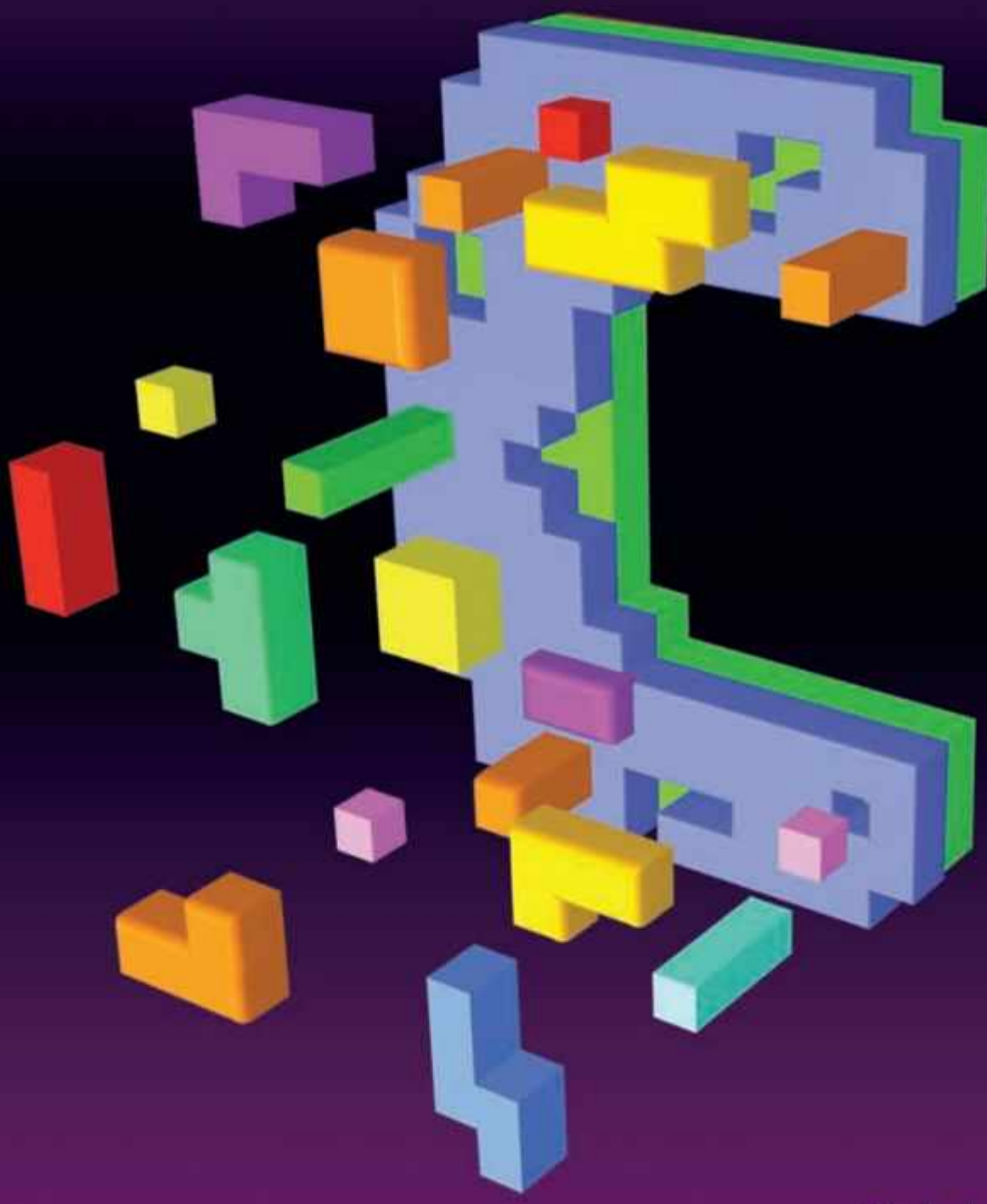
INTRODUCCIÓN A LA PROGRAMACIÓN ESTRUCTURADA EN C



Gabriela Márquez
Sonia Osorio
Noemí Olvera

PEARSON

Introducción a la programación estructurada en C

Primera edición

Teresa Gabriela Márquez Frausto

Sonia Osorio Ángel

Elzie Noemí Olvera Pérez

Centro Universitario de Ciencias Exactas
e Ingenierías (CUCEI)
Universidad de Guadalajara

Revisión técnica

Luis Arturo Jiménez Mendoza

Escuela Superior de Ingeniería Mecánica
y Eléctrica (ESIME), Zacatenco
Instituto Politecnico Nacional

Prentice Hall

México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

Datos de catalogación bibliográfica

Márquez Frausto, Teresa Gabriela;
Osorio Ángel, Sonia; Olvera Pérez, Elzie Noemí

Introducción a la programación estructurada en C.
Primera edición

PEARSON EDUCACIÓN, México, 2011

ISBN: 978-607-32-0600-6
Área: Computación

Formato: 18.5 × 23.5 cm Páginas: 376

Edición en español

Editor: Luis Miguel Cruz Castillo
luis.cruz@pearson.com
Editor de desarrollo: Bernardino Gutiérrez Hernández
Supervisor de producción: Enrique Trejo Hernández

PRIMERA EDICIÓN, 2011

ISBN 978-607-32-0600-6
ISBN e-book 978-607-32-0601-3
ISBN e-chapter 978-607-32-0602-0

Impreso en México. *Printed in Mexico*.
1 2 3 4 5 6 7 8 9 0 - 14 13 12 11

Prentice Hall
es una marca de

PEARSON

www.pearsoneducacion.net **ISBN: 978-607-32-0600-6**

Dedicado a Gabriel

CONTENIDO

PRESENTACIÓN

Este libro fue creado teniendo en mente aquellos alumnos que desean aprender a programar utilizando el lenguaje de C a través de un sistema muy práctico. Resultará muy útil para estudiantes con poca o ninguna experiencia, pero con iniciativa para resolver problemas mediante el uso de un lenguaje que "entienda" la computadora.

El contenido del libro está considerado para desarrollarse totalmente en un primer curso de programación; cada tema se trata con suficiente profundidad y detalle para ser comprendido por alguien que se inicia el conocimiento en esta disciplina.

El método de enseñanza se basa en el análisis de un problema y la descripción de los pasos necesarios para llegar a la solución. El objetivo principal es mostrar con detalle cómo crear un programa y cómo entenderlo. Todo esto se complementa con ejemplos y ejercicios resueltos, desglosados en cinco partes para una mejor comprensión. También se presenta la descripción de las operaciones, los datos, la codificación y la ejecución del código, además de una explicación detallada del procedimiento.

El libro se compone de nueve capítulos. El capítulo 1 habla de los elementos de un programa; el capítulo 2 describe las operaciones aritméticas que se pueden realizar y el uso de operadores lógicos relacionales, y se presentan algunos ejemplos. En el capítulo 3 se muestran los primeros programas con una estructura

de control (secuenciación) y se mencionan las bibliotecas de uso común.
El capítulo 4 describe las estructuras de control selectivas y sus diferentes formas de aplicarlas: simple, doble, con anidamientos y múltiple. El capítulo 5 contiene las estructuras de control repetitivas `while`, `do-while` y `for`, y se indica en qué caso es preferible usar cada una. El capítulo 6 presenta los datos estructurados tipo `array` y muestra la diferencia entre arreglos unidimensionales y bidimensionales, y arreglos de caracteres; el capítulo 7 describe el dato estructurado `struct` como aquel que puede contener diferentes tipos de datos, además de una introducción al manejo de apuntadores. El capítulo 8 describe las funciones, mostrando por separado las que devuelven un valor de aquellas que al ser invocadas requieren información (parámetros). El capítulo 9 se compone de un conjunto de ejercicios agrupados en estructuras de control, estructuras de datos y funciones. Estas aplicaciones están orientadas a diferentes carreras de ingeniería.

Finalmente, se anexa un listado de todos los ejercicios y de los ejemplos que se incluyen en el libro.

CAPÍTULO 1

Introducción al lenguaje C

Este libro aborda el desarrollo de programas en lenguaje de programación C. Está planeado para utilizarse en un curso de un semestre de tres horas por semana. En este capítulo repasaremos primero los conceptos de las ciencias de la computación, para describir después los elementos de un programa escrito en lenguaje C, los datos, así como instrucciones de entrada y salida de los mismos.

