, 300);
   setwritemode (COPY_PUT);
   setcolor (10);
   line (20, 60, 220, 100);
   setwritemode (XOR_PUT);
   line (20, 80, 220, 120);
   getch ( );
   closegraph ( );
 return 0; }
```

Función textheight

```
int far textheight (char far *texto);
```

Esta función es usada para determinar la altura, en píxels, de la cadena de texto especificada por el argumento ***texto**. La altura del texto se determina usando la fuente actual y el tamaño del carácter.

Valor de retorno

La función *textheight* retorna la altura, en píxels, del texto especificado por el argumento.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int anchura, altura;
   char mensaje [5] = "Hola";
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   anchura = textwidth (mensaje);
   altura = textheight (mensaje);
   closegraph ( );
   printf ("El mensaje: \"%s\" tiene de anchura: %d y altura: %d\n", mensaje,
anchura, altura);
   printf ("Pulsa una tecla para continuar...\n");
   getch ( );
   return 0;
}
```

Función textwidth

```
int far textwidth (char far *texto);
```

Esta función es usada para determinar la anchura, en píxels, de la cadena de texto especificada por el argumento ***texto**. La anchura del texto se determina usando la fuente actual y el tamaño del carácter.

Valor de retorno

La función *textwidth* retorna la anchura, en píxels, del texto especificado por el argumento.

EJEMPLO

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main ( ) {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int anchura, altura;
   char mensaje [5] = "Hola";
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver (EGAVGA_driver);
   initgraph (&gdriver, &gmodo, "");
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   Initgraph (&gdriver, &gmodo, "C:\\BC5\\BGI");
   anchura = textwidth (mensaje);
```

```
  altura = textheight (mensaje);
  closegraph ( );
  printf ("El mensaje: \"%s\" tiene de anchura: %d y altura: %d\n", mensaje, anchura, altura); printf ("Pulsa una tecla para continuar...\n"); getch ( );return 0; }
```

6.32 MACROS

Tablas de Colores

Colores de Fondo

Constante	Valor	Significado
BLACK	0	Negro
BLUE	1	Azul
GREEN	2	Verde
CYAN	3	Cían
RED	4	Rojo
MAGENTA	5	Magenta
BROWN	6	Marrón
LIGHTGRAY	7	Gris Claro
DARKGRAY	8	Gris Oscuro
LIGHTBLUE	9	Azul Claro
LIGHTGREEN	10	Verde Claro
LIGHTCYAN	11	Cían Claro
LIGHTRED	12	Rojo Claro
LIGHTMAGENTA	13	Magenta Claro
YELLOW	14	Amarillo
WHITE	15	Blanco

Colores para Modos de 16 Colores

Constante	Valor	Significado
BLACK	0	Negro
BLUE	1	Azul
GREEN	2	Verde
CYAN	3	Cían
RED	4	Rojo
MAGENTA	5	Magenta
BROWN	6	Marrón
LIGHTGRAY	7	Gris Claro
DARKGRAY	8	Gris Oscuro
LIGHTBLUE	9	Azul Claro
LIGHTGREEN	10	Verde Claro
LIGHTCYAN	11	Cían Claro
LIGHTRED	12	Rojo Claro
LIGHTMAGENTA	13	Magenta Claro
YELLOW	14	Amarillo
WHITE	15	Blanco

Colores para modos de CGA

Número de Paleta	Color 1	Significado	Color 2	Significado	Color 3	Significado
0	CGA_LIGHTGREEN	Verde Claro	CGA_LIGHTRED	Rojo Claro	CGA_YELLOW	Amarillo
1	CGA_LIGHTCYAN	Cían Claro	CGA_LIGHTMAGENTA	Magenta Claro	CGA_WHITE	Blanco
2	CGA_GREEN	Verde	CGA_RED	Rojo	CGA_BROWN	Marrón
3	CGA_CYAN	Cían	CGA_MAGENTA	Magenta	CGA_LIGHTGR	Gris Claro

			AY
Valor asignado:	1	2	3

Nota: Color 0 se reserva para el color de fondo y se asigna con lo función setbkcolor*, pero los demás colores son fijos. Estas constantes se usan con* setcolor*.*

Colores para la paleta

Constante (CGA)	**Valor**	**Constante (EGA/VGA)**	**Valor**
BLACK	0	EGA_BLACK	0
BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_LIGHTGRAY	7
LIGHTGRAY	7	EGA_BROWN	20
DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61
YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

Nota: Estas constantes se usan con las funciones setpalette *y* setallpalette*.*

Tabla de Dispositivos

Dispositivos Gráficos	
Dispositivo/Constante	**Valor**
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

Enlazar Dispositivos y Fuentes

Para enlazar los ficheros de dispositivos gráficos y de fuentes escalables a programas que usan el BGI, se ha de convertir estos ficheros a ficheros objetos (.OBJ). Los ficheros objetos pueden ser enlazados (o ligados) al fichero ejecutable. Este proceso es ventajosa ya que permite que el programa acceda a los dispositivos gráficos y/o fuentes escalables directamente, sin tener que cargarlos desde el disco mientras el programa se esté ejecutando. La desventaja de este proceso es el tamaño del fichero ejecutable que lógicamente incrementará debido a que incorpora los ficheros enlazados.

Esta conversión se hace mediante el programa BGIOBJ.EXE. La síntaxis simplificada de este programa es la siguiente:

```
BGIOBJ <nombre>
```

donde <nombre> es el nombre del fichero: fuente escalable o dispositivo gráfico. El programa producirá otro fichero del mismo nombre con la extensión .OBJ.

Por ejemplo:

```
BGIOBJ EGAVGA.BGI
```

Esto convertirá el fichero EGAVGA.BGI a EGAVGA.OBJ.

Ahora hay que enlazar (o ligar) este fichero objeto con la librería gráfica. El modo de hacer esto es usando el programa TLIB.EXE. La síntaxis simplificada es la siguiente:

```
TLIB <nombre_librería> +<nombre_objeto1> [+<nombre_objeto2>...]
```

donde <nombre_librería> es la librería al cual queremos añadir los ficheros objetos cuyos nombres son <nombre_objeto1>, <nombre_objeto2>, etc.. Como sólo se puede añadir ficheros objetos a la librería, por eso no hace falta incluir la extensión .OBJ.

Por ejemplo:

```
TLIB GRAPHICS +EGAVGA +GOTH +BOLD +EURO
```

Esto añadirá los ficheros objetos EGAVGA.OBJ, GOTH.OBJ, BOLD.OBJ, y EURO.OBJ a la librería GRAPHICS.LIB.

Para poder seleccionar estos dispositivos y/o fuentes, se ha de registrarlos para que sean enlazados en el programa. La forma de hacer esto es a través de las funciones registerbgidriver y registerbgifont.en el programa - antes de llamar a initgraph. Esto informa al sistema gráfico de que tales ficheros están presentes y asegura que están enlazados cuando el enlazador (o linker) cree el fichero ejecutable. Las rutinas de registro aceptan cada uno un parámetro; un nombre simbólico definido en graphics.h. Cada rutina de registro retorna un valor no negativo si el dispositivo o fuente se registra con éxito.

He aquí la lista de los dispositivos y fuentes que se usan con la funciones registerbgidriver y registerbgifont:

Dispositivos/Fuentes y sus Nombres Simbólicos

Dispositivo (.BGI)	**Nombre simbólico (registerbgidriver)**	**Fuente (.CHR)**	**Nombre simbólico (registerbgifont)**
CGA	CGA_driver	TRIP	triplex_font
EGAVGA	EGAVGA_driver	LITT	small_font
HERC	Herc_driver	SANS	sansserif_font
ATT	ATT_driver	GOTH	gothic_font
PC3270	PC3270_driver		
IBM8514	IBM8514_driver		

Tabla de Errores

Códigos de Errores

Constante Código Significado

GrOk	0	Ningún error
grNoInitGraph	-1	Gráficos no iniciados
grNotDetected	-2	Ningún adaptador gráfico detectado
grFileNotFound	-3	Fichero de dispositivo no encontrado
grInvalidDriver	-4	Fichero de dispositivo no válido
grNoLoadMem	-5	No hay memoria para cargar dispositivo
grNoScanMem	-6	No hay memoria para rellenar
grNoFloodMem	-7	No hay memoria para usar floodfill
grFontNotFound	-8	Fichero de fuente no encontrado
grNoFontMem	-9	No hay memoria para cargar la fuente
grInvalidMode	-10	Modo gráfico no válido
GrError	-11	Error gráfico
GrIOerror	-12	Error gráfico de Entrada/Salida
grInvalidFont	-13	Fichero de fuente no válido
grInvalidFontNum	-14	Número de fuente no válido
grInvalidDeviceNum	-15	Número de dispositivo no válido
grInvalidVersion	-18	Número de versión no válido

Tablas de Fuentes

Fuentes para Texto

Constante	**Valor**	**Significado**
DEFAULT_FONT	0	Fuente bitmap de 8x8
TRIPLEX_FONT	1	Fuente escalable de tipo triple
SMALL_FONT	2	Fuente escalable pequeña
SANS_SERIF_FONT	3	Fuente escalable de tipo sans serif
GOTHIC_FONT	4	Fuente escalable de tipo gótico

SCRIPT_FONT	5	Fuente escalable de tipo manuscrito
SIMPLEX_FONT	6	Fuente escalable de tipo manuscrito simple
TRIPLEX_SCR_FONT	7	Fuente escalable de tipo manuscrito triple
COMPLEX_FONT	8	Fuente escalable de tipo complejo
EUROPEAN_FONT	9	Fuente escalable de tipo europeo
BOLD_FONT	10	Fuente escalable en negrita

Orientaciones para texto

Constante	Valor	Significado
HORIZ_DIR	0	Texto horizontal
VERT_DIR	1	Texto vertical

Justificación de Texto en la Horizontal

Constante	Valor	Significado
LEFT_TEXT	0	Justificar a la izquierda
CENTER_TEXT	1	Centrar el texto
RIGHT_TEXT	2	Justificar a la derecha

Justificación de Texto en la Vertical

Constante	Valor	Significado
BOTTOM_TEXT	0	Justificar debajo
CENTER_TEXT	1	Centrar el texto
TOP_TEXT	2	Justificar arriba

Tablas de Líneas

Estilos de Líneas

Constante	Valor	Significado
SOLID_LINE	0	Línea continua
DOTTED_LINE	1	Línea hecha con puntos
CENTER_LINE	2	Línea centrada
DASHED_LINE	3	Línea discontinua
USERBIT_LINE	4	Línea definida por el usuario

Grosores para Líneas

NORM_THICK	1	Grosor es de 1 píxel
THICK_WIDTH	3	Grosor es de 3 píxels

Modos de Escritura

Constantes	Valor	Significado
COPY_PUT	0	Píxels de la línea sobrescriben los píxels existentes
XOR_PUT	1	Píxels de la pantalla son el resultado de la operación OR de los píxels existentes y los de la línea

Tabla de Modos

Modos Gráficos

Dispositivo	Modo/Constante	Código	Resolución	Paleta	Páginas
CGA	CGAC0	0	320x200	4 colores	1
	CGAC1	1	320x200	4 colores	1
	CGAC2	2	320x200	4 colores	1
	CGAC3	3	320x200	4 colores	1
	CGAHI	4	640x200	2 colores	1
MCGA	MCGAC0	0	320x200	4 colores	1
	MCGAC1	1	320x200	4 colores	1

	MCGAC2	2	320x200	4 colores	1
	MCGAC3	3	320x200	4 colores	1
	MCGAMED	4	640x200	2 colores	1
	MCGAHI	5	640x480	2 colores	1
EGA	EGALO	0	640x200	16 colores	4
	EGAHI	1	640x350	16 colores	2
EGA64	EGA64LO	0	640x200	16 colores	1
	EGA64HI	1	640x350	4 colores	1
EGAMONO	EGAMONOHI	3	640x200	2 colores	1* / 2**
VGA	VGALO	0	640x200	16 colores	2
	VGAMED	1	640x350	16 colores	2
	VGAHI	2	640x480	16 colores	1
ATT400	ATT400C0	0	320x200	4 colores	1
	ATT400C1	1	320x200	4 colores	1
	ATT400C2	2	320x200	4 colores	1
	ATT400C3	3	320x200	4 colores	1
	ATT400MED	4	640x200	2 colores	1
	ATT400HI	5	640x400	2 colores	1
HERC	HERCMONOHI	0	720x348	2 colores	2
PC3270	PC3270HI	0	720x350	2 colores	1
IBM8514	IBM8514LO	0	640x480	256 colores	
	IBM8514HI	1	1024x768	256 colores	

* Si la tarjeta es de 64K

** Si la tarjeta es de 256K

Tabla de Operaciones con Putimage

Operaciones con putimage

Constante	Valor	Significado
COPY_PUT	0	Sobrescribir los píxels existentes
XOR_PUT	1	Operación OR Exclusivo con los píxels
OR_PUT	2	Operación OR Inclusivo con los píxels
AND_PUT	3	Operación AND con los píxels
NOT_PUT	4	Invertir la imagen

Nota: Estas operaciones se usan exclusivamente con la función putimage*.*

Tabla de Tramas

Tramas predefinidas

Constante	Valor	Significado
EMPTY_FILL	0	Rellena con el color de fondo
SOLID_FILL	1	Rellena enteramente
LINE_FILL	2	Rellena con líneas horizontales: ---
LTSLASH_FILL	3	Rellena con rayas finas: ///
SLASH_FILL	4	Rellena con rayas gruesas: ***///***
BKSLASH_FILL	5	Rellena con rayas inversas y finas: \\\
LTBKSLASH_FILL	6	Rellena con rayas inversas y gruesas: ****
HATCH_FILL	7	Rellena con líneas cruzadas cuadriculadamente: +++
XHATCH_FILL	8	Rellena con líneas cruzadas diagonalmente: XXXX
INTERLEAVE_FILL	9	Rellena con líneas entrelazadas
WIDE_DOT_FILL	10	Rellena con lunares bastante distanciados
CLOSE_DOT_FILL	11	Rellena con lunares poco distanciados
USER_FILL	12	Rellena con la trama definida por el usuario

Nota: Todos los tipos de tramas menos EMPTY_FILL usan el color de relleno seleccionado; EMPTY_FILL usa el color de fondo para rellenar.

EJEMPLO DESGLOSADO USANDO FUNCIONES DE GRAFICACIÓN EN TURBO C

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>

void main ( )
{
    int gdriver = EGA, gmodo = EGAHI, n, f [100];
    int angulo [100], angulo1 [100], angulo2 [100], radio;
    int ancho, i, j, k=0, clases, puntos [8];
    div_t d;
    float x [300], mayor = 0, menor, rango, marca;
    float lonClas, inicio, flot [100], marcas [100];
    float p [100], frec;
    char texto [100], nM [100];
    clrscr ( );

    ancho = 400;
    printf ("\t\t----- Graficaci¢n de Frecuencias de una Muestra -----\n\n\n");

    /*El tama¤o de la muestra tiene que ser par, no funciona bien si es impar*/
    printf ("¨De que tamaño es la muestra? "); scanf ("%d", &n);
    /*-----------------------------------------------------------------*/
    printf ("¨Cuantas clases? "); scanf ("%d", &clases);
    d = div (ancho, (clases*clases));

      for (i=0; i<n; i++)
```

```
 {
     printf ("Escribe el valor %d: ", i+1); scanf ("%f", &x [i]);
 }
for (i=0; i<n; i++)
{
    for (j=0; j<n; j++)
    {
        if (mayor<=x [j])
        {
            mayor = x [j];
        }
    }
}
menor = mayor;
   for (i=0; i<n; i++)
   {
       for(j=0; j<n; j++)
       {
           If (menor>=x [j])
           {
               menor = x [j];
           }
       }
    }

rango = mayor - menor;
lonClas = rango/clases;

marca = menor;

  for (i=0; i<clases; i++)
```

```
   {
      f [i] = 0;
        if (i == clases-1) marca=mayor;
      marcas [i] = marca;
        for (j=0; j<n; j++)
        {
          if ((marca-lonClas/2)<=x[j])
          {
               if ((marca+lonClas/2)>=x[j])
               {
                    f [i]++;
               }
          }
          marca += lonClas;
       }
     for (i=0; i<clases; i++) {
     flot [i] = f [i];
      p [i] = (flot [i]/n)*100;
  }

printf ("\n\n\n\nPresiona cualquier tecla para continuar...");
getch ( );
initgraph (&gdriver, &gmodo, "");
   for (i=0; i<clases; i++)
   {
      if (k>11){k=0;
    }
 k++;
 setfillstyle (k, k);
 puntos [0] = d.quot*i;
 puntos [1] = 0;
```

```
    puntos [2] = d.quot*i;
    puntos [3] = f[i]*d.quot;
    puntos [4] = d.quot+d.quot*i;
    puntos [5] = f[i]*d.quot;
    puntos [6] = d.quot+d.quot*i;
    puntos [7] = 0;
    fillpoly (4, puntos);
    fillellipse (480, 20* (i+1), 5, 5);
    gcvt (marcas [i],3,nM);
    strcpy (texto, "M: ");
    strcat (texto, nM);
    outtextxy (490, 20*(i+1), texto);
    texto [0] = '\0';
    nM [0] = '\0';
    itoa (f [i],nM,10);
    strcpy (texto, "F: ");
    strcat (texto, nM);
    outtextxy (560, 20*(i+1), texto);
    texto [0] = '\0';
    nM [0] = '\0';
  }
outtextxy (10, 280, "Gr fica de Barras");
outtextxy (10, 300, "M: Marca de Clase; F: Frecuencia");
getch ( );
clearviewport ( );
angulo2 [0] = 0;
radio = 100;
for (i=1; i<=clases; i++)
 {
       If (k>11) {k=0;
 }
```

```
 k++;
 setfillstyle (k, k);
 flot [i] = f [i-1];
 Angulo [i] = (flot [i]/n)*360;
 angulo1 [i] = angulo2 [i-1];
 angulo2 [i] = angulo1 [i] + Angulo [i];
 pieslice (320, 240, angulo1 [i], angulo2 [i], radio);
 fillellipse (480, 20*(i+1), 5, 5);
  gcvt (marcas [i-1],3,nM);
  strcpy (texto, "M: ");
  strcat (texto, nM);
  outtextxy (490, 20*(i+1), texto);
  texto [0] = '\0';
  nM [0] = '\0';
  gcvt (p [i-1],3,nM);
  strcpy (texto, "P: ");
  strcat (texto, nM);
  strcat (texto, "\%");
  outtextxy (560, 20*(i+1), texto);
  texto [0] = '\0';
  nM [0] = '\0';
 }
 outtextxy (10, 10, "Gr fica de Pastel");
 outtextxy (10, 30, "M: Marca de Clase; P: Porcentaje");
 getch ( );
 closegraph ( );
 printf ("Presiona cualquier tecla para salir...");
 getch ( );
}
```