1.1 Conceptos

Una **computadora** es un dispositivo electrónico que procesa datos (un programa representado en código de máquina). La computadora está formada por algún tipo de dispositivo de entrada (comúnmente

el teclado), uno o varios de salida (como un monitor), la unidad central de procesamiento (CPU, *central processing unit*) y chips de memoria conocida como memoria interna, principal o RAM (*random access memory*).

Es importante recordar que la computadora funciona con al menos dos tipos de memoria: la interna, principal o RAM, y algún dispositivo de memoria externa (o secundaria). La primera es a la que el procesador accede para realizar sus operaciones; la segunda se emplea para almacenar archivos de manera permanente.

En la memoria interna se cargan los datos y el programa en lenguaje máquina que ejecutará la CPU (al programa que está en estado de ejecución se le llama **proceso**). La memoria externa o secundaria es cualquier dispositivo de almacenamiento —discos flexibles, CD, DVD, cintas magnéticas, memoria flash, BlueRay, etcétera— en el que es posible almacenar archivos.

1.2 Lenguajes utilizados en una computadora

En una computadora se utilizan diferentes tipos de lenguajes, algunos empleados directamente por el usuario, y otros que sólo entiende la computadora. Los primeros se conocen como lenguajes de alto nivel, y se usan para realizar la programación de un sistema operativo o de un simple programa que haga una cuenta; los otros los emplea directamente la computadora, ya que están en un lenguaje incomprensible para los humanos. A continuación se describen.

Los lenguajes de alto nivel generalmente se conocen como lenguajes de programación. Con ellos se "redacta" en un lenguaje similar al nuestro, aunque la mayoría se encuentran en inglés. Tienen la ventaja de ser portables, es decir, pueden entenderse y ejecutarse en diversas computadoras; por otro lado, estos lenguajes pueden ser compiladores o intérpretes.

Un **intérprete** toma una instrucción del programa, la traduce a lenguaje máquina y la ejecuta. Este proceso se repite con cada una de las instrucciones del programa; el intérprete más popular es Basic.

Un **compilador** toma las instrucciones escritas en un lenguaje de alto nivel, y las traduce a lenguaje intermedio (código objeto), que después se traduce a lenguaje máquina creando un archivo que contiene el programa ejecutable. El lenguaje C es un compilador.

El **lenguaje ensamblador**, o simplemente ensamblador, es un lenguaje de bajo nivel que se programa según las instrucciones que tiene definidas el procesador.

El **lenguaje máquina** es un programa escrito en ceros y unos, muy lejano a nuestra forma de expresión, pero es el único que entiende el procesador.

Un **programa** es un conjunto de instrucciones que tiene un objetivo específico. Las instrucciones están escritas usualmente en algún lenguaje de computadora; después, el programa se traduce a código objeto y finalmente a lenguaje máquina, que es el único que la computadora entiende; éste se carga en la memoria principal de la computadora para ser ejecutado por la CPU; el resultado son las acciones para lo que fue escrito.

Hay dos grandes tipos de programas: los de aplicación y los de sistemas. Los de aplicación son los más conocidos y se utilizan directamente en la computadora como los procesadores de palabras y las hojas de cálculo. Los de sistemas son un conjunto de programas que permiten que exista una interfaz de comunicación amigable entre el usuario y la computadora, algunos ejemplos son: los sistemas operativos, los compiladores, etcétera.

Programador es el término utilizado para denominar a la persona dedicada a escribir programas. Frecuentemente, una aplicación o sistema se elabora por equipos enteros de programadores.

Se denomina **software** a los programas utilizados en una computadora.

1.3 Etapas de desarrollo del software

Por lo general, el desarrollo o construcción de un programa abarca las siguientes etapas:

Análisis
Diseño
Programación
Codificación
Prueba
Mantenimiento
Documentación

El *análisis* y *diseño* dependen del tamaño y la finalidad del programa; la *programación* es escribir la solución propuesta en el idioma nativo, la *codificación* consiste en escribir el programa con instrucciones en un lenguaje de programación, en

este caso en C, considerando el diseño (módulos, algoritmos, diagramas). En ocasiones se emplea indistintamente *codificación* y *programación* para designar esta fase del proceso. Posteriormente se compila y prueba el código; se resuelven los errores de sintaxis, en caso de existir, y de lo contrario se procede a la ejecución del programa para probarlo y verificar que se obtenga lo planeado. Se detectan y corrigen los posibles errores de lógica. Una vez terminado el programa, puede requerir *mantenimiento* para realizar cambios o ajustes relativamente pequeños.

La compilación consiste en convertir el programa escrito en C a lenguaje máquina (C utiliza un enlazador de bibliotecas o *linker*). Al finalizar se crea un archivo ejecutable: es el que se utilizará para ejecutar ("correr") el programa.

1.4 El lenguaje C

"C es un lenguaje de programación de propósito general, asociado de modo universal al sistema operativo UNIX." [Joyanes].

C es un lenguaje de alto nivel, aunque también se utiliza para la programación de sistemas, ya que contiene instrucciones para el control a bajo nivel. Es una evolución de los lenguajes B y BCPL, los cuales carecían de la capacidad para manejar tipos de datos, una desventaja para el programador. Estos tres lenguajes fueron utilizados en los laboratorios Bell para crear el sistema operativo UNIX.

En 1978, con la publicación del libro *The C Programming Language*, escrito por Brian Kernighan y Dennis Ritchie, se inició formalmente el uso del lenguaje C. Ante el auge de este nuevo lenguaje, fue necesario escribir un estándar que cubriera las necesidades de compatibilidad y portabilidad. El estándar fue aprobado en **1989** por el comité técnico X3J11, del American National Standards Committe on Computers and Information Processing.

El lenguaje C ha evolucionado a C++ (creado por Bjarne Stroustrup en 1986) y a otros lenguajes que conservan características de C, como Java.

1.5 Elementos de un programa en C (conceptos básicos)

Un programa contiene varios elementos (aunque no es obligatorio que todos estén presentes siempre). A continuación se mencionan los más usuales.

1.5.1 Identificadores

En un programa siempre operan diversos elementos creados por el programador (variables, constantes, funciones, etcétera) o bien creados en bibliotecas de funciones (comúnmente llamadas librerías) junto con el lenguaje, como la función `printf`. Cada uno de estos elementos requiere un nombre exclusivo para diferenciarse de otros elementos usados en el mismo programa; a dichos nombres se les llama identificadores.

En C se siguen ciertas reglas para formar los identificadores:

- Un identificador se forma a partir de dígitos, letras y el carácter de subrayado (guión bajo); no se puede utilizar ningún otro carácter.
- El primer carácter de un identificador siempre debe ser una letra. Aunque también está permitido utilizar el guión bajo como primer carácter, no es muy común; más bien se emplea para formar identificadores con más de una palabra. No se puede utilizar un dígito como primer carácter en un identificador.
- El número de caracteres puede ser ilimitado; es decir, desde uno hasta los que el usuario quiera; sin embargo algunos compiladores de C reconocen únicamente los primeros 8 caracteres y en otros casos puede reconocer hasta 31. Un consejo es que los identificadores sean lo más compactos pero también sean claramente descriptivos.
- Es posible utilizar letras mayúsculas y minúsculas, pero se recomienda un empleo consistente, ya que el lenguaje C es sensible a mayúsculas y minúsculas, es decir, una 'a' es diferente de una 'A'.
- No se pueden utilizar palabras reservadas de C como identificadores; tampoco símbolos o espacios en blanco.

A continuación se muestran ejemplos de identificadores válidos:

`A`	`Nombre`	`Nombre_alumno`	`X1`
`a`	`_codigo`	`CODIGO`	`resultado_3`

Ahora se muestran ejemplos de identificadores no válidos y la razón.

Identificador	*Explicación*
`3id`	`El primer carácter debe ser una letra`
`Alumno#`	`El carácter # no es permitido`
`Codigo alumno`	`El espacio en blanco no es permitido`
`Codigo-alumno`	`El carácter - no es permitido`
`"alumno"`	`El carácter " no es permitido`

1.5.2 Tipos de datos

En un programa siempre se procesan datos, los cuales pueden ser de distinta naturaleza. Dependiendo de su tipo, se representará y almacenará el dato en la memoria de la computadora de una manera específica; es decir, el tipo de dato determina la cantidad de memoria requerida para almacenarlo. A continuación se listan los tipos de datos básicos de C, así como la cantidad de memoria que puede requerir cada uno. Cabe señalar que estas cantidades son las más usuales, ya que pueden variar de un compilador a otro; el rango de algunos de estos tipos de datos básicos pueden variar si se utilizan los modificadores de tipo como short, long, signed, unsigned.

Los tipos de datos básicos (o primitivos) son:

```
char
int
float
double
```

En la tabla 1.1 se muestran tipos de datos básicos con modificadores, el tamaño en bytes y el rango de valores que usualmente utilizan.

1.5.3 Variables

Para poder ejecutar un programa, es necesario que los datos estén almacenados, junto con las instrucciones, en la memoria. Muchas veces, dichos datos son proporcionados por el usuario del programa durante la ejecución del mismo o bien serán el resultado del procesamiento de otros datos. Es decir, no siempre es el programador quien define los datos, y sus valores no siempre se conocen de antemano. ¿Cómo entonces se puede almacenar datos que no se conocen desde el inicio? Con el uso de variables.

Una variable es un espacio en la memoria que el programador reserva con el fin de almacenar esos datos "desconocidos" cuando empieza la ejecución de un programa o que pueden ir cambiando durante ese proceso. Para poder reservar tantos espacios como se requieran es necesario declarar las variables.

Declaración de variables

La declaración de variables consiste en reservar los espacios de memoria que requiere el programa para su ejecución; para ello es necesario especificar el tipo de dato, así como el identificador con que se le hará referencia posteriormente.

TABLA 1.1 Tipos de datos y modificadores

Tipo	*Descripción*	*Cantidad de memoria requerida*	*Rango*
void	*Define vacío o valor NULL.*		
char	*Almacena un carácter. Puede almacenar un valor con signo.*	*1 byte*	*-128 a 127*
unsigned char	*Almacena un carácter o un valor sin signo.*	*1 byte*	*0 a 255*
int	*Define un valor numérico entero (sin fracción).*	*2 bytes*	*-32767 a 32768*
unsigned int	*Valor entero sin signo.*	*2 bytes*	*0 a 65,535*
short int	*Entero corto, puede ser igual al int o a la mitad.*	*2 bytes*	*-32767 a 32768*
float	*En punto flotante (puede ser una fracción o un entero con exponente).*	*4 bytes*	*3.4x10-38 a 3.4x10+38*
double	*En punto flotante del doble de tamaño del float (más cifras significativas para la fracción o mayor para el exponente).*	*8 bytes*	*1.7x10-308 a 1.7x10+308*
long	*Define un entero con signo, usualmente del doble de tamaño al int.*	*4 bytes*	*-2,147,483,648 a 2,147,483,647*
unsigned long	*Entero sin signo.*	*4 bytes*	*0 a 4,294,967,295*
long double	*Incrementa el tamaño del doublé.*	*10 bytes*	*3.4x10-4932 a 1.1x10+4932*

La sintaxis para la declaración de variables en C.

```
<tipo_dato>      <identificador>;
```

Ejemplos de declaración de variables:

```
int a;          Se reserva un espacio en la memoria llamado "a", con capacidad
                para un entero.
```

```
float b,c,d;    Se reservan 3 espacios en la memoria para guardar 3 números
                reales, a los cuales se hace referencia mediante "b","c" y "d"
                respectivamente.
char j;         Se reserva un espacio en la memoria para poder almacenar
                cualquier carácter y se puede hacer referencia a este espacio
                mediante el identificador "j".
```

Cómo reservar memoria

Recuerde que la memoria es un conjunto de celdas direccionables; es decir, se puede tener acceso a cada una mediante su dirección y en ellas puede almacenar datos.

Cuando se declara una variable en un compilador como C, el programador no necesita conocer la dirección absoluta de la celda o las celdas en las que va a almacenar los datos de su programa; simplemente declara sus variables y el compilador se encarga del resto. La siguiente figura muestra de manera descriptiva (no exacta) cómo quedaría la memoria luego de haber declarado gráficamente las variables de los ejemplos anteriores.

0	1	2	3 a	4 a	5	6	7	8	9 j
10 b	11 b	12 b	13 b	14	15 c	16 c	17 c	18 c	19
20	21	22 d	23 d	24 d	25 d	26	27	28	...

Suponiendo que cada celda midiera un byte, los espacios sombreados representan la reservación de memoria que se ha hecho de acuerdo con la declaración de variables de los ejemplos. El identificador `a` ocupa las celdas 3 y 4 (2 bytes por ser entero). De igual forma se reservaron las celdas 10, 11, 12 y 13 (4 bytes por ser un valor real) para el identificador `b`, así como las celdas de la 15 a la 18 para `c` y de la 22 a la 25 para `d`; finalmente se reservó la celda 9 (1 byte por ser carácter) para `j`.

Observe que se asignaron los espacios arbitrariamente, dado que en la realidad las variables no necesariamente quedarán de manera consecutiva en la memoria; lo que importa es que exista espacio suficiente para las variables a declarar.

Resumiendo: una variable es un espacio en la memoria cuyo contenido puede cambiar durante la ejecución de un programa; es decir, cuando declaramos una

variable, se asigna un espacio en la memoria para almacenar en él algún valor que sabemos puede cambiar durante el procesamiento de datos que el programa llevará a cabo cuando se ejecute.

1.5.4 Constantes

Las constantes son elementos frecuentemente utilizados en los programas; si ya quedó claro el concepto de variable, será más sencillo explicar el de constante. Una constante es un espacio en memoria que recibe un valor por primera vez y generalmente no se modifica durante la ejecución de un programa.

Una constante se utiliza cuando se conoce de antemano el valor de algún dato pero además se sabe que este dato no debe cambiar. Por ejemplo se puede pensar que en un programa (no importa para qué sirva éste) se requiere trabajar con el valor de π; sabemos que π generalmente maneja el valor de 3.1416 y que no puede tener otro. Sería ilógico pensar en reservar un espacio para una variable si dicho valor no tiene por qué cambiar en el transcurso del programa; de esta forma, lo más conveniente sería definir una constante para almacenar 3.1416.

En C, las constantes se crean utilizando la directiva del preprocesador *#define* (constantes simbólicas definidas) o bien la palabra reservada const (constantes simbólicas declaradas).

La constante simbólica *#define*

En el lenguaje C, *#define* se utiliza para declarar constantes simbólicas y crear macros. En este libro sólo se utilizará para definir constantes, mediante la siguiente sintaxis:

```
#define <identificador> <valor>
```

Ejemplos de constantes simbólicas:

```
#define SALUDO "Hola a todos"
#define PI 3.1416
#define NP 1506
#define CAR 'a'
```

Los nombres simbólicos son los identificadores; el preprocesador de C sustituye los valores "Hola a todos", 3.1416, 1506 y 'a' cada vez que se encuentra en un programa el nombre de las constantes simbólicas SALUDO, PI, NP y CAR, respectivamente.

Observe que a las constantes creadas con *#define*, no es necesario especificarles el tipo de dato que deberán tener; estas constantes adquieren su tipo dependiendo de la naturaleza del valor asignado. Así, las constantes definidas anteriormente son:

```
NP        constante entera, ya que representa la cantidad de 1506.
PI        constante real, representa la cantidad de 3.1416.
CAR       constante de carácter, ya que representa a la letra 'a'.
SALUDO    constante de cadena de caracteres, representa la secuencia
          "Hola a todos".
```

Cuando se utiliza **#define** para crear constantes, en realidad los valores no ocupan un espacio en la memoria como en el caso de las variables, sino que el compilador sustituye cada ocurrencia del nombre simbólico por su respectivo valor, antes de analizar sintácticamente el programa fuente.

Dicho de otro modo, cada vez que alguna instrucción utilice el identificador NP, éste será sustituido por el 1506 con que fue definido.

La constante numérica *const*

Por otro lado, una constante también se puede crear utilizando la palabra reservada *const*; si se procede de esta manera, el dato ocupa un espacio en memoria como si fuera una variable, excepto porque su contenido no cambia.

La sintaxis para declarar una constante *const* es:

```
<const> <tipo de dato> <identificador> = <valor>;
```

Ejemplos:

```
const int NP=1506;
const float PI=3.1416;
const char CAR='a';
const char SALUDO[ ]= "Hola a todos";
```

Las declaraciones anteriores podrían utilizarse en lugar del **#define**, la diferencia consiste en que *const* tiene un espacio de memoria reservado para cada dato; además de su sintaxis, observe que **const** sí especifica el tipo de dato de cada valor; se utiliza el operador = para asignar dicho valor y además cada sentencia (o instrucción) termina con ;.

1.5.5 Palabras reservadas

Las palabras reservadas de C son aquellas cuyo significado se encuentra definido en el lenguaje: ya tienen un uso específico. Éstas se escriben en las instrucciones de los programas.