PROBLEMAS SUGERIDOS

1. Realizar un programa que dibuje una línea.
2. Realizar un programa que dibuje una línea.
3. Realizar un programa que dibuje un círculo con centro y radio cualesquiera.
4. Realizar un programa que trace la tangente a un círculo en un punto dado.
5. Realizar un programa que dibuje un polígono y rellenarlo.
6. Realizar un programa que dibuje un triángulo equilátero y rellenarlo de cualquier color (ver tabla de colores).
7. Realizar un programa que dibuje un rectángulo.
8. Realizar un programa que dibuje 2 líneas paralelas.
9. Realizar un programa que dibuje un hexágono y rellenarlo.
10. Realizar un programa que rellene la pantalla de color amarillo.
11. Realizar un programa que dibuje un triángulo isóseles y rellenarlo de color azul.
12. dibujar un círculo y rellenar la pantalla de color verde.
13. Realizar un programa donde escribas una cadena de caracteres.
14. Realizar un programa que dibuje una elipse con radio horizontal 50 y radio vertical 30.
15. En base a los siguientes datos realizar una gráfica de pastel:

Las ventas de una zapatería en una semana fueron:

MODELO	# DE PARES VENDIDOS
2051	8
1822	7
3154	2
2010	9
5090	5
3040	3

16. Con los datos del ejercicio anterior realizar una gráfica de barras.
17. Hacer una tabla que tenga 3 columnas y 5 renglones.

18.Realizar un programa que dibuje 2 triángulos semejantes.

19.Realizar un programa que dibuje 4 círculos que se intercepten en cualesquier punto.

20.Realizar un programa que dibuje 1 cuadrado en cada esquina de la pantalla.

21.Realizar un programa que dibuje 3 circulos y rellenarlos de colores diferentes.

22.Realizar un programa que dibujo la función tangente.

23.Realizar un programa que dibuje los ejes coordenados.

24. En base a los siguientes datos relizar una gráfica de barras:

EDAD	ESTATURA
18	1.65
19	1.68
20	1.70
18	1.45
21	1.70
20	1.75
20	1.67
18	1.65
19	1.70
17	1.45

25. Realizar una gráfica de pastel con los siguientes datos:

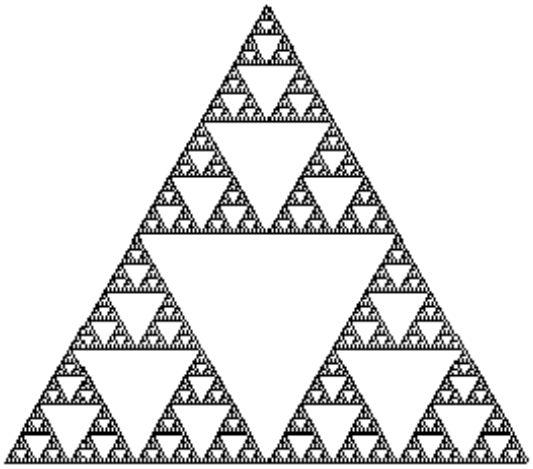
ELECCIONES PRESIDENCIALES

Partido	Porcentaje
PAN	**32%**
PRD	**31%**
PRI	**18%**
NUEVA ALIANZA	**7%**

DR. SIMI	12%

6.33 APLICACIONES DE PROGRAMACIÓN

6.33.1 Fractales



Fractales

La matemática tiene un lado artístico escondido este lado desconocido para muchos, es la geometría fractal, que recientemente pudo ser descubierta, casi casualmente con la ayuda de la computadora. Y pues de otro modo no podía ser ya que solo programando unas sencillas formulas matemáticas obtendrás unas imágenes de belleza infinita.