Las palabras reservadas de C son:

auto	*double*	*int*	*struct*
break	*else*	*long*	*switch*
case	*enum*	*register*	*typedef*
char	*extern*	*return*	*unión*
const	*float*	*short*	*unsigned*
continue	*for*	*signed*	*void*
default	*goto*	*sizeof*	*volatile*
do	*if*	*static*	*while*

Observe que la lista de palabras reservadas está escrita en letras minúsculas; y justamente así se deben utilizar para que cumplan su propósito dentro de un programa. Aunque es posible utilizar cualquiera de ellas escrita en mayúsculas como identificador, esto no se recomienda en la práctica.

1.5.6 Comentarios

Los comentarios son cadenas de caracteres o texto que describen partes del programa que el programador desea explicar; dicho texto no es parte del programa fuente, sino una descripción del mismo. Los comentarios generalmente están dirigidos a otros programadores, no a los usuarios.

Para poder usar comentarios en un programa en C y que el compilador no los considere como instrucciones del programa fuente, se utilizan los símbolos /* y */ para encerrar el texto: /*. . . texto. . . */.

1.6 Entrada y salida de datos

Un programa es un conjunto de instrucciones que la computadora ejecuta con el fin de obtener un resultado o bien la solución a un problema determinado. Casi siempre, este resultado se obtiene a partir del procesamiento de los datos. Generalmente, los datos dependen del usuario y éste tendrá que conocer los

resultados del programa que está utilizando. Para ello se requiere contar con instrucciones que permitan a los usuarios introducir datos y otras que permitan mostrar los resultados generados por los programas.

El lenguaje C cuenta con las funciones ***scanf( )*** y ***printf( )*** para entrada y salida de datos respectivamente, las cuales se pueden utilizar agregando el archivo de cabecera ***#include <stdio.h>***.

1.6.1 Salida de datos

La computadora dispone de diversos medios para proporcionar la salida de datos, como la impresora, archivos o el más utilizado: el monitor. Precisamente la función *printf( )* se utiliza para mostrar datos a través de este dispositivo. Su sintaxis es la siguiente:

```
printf("texto, cadena de control de tipo", argumentos);
```

donde texto y cadena de control de tipo son opcionales, dependiendo de lo que se desee mostrar. Cadena de control es una cadena de caracteres "%tipo" que indica el tipo de dato a desplegar (lo requiere la función *printf( )*). Por otro lado, argumento o argumentos es el valor o los valores que se pretende mostrar, y pueden ser variables, constantes, expresiones aritméticas, resultados de funciones o simplemente texto que el programa debe mostrar al usuario. A continuación se muestra un ejemplo del uso y la explicación.

```
int a=7;
float b=8.2;
char c='s';

printf("%d", a);              Se visualiza un 7, que es el contenido de la
                              variable a.
printf("%d", a+b);            Se visualiza un 15, ya que es la suma de a + b
                              mostrada como valor entero.
printf("%f", a+b);            Se visualiza un 15.2, ya que es la suma de a +
                              b mostrada como valor real.
printf("%c", c);              Se visualiza la letra 's' que es el contenido
                              de la variable c.
printf("%c %d %f", c, a,b);   Se visualiza s 7 8.2 que son los valores de
                              las variables c, a y b respectivamente.
printf("HOLA");               Se visualiza la palabra HOLA.
```

TABLA 1.2 Cadenas de control de tipo para salida

Cadena de tipo	*Descripción*
`%d`	*El dato es un entero decimal (int).*
`%i`	*El dato es un entero.*
`%o`	*El dato es un entero octal.*
`%x`	*El dato es un entero hexadecimal.*
`%u`	*El dato es un entero sin signo en decimal (unsigned int).*
`%c`	*El dato es un carácter (char).*
`%e`	*El dato es un real expresado en base y exponente (float).*
`%f`	*El dato es un real escrito con punto decimal con signo (float).*
`%g`	*El dato es un real (float).*
`%s`	*El dato es una cadena de caracteres que finaliza con el carácter nulo \0.*
`%lf`	*El dato es real de tipo long double.*

Observe que cuando se imprime texto no es necesario utilizar alguna cadena de tipo, sólo el texto tal como se desea mostrar, encerrado entre comillas (“ ”). Para mostrar variables es necesario usar la cadena de tipo adecuada, también dentro de las comillas. En la tabla 1.2 se presentan las diferentes cadenas de tipo que se utilizan en C dependiendo de los tipos de datos que se desea imprimir en pantalla.

Como ya se ha mencionado, `printf( )` puede manejar más de un argumento, para ello es necesario usar las cadenas de tipo correspondientes por cada argumento que se requiera visualizar, como en el ejemplo siguiente:

```
printf("%d%f%c", a,b,c);
```

muestra en pantalla 78.2s, que son los valores que tienen almacenados las variables que se usaron con la función, pero la forma en que se presentan no es muy conveniente, pues sería más práctico que hubiera un espacio entre cada valor y no mostrarlos como si fuera uno solo. ¿Cómo se puede solucionar este problema? Utilizando ***secuencias de escape***, de tal modo que si modificamos la llamada anterior así:

```
printf("%d \t %f \t %c", a,b,c);
```

al ejecutarse la instrucción, mostraría lo siguiente:

```
7  8.2      s
```

ya que la secuencia de escape \t que se añadió en la llamada a la función, inserta una tabulación en cada lugar en que es colocada.

Las secuencias de escape son también cadenas de caracteres que tienen un significado especial dependiendo de la cadena que se utilice. La tabla 1.3 que se presenta a continuación muestra las secuencias de escape que utiliza el lenguaje C, así como su acción.

Analicemos el siguiente ejemplo y su salida.

```
printf("%d \n \t %f \n \t\t %c", a,b,c);
```

```
7

    8.2

          s
```

TABLA 1.3 Secuencias de escape

Secuencia de escape	*Descripción*
\a	*Alarma*
\b	*Retroceso*
\f	*Avance de página*
\n	*Retorno de carro y avance de línea*
\r	*Retorno de carro*
\t	*Tabulación*
\v	*Tabulación vertical*
\\	*Diagonal invertida*
\?	*Signo de interrogación*
\"	*Comillas dobles*
\000	*Octal*
\xhh	*Hexadecimal*
\0	*Carácter nulo*

La salida se mostraría de esta manera ya que después de imprimir el valor de cada variable se imprime un "*enter*" seguido de un tabulador.

También es posible que en algunos casos se tenga que mostrar algún valor dentro de un mensaje, como en:

```
printf("El valor de la variable a es %d", a);
```

y en pantalla se vería del siguiente modo:

```
El valor de la variable a es 7
```

pero si modificamos la función así:

```
printf("%d Es el valor de la variable a ", a);
```

el resultado en pantalla sería:

```
7 Es el valor de la variable a
```

Observe que precisamente en el lugar que ocupa la cadena de tipo dentro del texto, es donde aparecerá el valor de la variable o el elemento que se va a mostrar.

1.6.2 Entrada de datos

La entrada de datos, u operación de escritura, se puede hacer a través de diferentes dispositivos como un teclado o un archivo, por ejemplo. Sin embargo, si se usa la función **`scanf( )`** se trata de una entrada de datos desde el teclado. La sintaxis de **`scanf( )`** es la siguiente:

```
scanf("cadena de control de tipo", &variable);
```

Igual que la función **`printf( )`**, la función **`scanf( )`** requiere la cadena de tipo por cada variable o variables que se desea leer, el símbolo `&` es un apuntador que "apunta" a la dirección asignada a la variable que viene a continuación y ésta será el área de memoria donde se almacenará ese dato de entrada. Es indispensable utilizar este símbolo, de lo contrario, el valor nunca será guardado en la variable.

EJEMPLO 1.1 Programa que lee dos datos y los muestra

```
int edad;
float est;
printf("Teclea tu edad");
scanf("%d",&edad);
```

(*continúa*)

(*continuación*)

```
printf("Teclea tu estatura");
scanf("%f",&est);
printf("Tienes %d años y mides %f mts.",edad,est);
```

En este ejemplo se declaran las variables `edad` y `est`; primero se utilizó la función `printf()` para mostrar en pantalla un mensaje que solicita al usuario su edad; el usuario debe escribir su edad, por ejemplo, 20, y este valor se almacenará en la variable edad que se está usando como argumento en la función `scanf( )`; posteriormente aparece otro mensaje solicitando su estatura, supongamos que el usuario teclea 1.75, que se almacena en `est`; finalmente aparece el mensaje:

```
Tienes 20 años y mides 1.75 mts.
```

Generalmente los programas interactúan con el usuario en la lectura de datos.