Historia

Los fractales fueron concebidos aproximadamente en 1890 por el francés Henri Poincaré. Sus ideas fueron extendidas más tarde fundamentalmente por dos matemáticos también franceses, Gastón Julia y Pierre Fatou, hacia 1918. Se trabajó mucho en este campo durante varios años, pero el estudio quedó congelado en los años 20. El estudio fue renovado a partir de 1974 en IBM y fue fuertemente impulsado por el desarrollo de la computadora digital.

El Dr. Mandelbrot, de la Universidad de Yale, con sus experimentos de computadora, es considerado como el padre de la geometría fractal. En honor a él, uno de los conjuntos que él investigó fue nombrado por su nombre.

El matemático francés Benoit Mandelbrot acuñó la palabra fractal en la década de los ’70, derivándola del adjetivo latín fractus. El correspondiente verbo latino: frangere, significa romper, crear fragmentos irregulares.

Son muchas las personas que han aportado su granito de arena, y muchas (en cierto modo infinitas) clases de fórmulas.

El siguiente cuadro muestra alguno de los "hitos" en la historia de las matemáticas no lineales.

Definición

Matemáticamente, el Fractal es una figura geométrica compleja y detallada en estructura a cualquier nivel de magnificación. A menudo los fractales son semejantes a sí mismos; esto es, poseen la propiedad de que cada pequeña porción del fractal, puede ser vizualizada como una réplica del todo a escala reducida.

Existen muchas estructuras matemáticas que son fractales: el triángulo de Sierspinski, la curva de Koch, el conjunto Mandelbrot, los conjuntos Julia, y muchas otras.

La característica que fue decisiva para llamarlos fractales es su dimensión fraccionaria. No tienen dimensión uno, dos o tres como la mayoría de los objetos a los cuales estamos acostumbrados.

Los fractales tienen usualmente una dimensión que no es entera, ejemplo: 1,55.

Es importante reconocer que los fractales verdaderos son una idealización. Ninguna curva en el mundo real es un fractal verdadero; los objetos reales son

producidos por procesos que sólo actúan sobre un rango de escalas finitas. En otras palabras, los objetos reales no tienen la infinita cantidad de detalles que los fractales ofrecen con un cierto grado de magnificación.

Tipos de fractales

Algoritmos de escape

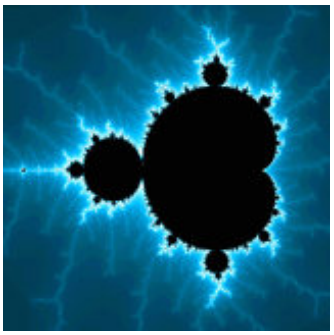
Por ejemplo, fractal de Mandelbrot se genera mediante un algoritmo de escape. Para cada punto se calculan una serie de valores mediante la repetición de una formula hasta que se cumple una condición, momento en el cual se asigna al punto un color relacionado con el número de repeticiones. Los fractales de este tipo precisan de millones de operaciones, por lo cual sólo pueden dibujarse con la inestimable ayuda del ordenador.

$$Z_n = Z_{n-1}^2 + C$$

Esa es la fórmula: z es la variable y c el valor de las coordenadas del punto analizado. Con cada punto, z comienza siendo (0,0), y se va aplicando reiteradamente esa fórmula. Si el módulo de z se hace en algún momento mayor que 2, significará que el punto no pertenece al conjunto de Mandelbrot. Dicho de otra forma, Mandelbrot es el conjunto de puntos cuya órbita generada con la fórmula dada nunca escapa de un círculo de radio 2.

En lenguaje parser, esto se traduce en lo siguiente:

```
Mandelbrot {
z = 0, c = Pixel:
z = z^2 + c
|z| <= 4
}
```



Mandelbrot

Funciones iteradas

El sistema de funciones iteradas (IFS) es un método creado por M. Barnsley, basándose en el principio de autosemejanza. En un fractal IFS siempre se puede encontrar una parte de la figura que guarda una relación de semejanza con la figura completa. Esa relación es a menudo muy difícil de apreciar, pero en el caso del helecho es bastante clara: cualquier hoja es una réplica exacta de la figura completa.

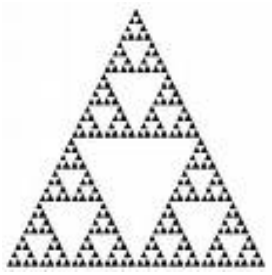
Helecho de Barnsley

Lindenmayer y Sierpinski

La idea es sencilla y antigua. Un triángulo en el que se aloja otro, uniendo los puntos medios de cada uno de sus lados. Esto se repite con todos y cada uno de los triángulos formados que tengan la misma orientación que el original, y así

sucesivamente. Quizá se pueda explicar de otra forma, pero lo mejor es verlo en la animación de la figura 3.

El triángulo de Sierpinski es uno de los pocos fractales que se puede dibujar con exactitud sin ayuda de un ordenador, siguiendo las instrucciones anteriores.



Codigo Fuente (Visual Basic)