Resumen

La computadora es un dispositivo electrónico que procesa instrucciones y datos. En ella se pueden utilizar varios tipos de lenguajes como los de alto nivel, que son los compiladores; el lenguaje ensamblador, que es de bajo nivel, y el lenguaje máquina, en ceros y unos, que es el único que la computadora entiende.

El lenguaje que esta obra aborda es el C, un lenguaje de programación de propósito general.

Algunos de los elementos utilizados en un programa son identificadores, variables, constantes y comentarios.

La instrucción usada comúnmente para leer datos por la entrada estándar (teclado) es `scanf`; para mostrar en la salida estándar (monitor) se utiliza la instrucción `printf`.

Evaluación

I. Describa los siguientes conceptos.

1. ¿Qué es un programa?
2. ¿Qué es la CPU?
3. ¿A qué se le llama software?
4. ¿Qué es la memoria principal?
5. ¿Qué es lenguaje de programación?

6. ¿Qué es lenguaje C?
7. ¿Qué lenguajes pueden usarse en una computadora?
8. ¿Cuáles son las formas para declarar una constante?
9. ¿Cómo se declara una variable?

II. Responda las siguientes preguntas.

1. ¿Por qué no se debe utilizar una variable antes de asignarle un valor?
2. ¿Por qué no se debe asignar un valor real a una variable declarada como entero?
3. Los errores de _____ son detectados por el compilador y los errores _____ sólo pueden ser detectados por el programador.
4. ¿Cuál es la estructura general de un programa en C?
5. ¿Cuál es la razón de que los siguientes identificadores sean inválidos?
   a) .uno
   b) 1uno
   c) U no

III. Escriba lo que se pide a continuación.

1. Tres identificadores válidos.
2. Tres identificadores no válidos.
3. Una variable de tipo `char`, otra tipo `int`, otra tipo `float` y otra tipo `double`. Asígnele un valor 4 a cada una de las variables con la instrucción `scanf` y muestre el contenido de cada variable usando la instrucción `printf`.
4. Tres constantes usando `const: char, int` y `float`.
5. Tres constantes usando `#define`.
6. Cinco palabras reservadas.
7. Un comentario que contenga el nombre completo de usted y la fecha de hoy.

Ejercicios propuestos

I. Describa qué imprimen los siguientes fragmentos de código.

1.
```
char A;
A='a';
printf(" %c ",A); _____
printf(" %d ",A); _____
A='a'+ 10;
printf(" %c ",A); _____
printf(" %d ",A); _____
```

2.
```
int B;
B=5;
printf(" %d ",B); _____
printf(" %c ",B); _____
B=5+'A';
printf(" %d ",B); _____
printf(" %c ",B); _____
```

3.
```
#define num 15
 printf(" %d ",num); _____
 printf(" %c ",num); _____
```

4.
```
#define num 15
 num=5+'A';
 printf(" %d ",num); _____
 printf(" %c ",num); _____
```

5.
```
#define p printf
 int a=1,b=2;
 float x=3,y=4;
 long z=5;
 short int f=6;
 unsigned int i=9;
 unsigned long g=7;
 double h=8;
 p(" entero %d \n",a+b); _____
 p(" real %f \n",(float)(a+b); _____
 p(" largo %lf \n",z); _____
 p("short %d\n",f); _____
 p("unsigned int %d \n",i); _____
 p(" unsigned long %d\n",g); _____
 p(" doble %lf \n",h); _____
```

II. ¿Qué dato se almacenaría en las siguientes variables?

1.
```
int a;
/*leer a=3 */
scanf("%d",&a);  _____
scanf("%f",&a);  _____
scanf("%c",&a);  _____
```

2.
```
char x;
/*leer x=3 */
scanf("%d",&x);  _____
scanf("%f",&x);  _____
scanf("%c",&x);  _____
```

3.
```
float w;
/*leer w=2.5 */
scanf("%d",&w);  _____
scanf("%f",&w);  _____
scanf("%c",&w);  _____
```

Capítulo 2

Aritmética de C

En este capítulo se revisa el uso de operadores, se mencionan los más comunes, su clasificación y algunos ejemplos.

Un operador es un signo (símbolo) que indica al compilador el tipo de operación que se efectuará con los datos. El lenguaje C cuenta con diferentes categorías de operadores, los más elementales son ***aritméticos***, ***relacionales*** y ***lógicos***, y podemos encontrarlos prácticamente en cualquier otro lenguaje, aunque posiblemente se representen de otra manera. Además de estos operadores, existen otros que son propios de C, y se explicarán más adelante.

2.1 Inicialización y asignación de variables

Inicializar una variable ya declarada consiste en asignarle un valor antes de que se utilice en un programa; una vez inicializadas, las variables

pueden modificar su contenido conforme se requiera en el programa mediante nuevas asignaciones. Sin embargo, cabe aclarar que no siempre será necesario darles un valor inicial (ejemplificaremos esto más adelante).

El lenguaje C permite inicializar el valor de la variable, ya sea en el momento de la declaración o posteriormente; el formato es el siguiente:

```
<tipo dato> <identificador> = <valor>;
```

donde valor puede ser una constante, una variable, una expresión aritmética o la llamada a una función; más adelante se detalla sobre los temas de expresiones y funciones. Por ejemplo:

```
int a = 3;
a = 5;
float b=4.56,c=7.2,d;
d = b+c;
```

En el primer caso se declara la variable entera `a` y su valor inicial es 3; en la segunda asignación se almacena un 5 en la variable `a`, lo cual hace que el 3 anterior se pierda. La variable conserva siempre el último valor asignado.

En este ejemplo se declaran `b`, `c` y `d`, tres variables de tipo real. A las dos primeras se les asigna 4.56 y 7.2 respectivamente; `d` no se inicializa, ya que se le asigna el resultado de la expresión `a + b`, dando como resultado 11.76. Un ejemplo más: se declara la variable `t` cuyo valor inicial es la letra `r`.

```
char t = ' r ';
```

En los ejemplos anteriores se ha utilizado el operador = (operador de asignación) que, en lenguaje C, no significa igualdad, como en otros contextos más comunes.

El operador asignación = se utiliza para almacenar un valor en una variable. Si la variable ya está declarada la sintaxis de asignación es la siguiente:

```
<variable > = <valor>;
```

La asignación siempre será de derecha a izquierda, por lo tanto, del lado izquierdo del operador asignación siempre se deberá utilizar una variable; en cuanto al valor, como ya se mencionó, puede ser otra variable, una constante, una expresión aritmética o el resultado de una función. Se muestran los siguientes ejemplos, asumiendo que las variables ya están declaradas:

```
m=5;
m=m+3;
```

```
n=m;
z=m+5;
w=sqrt(9);
```

En el primer ejemplo a `m` se le asigna 5; en la segunda instrucción a la variable `m` se le asigna lo que tiene `m` más 3 (quedando con 8); en el tercer ejemplo a `n` se le asigna el valor de `m`, es decir 8; en el siguiente ejemplo a la variable `z` se le asigna el resultado de la suma de `m+5` (13); y por último, a la variable `w` se le asigna la raíz cuadrada de 9, la cual se calcula utilizando la función `sqrt( )` de la biblioteca `math.h`.

El lenguaje C también permite asignar valores de la siguiente forma:

```
int m=n=z=0;
```

Utilizada de esta forma es llamada asignación múltiple, y significa que se les asigna el mismo valor (cero) a todas las variables; la asignación siempre es de derecha a izquierda.

2.2 Operadores aritméticos

Los operadores aritméticos son los que utilizamos normalmente para realizar las operaciones básicas aritméticas: suma, resta, multiplicación, división y residuo o módulo. Frecuentemente escucharemos que a estos operadores se les conoce como binarios, esto significa que siempre se utilizan dos operandos (datos) para que funcionen. A continuación la tabla 2.1 los muestra.

TABLA 2.1 Operadores aritméticos con ejemplos

Operador	*Operación*	*Tipo de datos de los operandos*	*Ejemplo*	*Resultado*

+	*Suma*	*Enteros y reales (pueden ser diferentes).*	`3 + 2`
			`3.3 + 5`
			`8.2 + 7.1`
				`5`
				`8.3`
				`15.3`

-	*Resta*	*Enteros y reales (pueden ser diferentes).*	`3 - 2`
			`3.3 - 5`
			`8.2 - 7.1`
				`1`
				`-1.7`
				`1.1`

*	*Multiplicación*	*Enteros y reales (pueden ser diferentes).*	`3 * 2`
			`3.3 * 5`
			`8.2 * 7.1`
				`6`
				`16.5`
				`58.22`

(continúa)

(*continuación*)

/	División	Enteros y reales (pueden ser diferentes).	3 / 2 3.3 / 5 8.2 / 7.1	1 0.66 1.1549296
%	Residuo o módulo	Sólo acepta operandos enteros.	3 % 2 3.3 % 5 8.2 % 7.1	1 inválido inválido

Todos los operadores aceptan la combinación de tipos de datos, excepto módulo %, que sólo se utiliza con operandos enteros o carácter.