Mandelbrot

```
Option Explicit
Dim zr As Double
Dim zi As Double
Dim zar As Double
Dim zir As Double
Dim ci As Double
Dim cr As Double
Dim otropunto As Double
Const Fecha_comp = "15-08-04"
Const Nombre_aplicacion = "El fractal de Mandelbrot"

Private Sub Command1_Click ( )
Call Text1_Change
```

```
Dibuja (Val (Text1.Text))
End Sub

Private Sub Form_Load ( )
Picture1.Scale (-2, 2)-(2, -2)

otro punto = 0.01
End Sub

Public Sub Dibuja (iteraciones As Integer)
'ProgressBar3.Max = iteraciones
'ProgressBar3.Min = 0
'ProgressBar3.Value = 0
Picture1.Cls
Command1.Enabled = False

Dim ite As Integer
Me.Caption = "Fractal de Mandelbrot - Dibujando"
Dim X As Double
Dim Y As Double
X = -2
Y = X
While X <= 2
  cr = X
   While Y <= 2
      ite = 0
      ci = Y

      zr = 0
      zi = 0
```

```
        zar = 0
        zir = 0
        While ite < iteraciones And (zr ^ 2) + (zi ^ 2) < 4

           zr = (zar * zar) - (zir * zir) + cr
           zi = (zar * zir) + (zir * zar) + ci
           zar = zr
           zir = zi
           'ProgressBar3.Value = ite
           ite = ite + 1

        Wend
        If Option1 And (zr ^ 2) + (zi ^ 2) < 4 Then
        Picture1.PSet (X, Y)
        Else
           If Option2 Then
              Picture1.PSet (X, Y), QBColor (ite Mod 16)
           Else
           If Option3 And ite < 100 Then Picture1.PSet (X, Y), ite * ite
           If Option7 Then Picture1.PSet (X, Y), RGB (Int (Abs (zr)) * 100, 0, ite)
           If Option8 Then Picture1.PSet (X, Y), RGB (Int (Abs (zr)) * 100, 0, Abs (ite
Mod 16))
           If Option9 Then Picture1.PSet (X, Y), RGB (Int (Abs(zr)) * 100, Int (Abs (zi))
* 100, Abs(ite Mod 16))
          If Option10 Then Picture1.PSet (X, Y), RGB (Int (Abs (zr)) * 100, Int (Abs
(zr)) * 100, Int (Abs (zr)) * 80)
           End If
        End If

     Y = Y + otropunto
     DoEvents
```

```
    ProgressBar2.Value = (Int (Y) + 2) * 100 / 4
    Wend
    Y = -2

X = X + otropunto
ProgressBar1.Value = (Int(X) + 2) * 100 / 4

Wend

Me.Caption = "Fractal de Mandelbrot"

Command1.Enabled = True

End Sub
Private Sub Check1_Click ( )
If Check1.Value = 1 Then
linex.X1 = -2
linex.Y1 = 0#
linex.X2 = 2
linex.Y2 = 0#
liney.X1 = 0#
liney.Y1 = -2
liney.X2 = 0#
liney.Y2 = 2
linex.Visible = True
liney.Visible = True
puntov.Visible = True
puntoh.Visible = True
Else
linex.Visible = False
liney.Visible = False
```

```
puntov.Visible = False
puntoh.Visible = False
End If

End Sub

Private Sub Form_Unload (Cancel As Integer)
MsgBox Nombre_aplicacion & vbCrLf & "Programado por Yango para el 'treball de recerca'" & vbCrLf & "http://usuarios.lycos.es/sisar" & vbCrLf & vbCrLf & "v: " & App.Major & "." & App.Minor & "." & App.Revision & vbCrLf & Fecha_comp, vbSystemModal, Nombre_aplicacion.

End

End Sub

Private Sub Option4_Click ( )
otropunto = 0.1
End Sub

Private Sub Option5_Click ( )
otropunto = 0.01
End Sub
Private Sub Option6_Click ( )
otropunto = 0.001
MsgBox "El proceso de dibujo puede ser muy lento", vbInformation, "Advertencia"
End Sub
Private Sub Picture1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
'Me.Caption = X & ", " & Y
If Y < 0 Then
```

```
Label4.Caption = "Punto donde se encuentra el ratón: (" & Format (X, "00.00") & "-
" & Format (Abs (Y), "00.00i") & ")"
Else
Label4.Caption = "Punto donde se encuentra el ratón: (" & Format(X, "00.00") & "+"
& Format(Y, "00.00i") & ")"
End If
If Check1.Value = 1 Then
puntov.X1 = X
puntov.X2 = X
puntov.Y1 = 0
puntov.Y2 = Y
puntoh.Y1 = Y
puntoh.Y2 = Y
puntoh.X1 = 0
puntoh.X2 = X
puntov.Visible = True
puntoh.Visible = True
Else
puntov.Visible = False
puntoh.Visible = False

End If
End Sub
Private Sub Text1_Change ( )
If Val (Text1.Text) <= 100 Then
Option3.Enabled = True
Else
Option3.Enabled = False
End If
End Sub
```

EJERCICIOS RESUELTOS EN C

EJERCICIOS CON OPERACIONES SIMPLES

- ✓ Dados dos números, hacer un programa que haga la suma de ellos.

1)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int a,b,c;
    clrscr ( );
    a=5;
    b=8;
    c=a+b;
      printf ("%d",c);
   getch ( );
}
```

2)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
     int a,b,c;
     clrscr ( );
     printf (“Dame el valor del primer número “);
     scanf (%d,&a);
     printf (“Dame el valor del segundo número “);
     scanf (%d,&b);
     c=a+b;
       printf ("%d",c);
```

```
    getch ( );
}
```

- ✓ Dados dos números, hacer un programa que haga la multiplicación.

1)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int a,b,c;
    clrscr ( );
    a=5;
    b=8;
    c=a*b;
       printf ("%d",c);
    getch ( );
}
```

2)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int a,b,c;
    clrscr ( );
    printf (“Dame el valor del primer número “);
    scanf (%d,&a);
    printf (“Dame el valor del segundo número “);
    scanf (%d,&b);
    c=a*b;
      printf ("%d",c);
    getch ( );
```

```
}
```

- ✓ Dados dos números, hacer un programa que haga la división.

1)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
   int a,b;
   float c;
   clrscr ( );
   a=40;
   b=5;
   c=a/b;
      printf ("%f",c);
   getch ( );
}
```

2)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int a,b;
    float c;
    clrscr ( );
    printf (“Dame el valor del primer número “);
    scanf (%d,&a);
    printf (“Dame el valor del segundo número “);
    scanf (%d,&b);
    c=a/b;
       printf ("%f",c);
```

```
    getch ( );
}
```

✓ Dada la longitud de un lado de un cuadrado, obtener su área.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int a,b;
    clrscr ( );
    a=5;
    b=a*a;
      printf ("El area es %d",b);
   getch ( );
}
```

2)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int a,b;
    clrscr ( );
    printf (“Dame el valor de la longitud del lado “);
    scanf (%d,&a);
    b=a*a;
        printf ("El area es %d",b);
   getch ( );
}
```

✓ Dada la longitud de un lado de un cubo, obtener su volumen.

1)

```
#include<stdio.h>
#include<conio.h>
```

```
void main ( )
{
    int a,b;
    clrscr ( );
    a=5;
    b=a*a*a;
       printf ("El volumen es %d",b);
    getch ( );
}
```

2)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int a,b;
    clrscr ( );
    printf (“Dame el valor de la longitud del lado “);
    scanf (%d,&a);
     b=a*a*a;
       printf ("El volumen es %d",b);
    getch ( );
}
```

✓ Dadas las cantidades en grados centígrados, dólares, litros y metros, convertir a grados Fahrenheit, pesos, galones y pulgadas respectivamente.

1)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    float c,d,l,m,f,n,g,p;
    clrscr ( );
```

```
    c=29;
    d=1000;
    l=30;
    m=50;
     f=(c*1.8)+32;
     n=d*12;
     g=l/3.7854;
     p=(m*10)/0.26;
      printf ("øF= %f \n pesos= %f \n galones= %f \n pulgadas= %f",f,n,g,p);
    getch ( );
}
2)
#include<stdio.h>
#include<conio.h>
void main ( )
{
    float c,d,l,m,f,n,g,p;
    clrscr ( );
     printf ("Dame el valor de los grados centigrados ");
     scanf ("%f",&c);
     printf ("Dame la cantidad de dolares ");
     scanf ("%f",&d);
     printf ("Dame la cantidad de litros ");
     scanf ("%f",&l);
     printf ("Dame el numero de metros ");
     scanf ("%f",&m);
       f=(c*1.8)+32;
       n=d*12;
       g=l/3.7854;
       p=(m*10)/0.26;
        printf ("øF= %f \n pesos= %f \n galones= %f \n pulgadas= %f",f,n,g,p);
```

```
    getch ( );
}
```

- ✓ Dados dos números complejos, obtener su suma y su producto.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    float a,b,c,d,m,n,s,t;
    clrscr ( );
      printf ("Dame la parte no imaginaria del primer numero a= ");
      scanf ("%f",&a);
      printf ("Dame el valor de la parte imaginaria del primer numero b= ");
      scanf ("%f",&b);
      printf ("Dame el valor de la parte no imaginaria del segundo numero c=
");
      scanf ("%f",&c);
      printf ("Dame el valor de la parte no imaginariadel segundo numero d=
");
      scanf ("%f",&d);
        m=a+c;
        n=b+d;
        s=(a*c)-(b*d);
        t=(a*d)+(b*c);
          printf ("El numero que forma la suma de los imaginarios es %f+%fi
\n",m,n);
          printf ("El numero que forma la multiplicacion de los imaginarios es
%f+%fi",s,t);
    getch ( );
}
```

- ✓ Progama que dice si es un número par o impar

```
#include <stdio.h>
```

```
#include <conio.h>
void main ( )
{
  int x,d;
  clrscr ( );
  printf ("/*programa para saber si un n£mero es par o impar*/|n|n");
    printf ("dame el valor de x=");
    scanf ("%d",& x);
     d=x%2;
      if (d==0)
      {
         printf ("%d es par",x);
      }
      else
      {
         printf ("%d es impar",x);
      }
  getch ( );
}
```

✓ Dado 5 números sacarsu promedio

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
  float a,b,c,d,e,p;
  clrscr ( );
    printf ("Dame el primer valor a=");
    scanf ("%f",& a);
    printf ("Dame el segundo valor b=");
    scanf ("%f",& b);
    printf ("Dame el tercer valor c=");
```

```
      scanf ("%f",& c);
      printf ("Dame el cuarto valor d=");
      scanf ("%f",& d);
      printf ("Dame el quinto valor e=");
      scanf ("%f",& e);
        p=(a+b+c+d+e)/5;
          printf ("el Promedio es %f",p);
   getch ( );
}
```

✓ Programa dela tabla del 5.

```
#include <stdio.h>
#include <conio.h>

void main ( )
{
  int a,b;
  clrscr ( );
   a=0;
   b=0;
   inicio: printf ("5x%d=%d",a,b);
    if (a<10)
    {
       a=a+1;
       b=5*a;
       goto inicio;
    }
    else
    {
       printf ("fin del programa");
     }
  getch ( );
```

```
}
```

EJERCICIOS CON RAÍCES

- ✓ Dados dos puntos en $\mathbb{R}^2$, obtener su distancia:

1)

```
#include<stdio.h>
#include<conio.h>
#include<fdist.h>
void main ( )
{
    int x,y,z,w;
    float a,b,c;
    clrscr ( );
      x=5;
      y=6;
      z=2;
      w=8;
        a=(x-z)^2;
        b=(y-w)^2;
        c=sqrt (a+b);
          printf ("%f",c);
     getch ( );
}
```

2)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
     int x,y,z,w;
     float a,b,c;
```

```
    clrscr ( );
      printf ("Dame la 1er. coordenada del punto 1 ");
      scanf ("%d",&x);
      printf ("Dame la 2da. coordenada del punto 1 ");
      scanf ("%d",&y);
      printf ("Dame la 1er. coordenada del punto 2 ");
      scanf ("%d",&z);
      printf ("Dame la 2da. coordenada del punto 2 ");
      scanf ("%d",&w);
        a=(x-z) ^2;
        b= (y-w) ^2;
        c=sqrt (a+b);
          printf ("%f",c);
    getch ( );
}
```

EJERCICIOS CON CICLO IF-ELSE

- ✓ Crear un programa que imprima los números del 1 al 10, en lista.

1)

```
#include<stdio.h>
#include<conio.h>
int main ( )
{
    int a;
    a=0;
    clrscr ( );
    inicio : printf ("%d \n",a);
      if (a<10)
      {
        a++;
```

```
            goto inicio;
          }
          else
          {
            printf ("FIN DE PROGRAMA");
           }
       getch ( );
}
```

2)

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
    int a;
    clrscr ( );
      a=0;
      pudin:printf ("%d\n",a++);
        if (a>10)
        {
          printf ("fin del programa");
        }
        else
        {
          goto pudin;
        }
   getch ( );
}
```

✓ Crear un programa que imprima los números de 5 en 5 hasta 100 en lista.

1)

```
#include<stdio.h>
#include<conio.h>
```

```
int main ( )
{
      int a;
      a=0;
      clrscr ( );
      inicio : printf ("%d \n",a);
        if (a<100)
         {
            a=a+5;
            goto inicio;
         }
         else
         {
           printf ("FIN DE PROGRAMA");
         }
     getch ( );
}
```

2)

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
    int a;
    clrscr ( );
    a=5;
      pudin:printf ("%d\n",a);
      a=a+5;
        if (a>100)
        {
            printf ("fin del programa");
         }
```

```
        else
        {
            goto pudin;
        }
    getch ( );
}
```

✓ Crear un programa que imprima la tabla del 5.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int b,c;
    clrscr ( );
     c=0;
     b=0;
      inicio : printf ("5*%d=%d \n",c,b);
       if (c<10)
       {
          c=c+1;
          b=5*c;
          goto inicio;
       }
       else
       {
          printf ("FIN DE PROGRAMA");
       }
     getch ( );
}
```

✓ Crear un programa que haga la suma de n números.

```
#include<stdio.h>
#include<conio.h>
```

```
void main ( )
{
     float a,b,n;
     a=0;
     b=0;
     clrscr ( );
       printf ("¨Cuántos numeros quieres sumar?");
       scanf ("%f",&n);
       inicio : b=b+a;
         a=a+1;
         if (a<=n)
         {
            goto inicio;
         }
         else
         {
            printf ("La suma es %f",b);
         }
     getch ( );
}
```

2)

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
  int a,n,s;
  clrscr ( );
    printf ("asta q n£mero qieres leer n= ");
    scanf ("%d",& n);
    s=0;
    a=0;
```

```
pudin:
a++;
s=s+a;
 if (a==n)
 {
    printf ("suma = %d",s);
 }
 else
 {
    goto pudin;
 }
getch ( );
}
```

✓ Crear un programa que realice el producto de n números.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    float a,b,n;
    a=1;
    b=1;
    clrscr ( );
       printf ("¨Cua ntos numeros quieres multiplicar?");
       scanf ("%f",&n);
        inicio : b=b*a;
        a=a+1;
         if (a<=n)
         {
            goto inicio;
          }
          else
```

```
            {
                printf ("La multiplicaci¢n es %f",b);
            }
        getch ( );
}
```

2)

```
#include <conio.h>
void main ( )
{
  int a,b,c,n,s;
  clrscr ( );
    printf ("asta q n£mero leeras n= ");
    scanf ("%d",& n);
    a=0;
    c=0;
   b=1;
    pudin:
     if (a<n)
     {
       a++;
       c++;
       b=b*c;
       goto pudin;
     }
     else
     {
       printf ("multiplicacion = %d",b);
     }
  getch ( );
}
```

✓ Crear un programa que calcule el factorial de un número.

1)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
     int a,b,c;
     clrscr ( );
       b=1;
       c=1;
       printf ("dame el valor de un numero entero a=");
       scanf ("%d",&a);
       inicio:b=b*c;
         if (c<a)
         {
              c=c+1;
              goto inicio;
         }
         else
         {
              printf ("%d",b);
         }
      getch ( );
}
```

2)

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
   int x,n,a,b,c;
   clrscr ( );
   printf ("*/PROGRAMA PARA SACAR FACTORIAL/*\n\n");
```

```
    printf ("dame el n£mero x=");
    scanf ("%d",& x);
    c=1;
     if (x==0)
     {
        printf ("%d es el factorial",c);
     }
     else
     {
        a=1;
        b=1;
        c=0;
        inicio: printf ("%d!=%d\n",b,a);
        c=c+1;
        b=b+1;
        a=b*a;
          if (c==x)
          {
             printf ("Fin del programa");
          }
          else
          {
             goto inicio;
          }
       }
   getch ( );
}
```

✓ Dados dos números imprimir cuál es el mayor, el menor, o si son iguales.

```
#include <stdio.h>
#include <conio.h>
void main ( )
```

```
{
    float a,b;
    clrscr ( );
      printf ("Ingresa el primer número a=");
      scanf ("%f",&a);
      printf ("Ingresa el segundo número b=");
      scanf ("%g",&b);
        if(a>b)
        {
      printf ("el número %f es el mayor y el número %g es el menor",a,b);
        }
        else
        {
              if(a<b)
              {
       printf ("el número %g es el mayor y el número %f es menor",b,a);
              }
              else
              {
                    printf ("los números son iguales");
              }
         }
      getch ( );
}
```

✓ Dado un número, imprimir si es mayor, menor o igual a cero.