El tipo de dato del resultado de la operación depende de los tipos de datos de los operandos que se utilicen; es decir, si se usan sólo enteros, el resultado será un entero, pero si se combinan enteros y reales, el resultado será un valor real.

Por otro lado, observe también que en los casos de las operaciones 3 / 2 y 3 % 2 el resultado es 1.

Esto es porque tanto el cociente como el residuo en esta operación son precisamente 1. Veamos esto en el ejemplo siguiente:

```
  1 ← (cociente)
2|3
  1 ← (residuo)
```

Y como el operador residuo maneja datos sólo de tipo entero, el resultado también será de tipo entero. Así que si el divisor es menor que el dividendo, no se debe esperar un resultado con punto decimal, aunque sí un residuo; por ejemplo, el resultado de la siguiente expresión:

```
1 % 4
```

será 1, ya que

```
  0
4|14
  1 ← (residuo)
```

2.2.1 Prioridad de los operadores aritméticos

Cuando en una expresión existen operadores diferentes, ¿cuál se evalúa primero? Estos operadores siguen las reglas matemáticas en cuanto a precedencia o jerarquía de operadores.

La tabla 2.2 muestra esa precedencia:

TABLA 2.2 Precedencia de operadores aritméticos

( )
* , / , %
+, -

En primer lugar aparece el operador ( ), debido a que en las matemáticas tradicionales se utiliza este símbolo para determinar la prioridad de operadores en expresiones grandes. En segundo lugar se evalúan multiplicación, división y módulo, los tres con la misma prioridad; posteriormente se evalúan la suma y la resta.

EJEMPLO 2.1 Expresiones válidas y comentario a la solución

Expresión	*Reglas*
a) 3 + 4 - 5 *7 - 5* *2*	*Cuando en una expresión se encuentran operadores con la misma prioridad, como en este caso la suma y la resta, las operaciones se realizan de izquierda a derecha.*
*b) 8+5*9-6* *8 + 45 - 6* *53 - 6* *47*	*Aunque la suma está primero, el operador de mayor jerarquía es la multiplicación; posteriormente se lleva a cabo la suma, es decir, nuevamente se evalúan de izquierda a derecha.*
6/2+4(5-2)* *6/2+4*(3)* *6/2+12* *3+12* *15*	*En este ejemplo se efectúa primero la división; para efectuar la suma, es necesario llevar a cabo la operación entre paréntesis (resta) y luego la multiplicación. Con esos resultados es posible efectuar la suma.*
d) 6+7((3-1)%2)* *6+7*(2%2)* *6 + 7 * 0* *6 + 0* *6*	*Cuando en una expresión existan paréntesis anidados, se evaluarán de adentro hacia afuera, luego se evalúan los demás operadores de acuerdo a su precedencia.*

Nota: Para definir prioridad de operadores, C utiliza únicamente los símbolos ( ). Las { } (llaves) y los [ ] (corchetes) en el lenguaje C no se utilizan para asociar. Su significado se explicará más adelante.

Cuando se combinan variables, constantes y otros elementos mediante los operadores aritméticos, se forman expresiones aritméticas; el resultado de una expresión aritmética puede ser cualquier valor numérico, todo depende de los operandos y la asociatividad de los operadores.

2.2.2 Otros operadores de asignación

Existen otros operadores de asignación muy particulares del lenguaje C.

```
+=
-=
*=
/=
%=
```

Los cuales también sirven para sumar, restar, multiplicar y dividir, respectivamente, reduciendo las instrucciones, como se muestra la tabla 2.3. Suponiendo que declaramos

```
int a=10;
```

TABLA 2.3 Operadores de asignación

Expresión	*Equivale a*	*Resultado*
`a+=2;`	`a=a+2;`	`12`
`a-=2;`	`a=a-2;`	`8`
`A*=2;`	`a=a*2;`	`20`
`a/=2;`	`a=a/2;`	`5`
`A%=2;`	`a=a%2;`	`0`

2.3 Operadores de incremento y decremento

Anteriormente se mencionó que a los operadores aritméticos básicos se les suele clasificar como binarios porque cada uno ellos requiere siempre de dos operandos. Sin embargo, el lenguaje C cuenta con una serie de operadores propios a los que se les llama unarios. Eso significa que únicamente requieren de un operando cuando se utilizan, lo cual proporciona ciertas ventajas.

El **operador de incremento** `++` se utiliza para modificar el valor de la variable sumándole 1 al valor que tiene.

```
a=9;
a++;
```

suponiendo que la variable `a` se ha declarado previamente, en la primera instrucción se le asigna el valor de 9 y en la siguiente se incrementa su valor en 1, lo cual hace que la variable `a` ahora tenga almacenado un 10.

Este operador reduce el código, ya que la expresión `a++` equivale a la expresión `a=a+1`, que dicho en palabras significa a `la variable a` se le asigna lo que tenga más 1.

Por otro lado también se cuenta con el operador decremento `--`. Este operador es lo contrario al anterior, es decir, disminuye en 1 el valor que tenga la variable, como en el siguiente caso:

```
a=9;
a--;
```

de igual manera se le asigna un 9 a la variable `a` y posteriormente se reduce su valor quedando con valor de 8.

Los operadores `++` y `--` se pueden usar como prefijos o sufijos: se pueden usar antes o después de la variable y aunque en ambos casos ésta se incrementa o reduce en 1, existe cierta diferencia en la forma de utilizarlos.

Veamos los siguientes ejemplos:

```
1) a=10;
   X=++a;
```

```
2) a=10
   x=a++;
```

En el ejemplo 1 la variable `x` recibe un 11, porque `a` primero se incrementa y luego se asigna. En el ejemplo 2 `x` recibe un 10, ya que primero se asigna el valor de `a` y posteriormente se incrementa.

2.4 Operadores relacionales

Estos operadores se utilizan para expresar condiciones en los programas y así determinar el orden en que se ejecutarán las instrucciones; una condición en C es una expresión booleana cuyo resultado puede ser únicamente verdadero o falso. La tabla 2.4 muestra los operadores relacionales.

TABLA 2.4 Operadores relacionales

Operador	*Descripción*
`<`	*menor que*
`<=`	*menor o igual que*
`>`	*mayor que*
`>=`	*mayor o igual que*
`==`	*igual que*
`!=`	*no igual que (diferente de)*

Estos operadores actúan con dos operandos que pueden ser variables, constantes, expresiones aritméticas o funciones, y el resultado obtenido es un valor entero, ya que verdadero se representa con un 1 y falso con el valor 0. Cuando una expresión puede dar como resultado sólo verdadero o falso, se le llama ***expresión lógica*** o ***booleana***. A continuación se muestra un ejemplo:

si a=10 y b=5

Operación	*Descripción*	*Resultado*
`a < b`	*a menor que b*	*Falso (0)*
`a <= b`	*a menor o igual que b*	*Falso (0)*
`a > b`	*a mayor que b*	*Verdadero (1)*
`a >= b`	*a mayor o igual que b*	*Verdadero (1)*
`a == b`	*a igual que b*	*Falso (0)*
`a != b`	*a diferente de b*	*Verdadero (1)*

2.5 Operadores lógicos

Los operadores lógicos `&&` y `||` actúan con dos operandos; el operador `!`, con un operando. Se requiere que sean expresiones lógicas, generalmente formadas con los operadores relacionales.

Los operadores lógicos sirven para unir más de una condición en un programa y poder así formar condiciones más complejas, cuyo resultado también puede ser únicamente cierto o falso, la tabla 2.5 muestra los operadores lógicos.

TABLA 2.5 Operadores lógicos

Operador	*Descripción*
`&&`	*Y (and) El resultado de una operación y lógica será verdadero si ambos operandos son verdaderos, de lo contrario, será falsa.*
`||`	*O (or) El resultado de una operación o lógica será verdadero si alguno de los operandos o los dos son verdaderos; si todos son falsos, será falsa.*
`!`	*No (not) El resultado de una operación no lógica sólo será verdadero si el operando es falso, de lo contrario, será verdadero.*

Ejemplo del resultado del uso de operadores lógicos.

si a=10, b=5, c=10 y d=3

Operación	*Descripción*	*Resultado*
`(a<b)&&(a==c)`	`a no es menor que b` `a si es igual que c`	`Falso (0)` `la primera no se cumple`
`(a>b)&&(a>=d)`	`a si es mayor que b` `a si es mayor o igual que d`	`Verdadero (1)` `las dos se cumplen`
`(a==c) || (c != d)`	`a si es igual que c` `c si es diferente de d`	`Verdadero (1)` `las dos se cumplen`
`(d>c) || (b>a)`	`d no es mayor c` `b no es mayor que a`	`Falso (0)` `ninguna se cumple`
`(c>=d) || (c>a)`	`c si es mayor o igual que d` `c no es mayor que a`	`Verdadero ( 1 )` `la primera sí se cumple`

2.6 Operador condicional

Este operador se utiliza para expresar condiciones en un programa y puede sustituir a la estructura de control `if-else` que se explica más adelante. El operador requiere una expresión lógica, la cual se evalúa y dependiendo si es verdadera o falsa se ejecutan instrucciones distintas, el formato es el siguiente:

<expresión 1> ? <expresión 2> : <expresión 3>

donde

<expresión 1> es la condición que se evalúa (debe ser una expresión booleana), si es verdadera, se ejecutará lo indicado en *<expresión 2>*; pero si *<expresión 1>* es falsa, entonces se ejecutará *<expresión 3>*.

Si `a` es una variable de tipo entero. ¿Cuál será el resultado de la siguiente expresión condicional?

```
(a>10) ? 1 : 0;
```

dependiendo del valor que se le haya asignado a la variable `a`, será el valor de toda la expresión condicional. Por ejemplo si `a` tiene un valor de 11 o mayor, el resultado será 1; si la variable tiene almacenado un 10 o menos, el resultado será un 0. Se muestra el siguiente ejemplo:

```
a=3; b=4;
c = (a>b) ? a+b : a*b;
```

Aquí el resultado de la expresión condicional se asigna a la variable `c`. Primero se evalúa si `a` es mayor que `b`; como en este caso el resultado es falso, se ejecuta la expresión que está después de : (dos puntos), es decir, la multiplicación de `a` por `b`, cuyo resultado es 12, el cual es almacenado en la variable `c`.

Si cambiáramos los valores de `a` y `b` por 8 y 6 respectivamente, el resultado de la asignación a `c` sería 14. Se evalúa si `a` es mayor que `b`, y como es verdadero, se ejecuta la expresión que está después del signo ?; la suma de `a` y `b`.

2.6.1 Prioridad de operadores

A continuación se muestra la tabla 2.6 con la lista de operadores que se han explicado en orden de prioridad.

TABLA 2.6 Operadores y su prioridad

Operador	
`( )`	`==, !=`
`!`	`&&`
`++, --`	`||`
`*, /, %`	`?:`
`+, -`	`+=, -=, *=, /=, =`
`<, <=, >, >=`	

Cabe aclarar que éstos no son los únicos operadores con que cuenta C. Este lenguaje opera con muchos otros que se utilizan para distintos propósitos; como el presente es un texto introductorio, se estudiarán únicamente los mencionados.

EJEMPLO 2.2 Calcule el resultado de la siguiente expresión

```
X= 3*a+(--a)-(-a-2)*2/3
```

Para a=4 se resolverá así:

```
X=12+(--a)-(-a-2)*2/3
X=12+(3)-(-a-2)*2/3
X=12+3-(-3-2)*2/3
X=12+3-(-5)*2/3
X=12+3+5*2/3
X=12+3+10/3
X=12+3+3.3
X=15+3.3
X=18.3
```

Resumen

En este capítulo se analizaron los operadores que existen en el lenguaje C. Un operador es un símbolo que indica al compilador el tipo de operación que se llevará a cabo sobre los datos; los más usuales son los aritméticos, relacionales y lógicos. Al relacionarlos con los datos se forman expresiones.

Existen dos tipos de expresiones: aritméticas y lógicas. Una expresión aritmética es aquella cuyo resultado es un valor numérico. Una expresión lógica podrá tener como resultado sólo cierto o falso.

Al utilizar los operadores es necesario tomar en cuenta las prioridades que matemáticamente tienen éstos.

El operador de asignación se utiliza para asignar valor a una variable. El lenguaje C ofrece algunos otros operadores de asignación propios del lenguaje que llevan implícitas operaciones aritméticas como +=, -= entre otros.

Evaluación

I. Conteste las siguientes preguntas.

1. Un _____ es un símbolo que le indica al compilador cómo se han de manejar los datos.

2. La _____ consiste en asignar un valor a una variable antes de utilizarla en un programa.

3. ¿Cuál de las siguientes es una operación de asignación?

a) `5=x` b) `x==5` c) `x+=5` d) `x-5`

4. ¿Qué significa que un operador sea binario o unario?

5. El operador _____ se puede usar sólo sobre datos enteros.

6. ¿Cuál operador se utiliza para definir prioridades cuando en una expresión coinciden operadores del mismo nivel jerárquico?

a) `( )` b) `{ }` c) `[ ]` d) `&&`

7. El operador incremento se escribe _____ y se utiliza cuando se quiere sumar uno a las variables.

8. Los operadores _____ se utilizan para unir condiciones en los programas.

9. Cuando se combinan operadores relacionales y lógicos en una expresión, ésta puede tener como resultado sólo cierto o falso, lo cual significa que es una expresión de tipo _____.

10. Operador que evalúa una condición y dependiendo si es cierta o falsa ejecuta una determinada instrucción u otra _____.

II. Escriba el resultado del siguiente ejercicio.

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

main()
{
 int j=4, k=3,i=2;
```

```
float x=1.5, z=2.4, t;
t=((float)(j%k)/2);
t++;
x*=++z;
t-= (x+=++i);
printf("\n el valor de t es %f\n",t);
getch();
return 0;
}
```


Ejercicios propuestos

I. ¿Qué resultado se imprime en los siguientes programas?

1.

```
#include <stdio.h>

main()
{
 int a,b,c=3,d=2;
 a= 8-5 * 3 + 2;
 b= 7%3 + 4 * 2;
 printf(" valor de a %d\tvalor de b %d\n",a,b);
 b%=a;
 printf(" valor de b %d\t\n",b);
 b=(-4)%3;
 printf(" valor de b %d\t\n",b);
}
```

2.

```
#include <stdio.h>

main()
{
 int c=3,d=2;
 c -= d++ * 3;
```

```
 printf(" valor de c %d\tvalor de d %d\n",c,d);
 system("pause");
}
```

3.

```
#include <stdio.h>

main()
{
 int c=3,1;
 c = (c * 2 - (1 = 4, --1));
 printf(" valor de c %d\t valor de 1 %d\n",c,1);
 getch();
 return 0;
}
```

4.

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

main()
{
 int a=7, b=3, c=2, resultado;
 resultado =((b%c)/2)+1;
 resultado -= a;
 resultado %= ++a;
 printf("\n el valor de resultado = %d\t a= %d ",resultado,a);
 getch();
 return 0;
}
```

II. Escriba el resultado de la siguiente expresión, considerando los valores.

Para $a = 2$, $b = 3$, $c = 4$, $d = 5$

1. `a+b*c/d`
2. `a % 5 % 2 + c`
3. `(a+b)*c % d`
4. `(d<65)&&(3==d)`
5. `a<=b || (3>a)`
6. `(c<=7+d) || (7>c)`
7. `(a>=d) && (2-14==c+1)`
8. `a % d % c`
9. `3+5*d % 10`
10. `a/b % 2`
11. `-a*7`
12. `(a>d) && (b<c) || (d>5)`
13. `((a>b) || (a<c)) && ((a=c)`
14. `||(a>=b))`
15. `((a>=b) || (a<d)) && 16.`
16. `((a>=d) && (c>d))`
17. `!(a<=c) || (c>d)`

CAPÍTULO 3

Programación estructurada

En este capítulo se describen las partes de un programa y la estructura de control básica secuencial como inicio para la creación de programas.

La programación estructurada es una técnica para crear programas siguiendo ciertas reglas que permiten modificarlo, leerlo y mantenerlo fácilmente. Entre las características principales de esta metodología se incluye el empleo de estructuras de control que realizan acciones sobre las estructuras de datos, las cuales se explican más adelante. El programa debe tener una sola entrada y una sola salida.