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
      float a;
      clrscr ( );
```

```
        printf ("Dame un número a=");
        scanf ("%f",&a);
          if(a>0)
          {
               printf ("El número es mayor que cero");
           }
           else
           {
                if(a<0)
                {
                       printf ("El número es menor que cero");
                }
                else
                {
                        printf ("El número es cero");
                }
           }

     getch ( );
}
```

✓ Dados 3 números imprimir el mayor y el menor.

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
     float a,b,c;
     clrscr ( );
         printf ("Dame el primer n£mero a=");
         scanf ("%f",a);
         printf ("Dame el segundo n£mero b=");
         scanf ("%g",b);
```

```
printf ("Dame el tercer n£mero c=");
scanf ("%h",c);
   if(a>b)
   {
        if(a>c)
        {
          if(b>c)
          {
printf ("el número %f es el mayor y el número %h es el menor",a,c);
           }
           else
           {
printf ("el número %f es el mayor y el número %g es el menor",a,b);
            }
        }
       else
   {
printf ("el número %h es el mayor y el número %g es el menor",c,b);
       }
   }
   else
   {
        if(b>c)
        {
            if(a>c)
            {
printf ("el número %g es el mayor y el número %h es el menor",b,c);
            }
            else
            {
printf ("el número %g es el mayor y el número %f es el menor",b,a);
```

```
                    }
                  }
                   else
                  {
      printf ("el número %h es el mayor y el número %f es el menor",c,a);
                  }
               }
getch ( );
}
```

✓ Crear un programa que calcule el valor absoluto de un número.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
      int a,b;
      clrscr ( );
        printf ("dame el valor de un numero a=");
        scanf ("%d",&a);
         if (a<0)
         {
             b=-a;
             printf ("%d",b);
         }
         else
         {
             printf ("%d",a);
          }
      getch ( );
}
```

SERIES CON EL CICLO IF

✓ ***SERIE 1: 1-(1/5) + (1/10)-(1/15) + (1/20)-…***

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
      int a,b,c,d,e,f;
      float s,t,r;
      clrscr ( );
 printf ("*/PROGRAMA PARA CONOCER LA SUMA DE LA SERIE/*\n\n");
         printf ("¿cuantos números quieres leer? n=");
         scanf ("%d",& f);
         b=1;
         a=1;
         c=0;
         s=1;
         serie: printf ("1/%d  ",a);
           a=5*b;
           b=b+1;
           c=c+1;
             if (a%2==0)
             {
               t=a;
               s=s+(1/t);

               if (c==f)
               {
                  r=s-(1/t);
                  printf ("fin del programa la suma es %f",r);
               }
               else
```

```
                {
                    goto serie;
                }
            }
            else
            {
                a=-a;
                t=a;
                s=s+(1/t);
                  if (c==f)
                  {
                      r=s-(1/t);
                      printf ("fin del programa la suma es %f",r);
                  }
                  else
                  {
                       goto serie;
                  }
            }
        getch ( );
}
```

- ✓ **SERIE 2: (3/5)-(5/7)+(7/9)+(11/13)-…**

```
 #include <stdio.h>
 #include <conio.h>
void main ( )
{
    int a,b,c,d,e,n;
    float s,r,t,h,i,j;
    clrscr ( );
 printf ("*/PROGRAMA PARA SACAR SUMA DE LA SERIE Y SU
SUCESION/*\n\n");
```

```
printf ("¿cuantos números de la serie quieres? n=");
scanf ("%d",& n);
a=1;
b=3;
c=0;
 i=a;
 j=b;
 s=i/j;
   serie: printf ("%d/%d  ",a,b);
   a=b;
   b=b+2;
   c=c+1;
    if (c%2==1)
    {
        a=-a;
        t=a;
        h=b;
        s=s+(t/h);
          if (c==n)
          {
              r=s-(t/h);
              printf ("fin de la serie la suma de ella es %f",r);
          }
          else
          {
              goto serie;
          }
      }
      else
      {
            t=a;
```

```
                h=b;
                 s=s+(t/h);
                    if (c==n)
                    {
                        r=s-(t/h);
                        printf ("fin de la serie la suma de ella es %f",r);
                    }
                    else
                    {
                        goto serie;
                    }
            }
    getch ( );
}
```

✓ **SERIE 3:**

```
 #include <stdio.h>
 #include <conio.h>
 #include <math.h>
void main ( )
{
     int a,b,c,d,e,f,n;
     float s,r,t,h,i,j;
     clrscr ( );
       printf ("¿cuantos números de la serie quieres? n=");
       scanf ("%d",& n);
       printf ("1");
       a=3;
       b=0;
       c=2;
         serie: printf ("(%d^%d)/%d",c,a);
         a=a+2;
```

```
        b=b+1;
          if (b%2==1)
          {
                goto serie;
          }
    getch ( );
}
```

CADENAS DE CARACTERES CON CICLOS

- ✓ Crear un programa que al teclear caracteres imprima la cantidad de ellos.

1)

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
      char s [30];
      int c;
      clrscr ( );
        printf ("Teclea la cadena de caracteres");
        gets (s);
        c=0;
          inicio:
            if (s[c]=='\0')
            {
                  printf ("%d",c);
            }
            else
            {
                  c++;
                  goto inicio;
```

```
            }
      getch ( );
}
```

2)

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
   int n;
   char pal [30];
   clrscr ( );
   printf ("*/PROGRAMA PARA CONOCER LA LONGITUD DE UNA
ORACION/*\n\n");
     printf ("dame la oraci¢n  ");
     gets (pal);
       for (n=0;pal[n]!='\0';n=n+1)
       {
       }
       printf ("longitud= %d",n);
  getch ( );
}
```

✓ Crear un programa que al teclear un carácter imprima si es letra, número o símbolo.

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
   char x;
   clrscr ( );
      printf ("*/PROGRAMA QUE LEE SI ES NUMERO, SIMBOLO O
PALABRA/*\n\n");
```

```
printf ("introduce el caracter=");
scanf ("%c",& x);
 if (x>47 && x<58)
 {
     printf ("%c es un n£mero",x);
 }
 else
 {
     if (x>64 && x<123)
     {
        if (x<91)
        {
            printf ("es una palabra");
         }
         else
         {
            if (x<97)
            {
                printf ("%c es un simbolo",x);
            }
            else
            {
                 printf ("es una palabra");
             }
          }
       }
       else
       {
           printf ("%c es un simbolo",x);
       }
 }
```

```
    getch ( );
}
```

✓ Crear un programa que concatene dos cadenas de caracteres.

CICLO WHILE

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
   int n,m;
   char s[30],t[30];
   clrscr ( );
     printf ("Teclea los primeros caracteres ");
     gets (s);
     printf ("Teclea los siguientes caracteres ");
     gets (t);
     n=0;
     m=0;
       while (s[n]!='\0')
       {
            printf ("%c",s[n]);
            n++;
       }
       while (t[m]!='\0')
       {
             n++;
             s[n]=t[m];
             m++;
             printf("%c",s[n]);
        }
    getch ( );
}
```

CICLO FOR

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
   int n,m;
   char s[30],t[30];
   clrscr ( );
      printf ("Teclea los primeros caracteres ");
      gets (s);
      printf ("Teclea los siguientes caracteres ");
      gets (t);
        for (n=0;s[n]!='\0';n++)
        {
            printf ("%c",s[n]);
         }
        for (m=0;t[m]!='\0';m++)
        {
             n++;
             s[n]=t[n];
             printf ("%c",s[n]);
         }
    getch ( );
}
```

CICLO DO-WHILE

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int n,m;
    char s[30],t[30];
```

```
    clrscr ( );
      printf ("Teclea los primeros caracteres ");
      gets (s);
      printf ("Teclea los siguientes caracteres ");
      gets (t);
      n=0;
      m=0;
       do
       {
          printf ("%c",s[n]);
          n++;
       } while (s[n]!='\0');
       do
       {
           n++;
           s[n]=t[m];
           m++;
           printf ("%c",s[n]);
        } while (t[m]!='\0');
   getch ( );
}
```

✓ Crear un programa que compare dos cadenas de caracteres e imprima si son iguales o distintas.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int n,m;
    char s[30],t[30];
    clrscr ( );
      printf ("Teclea los primeros caracteres ");
```

```
        gets (s);
        printf ("Teclea los siguientes caracteres ");
        gets (t);
        n=0;
        m=0;
        inicio:
          if (s[n]==t[m])
          {
              if (s[n]=='\0' && t[m]=='\0')
              {
                  printf ("Los caracteres son iguales");
              }
              else
              {
                   n++;
                   m++;
                   goto inicio;
              }
          }
          else
          {
                printf ("Los caracteres son distintos");
          }
       getch ( );
}
```

FUNCIONES

- ✓ Sacar la suma de 2 números.

```
#include <stdio.h>
#include <conio.h>
```

```
int suma (int a,int b)
{
   float c;
   c=a+b;
   return (c);
}
void main ( )
{
   int a,b,c;
   clrscr ( );
     printf ("dame el valor del primer n£mero a=");
     scanf ("%d",&a);
     printf ("dame el valor del egundo n£mero b=");
     scanf ("%d",&b);
       c=suma (a,b);
       printf ("suma=%d",c);
   getch ( );
}
```

✓ Sacar la resta de 2 números.

```
#include <stdio.h>
#include <conio.h>
int res (int a,int b)
{
  int c;
  c=a-b;
  return (c);
}
void main ( )
{
   int a,b,c;
   clrscr ( );
```

```
      printf ("dame el valor del primer n£mero a=");
      scanf ("%d",&a);
      printf ("dame el valor del egundo n£mero b=");
      scanf ("%d",&b);
        c=res (a,b);
          printf ("resta=%d",c);
    getch ( );
}
```

✓ Sacar la multiplicación de 2 números.

```
#include <stdio.h>
#include <conio.h>
int multi (int a,int b)
{
    int c;
    c=a*b;
    return (c);
}
void main ( )
{
    int a,b,c;
    clrscr ( );
      printf ("dame el valor del primer n£mero a=");
      scanf ("%d",&a);
      printf ("dame el valor del segundo n£mero b=");
      scanf ("%d",&b);
        c=multi (a,b);
           printf ("multiplicaci¢n=%d",c);
    getch ( );
}
```

✓ Sacar la división de 2 números.

```
#include <stdio.h>
```

```
#include <conio.h>
float divi (float a,float b)
{
    float c;
    c=a/b;
    return (c);
}
void main ( )
{
    int a,b;
    float c;
    clrscr ( );
      printf ("dame el valor del numerador a=");
      scanf ("%d",&a);
      printf ("dame el valor del denominador b=");
      scanf ("%d",&b);
        c=divi (a,b);
          printf ("division=%f",c);
    getch ( );
}
```

✓ Dado un lado de un cuadrado sacar su área.

```
#include <stdio.h>
#include <conio.h>
int area (int a)
{
  int c;
  c=a*a;
  return (c);
}
void main ( )
{
```

```
  int a,c;
  clrscr ( );
    printf ("dame el valor del lado del cuadrado a=");
    scanf ("%d",&a);
      c=area(a);
      printf ("area=%d",c);
  getch ( );
}
```

- ✓ Dado un lado sacar el volumen de un cubo.

```
#include <stdio.h>
#include <conio.h>
int volum (int a,int b)
{
   int c;
   c=a*a*b;
   return (c);
}
void main ( )
{
    int a,b,c;
    clrscr ( );
      printf ("dame el valor del lado del cuadrado a=");
      scanf("%d",&a);
      printf ("dame el valor de la altura b=");
      scanf ("%d",&b);
        c=volum (a, b);
          printf ("volumen=%d",c);
   getch ( );
}
```

- ✓ Dado un lado sacar el volumen de un tetaedro.

```
#include <stdio.h>
#include <conio.h>
int table (int g,int t)
{
   int h;
   h=g*t;
   return (h);
}
void main ( )
{
    int a,b,c;
    clrscr ( );
      b=1;
      a=13;
      c=13;
        while (b<50)
        {
           printf ("%d, ",c);
           b++;
           c=tabla (a,b);
        }
      printf ("  fin");
    getch ( );
}
```

✓ Tabla del 3.

```
#include <stdio.h>
#include <conio.h>
int table (int g,int t)
{
    int h;
    h=g*t;
```

```
    return (h);
}
void main ( )
{
    int a,b,c;
    clrscr ( );
     b=1;
     a=5;
     c=5;
       do
       {
            printf ("%dx%d=%d\n",a,b,c);
            b=b+3;
            c=tabla (a,b);
        }
        while (b<30);
           printf ("fin");
   getch ( );
}
```

✓ Cambiar de grados a farenheit.

```
#include <stdio.h>
#include <conio.h>
float faracen (float a)
{
  float c;
  c=(a-32)/1.8;
  return (c);
}
void main ( )
{
  float a,c;
```

```
   clrscr ( );
      printf ("dame los fahrenheit a=");
      scanf ("%f",&a);
      c=faracen (a);
     printf ("centigrados=%f",c);
getch ( );
}
```

✓ Convertir de galones a litros.

```
#include <stdio.h>
#include <conio.h>
float gallit (float a)
{
     float c;
     c=a*3.7854;
     return (c);
}
void main ( )
{
     float a,c;
     clrscr ( );
       printf ("dame los galones q son a=");
       scanf ("%f",&a);
         c=gallit (a);
           printf ("litros=%f",c);
     getch ( );
}
```

✓ Dados 2 puntos sacar su distancia.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float dist (int a,int b,int c,int d)
```

```
{
   float e,t,g,f,k;
   t=a-c;
   g=b-d;
   f=t*t+g*g;
   e=sqrt (f);
   return (e);
}
void main ( )
{
   float a,b,c,d,e;
   clrscr ( );
     printf ("dame el valor de x1=");
     scanf ("%f",&a);
     printf ("dame el valor de y1=");
     scanf ("%f",&b);
        printf ("dame el valor de x2=");
      scanf ("%f",&c);
      printf ("dame el valor de y2=");
      scanf ("%f",&d);
         e=dist (a,b,c,d);
           printf ("distancia=%f",e);
  getch ( );
}
```