3.1 Estructura de un programa en C

El programa consta generalmente de varias funciones. Una función es un subprograma con una tarea específica. Estos subprogramas

(generalmente pequeños) se diseñan con el fin de utilizarse no sólo una vez ni en un solo programa, sino las veces que se requieran y en cualquier programa.

El lenguaje C proporciona fundamentalmente dos formas de manejo de funciones: las internas y las externas.

Las funciones internas son las ya implementadas e incorporadas en el lenguaje de programación. Para poder hacer uso de ellas, es necesario incluir el archivo de cabecera estándar correspondiente (también llamada biblioteca) al que corresponda cada función. La forma de hacer esto es escribiendo la directiva `#include` generalmente al principio de un programa. La tabla 3.1 muestra los archivos de cabecera que utilizaremos en este libro:

TABLA 3.1 Bibliotecas de C

Archivo de cabecera	*Descripción*
`stdio.h`	*Contiene funciones de entrada y salida de datos.*
`conio.h`	*Contiene funciones de consola que sirven para interactuar con dispositivos como la pantalla y/o el teclado.*
`math.h`	*Contiene funciones matemáticas y trigonométricas.*
`string.h`	*Contiene funciones para el manejo de cadenas de texto.*

No se explicarán todas las funciones de cada uno de estos archivos debido a que cada uno incorpora una gran cantidad de ellas y el tema está fuera del alcance de este libro, pero se estudiarán las más usadas y se irán explicando con ejemplos conforme avancemos en los temas; además existen otras bibliotecas con funciones que no se tratarán en el texto.

Las bibliotecas del estándar son: `assert.h`, `ctype.h`, `errno.h`, `float.h`, `limits.h`, `locale.h`, `math.h`, `setjmp.h`, `signal.h`, `stdarg.h`, `stddef.h`, `stdio.h`, `stdlib.h`, `string.h`, `time.h`. En la dirección electrónica, *www.acm.uiuc.edu*, es posible revisar las funciones que se encuentran en cada biblioteca, así como su sintaxis y funcionamiento.

Pero también existen las funciones externas o definidas por el usuario, que pueden ser diseñadas por cada programador de acuerdo con sus propios requerimientos. El tema de las funciones externas se trata con más detalle en el capítulo 8, dedicado a las mismas.

Es posible utilizar en un programa múltiples funciones, pero siempre debe haber una función principal, de la cual depende el control del programa completo.

En el caso de C se utiliza una función llamada `main( )`. Ésta constituye el programa principal y desde ahí se puede hacer uso tanto de las funciones internas como de las externas.

La estructura o sintaxis de un programa creado en C se muestra a continuación en un primer programa elemental.

EJEMPLO 3.1 Estructura mínima de un programa en lenguaje C, versión 1

```
/*estructura de un programa en C.*/
#include<stdio.h>
void main( )
{
  printf("Mi primer programa");
}
```


La primera línea es un comentario opcional (puede incluirse o no), ya que se encuentra entre los caracteres `/*` y `*/`. Recuerde que los comentarios son una explicación del programa y son ignorados por el compilador; en este caso se trata de una nota acerca de la estructura del programa.

La directiva `#include` de la segunda línea es necesaria para que el programa tenga acceso a las funciones del archivo `stdio.h` en el que se proporciona la información relativa a la función `printf( )` que se utiliza en la quinta línea. Observe que los símbolos < y > no son parte del nombre del archivo; se utilizan sólo para indicar que el archivo se encuentra en la biblioteca estándar `stdio.h`.

La tercera línea contiene la cabecera de la función principal `main( )`, ésta indica al compilador el comienzo del programa y requiere los paréntesis `( )` a continuación de `main`. Pero también está antecedida por la palabra reservada `void`, que es un especificador de tipo que indica que una función no devuelve valor. Esto se entenderá mejor cuando se haya estudiado el capítulo de las funciones definidas por el usuario; por lo pronto se recomienda utilizar la palabra `void` antes de `main`, aunque cabe señalar que no es obligatoria, sin embargo, si se omite o se trabaja en un compilador que no admita esta palabra reservada, entonces se deberá utilizar la palabra `return` que se explica en la segunda versión.

La cuarta y sexta líneas contienen los símbolos `{` y `}` que encierran el cuerpo de la función `main( )` y agrupan las instrucciones a ejecutar; son necesarios en todos los programas para indicar el inicio y fin respectivamente. En este ejemplo

se encuentra la instrucción `printf( )`, dando salida a la cadena "Mi primer programa". Otra forma de sintaxis de un programa podría ser:

EJEMPLO 3.2 Estructura mínima de un programa en lenguaje C, versión 2

```
/*estructura de un programa en C.*/
#include<stdio.h>
main( )
{
 printf("Mi primer programa");
 return 0;
}
```

Aunque finalmente este programa brinda el mismo resultado, hay algunas diferencias que tal vez a primera vista parezcan imperceptibles, pero que vale la pena observar.

La diferencia empieza en la tercera línea, ya que en ella se omitió la palabra `void`, motivo por el cual en la sexta línea se tuvo que hacer uso de la instrucción `return 0`. Esta instrucción indica que termina la ejecución del programa y devuelve el control al sistema operativo de la computadora, el 0 se utiliza para señalar que el programa ha terminado. Si se omite esta instrucción, luego de la compilación, C mostrará una advertencia (*warning*) indicando que la función debería regresar un valor.

Lo anterior se debe a que si se omite el indicador de tipo de una función, ésta toma el valor de entero por defecto, es decir, se espera que regrese un entero, pero como en este caso no hay nada que regresar, se usa `return 0`.

Observe también que cada instrucción finaliza con punto y coma (`;`), ya que este símbolo indica el final de las sentencias.

3.2 Directivas del preprocesador

El preprocesador forma parte del lenguaje C y se encarga de llevar a cabo una etapa que antecede a la fase de compilación. Previamente a la traducción del programa principal, se instruye al compilador para que ejecute elementos denominados directivas o directrices de procesamiento. Las dos directivas más usuales son `#define` e `#include`.

Ya se han mencionado las dos directivas anteriormente: la primera vez cuando se habló de las constantes, ya que `#define` permite sustituir un símbolo por una secuencia cualquiera de caracteres; ya se explicó que el preprocesador antes de la compilación sustituye dicha cadena de caracteres por el valor que le fue definido.

La directiva `#include` permite al compilador tener acceso al archivo fuente que viene a continuación; recordemos que estos archivos se denominan archivos de cabecera o de inclusión.

Los archivos de cabecera generalmente tienen la extensión `.h` y contienen código fuente en C.

Estos archivos se encuentran usualmente en el compilador. En realidad, la directiva del preprocesador mezcla un archivo fuente en su programa fuente.

Regularmente, los programadores de C sitúan las directivas del preprocesador al principio del programa, aunque esta posición no es obligatoria. Además, el orden de los archivos de cabecera no importa con tal que se incluyan antes de que se utilicen las funciones correspondientes. La mayoría de los programas C incluyen todos los archivos de cabecera necesarios antes de la primera función del archivo.

La directiva `#include` puede adoptar uno de los siguientes formatos:

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

La primera forma, es decir el nombre del archivo entre < > indica que los archivos se encuentran en el directorio predeterminado `include`. El segundo ejemplo muestra que el archivo está en el directorio actual, donde se encuentra el programa fuente. Estos métodos no se excluyen, y pueden existir en el mismo programa archivos de cabecera estándar utilizando corchetes angulares (<_>) y otros archivos de cabecera utilizando comillas.

3.3 Estructuras de control

Un programa está compuesto por estructuras de control y por estructuras de datos. El programa inicia con las primeras, las estructuras de control, que guían el orden de ejecución. Son tres las estructuras de control básicas: secuenciación, selección e iteración.

3.3.1 Secuenciación

Es una estructura de control en la que las instrucciones se ejecutan una después de otra en el orden en el que se encuentran escritas en un programa. El formato de una secuenciación es:

```
instrucción 1;
instrucción 2;
      .
      .
      .
instrucción n;
```

donde *n* es infinito; las instrucciones se van ejecutando desde la primera hasta la enésima instrucción, una a una.

Un programa que se diseña únicamente mediante esta estructura de control, generalmente tendrá como instrucciones entradas de datos, asignaciones de cálculos realizados con los datos de entrada y salidas de nuevos datos. A continuación se presentan algunos ejercicios resueltos.

Ejercicios resueltos

EJERCICIO 3.1 Calcule el promedio de edad de tres personas

Descripción

Solicitar las tres edades.

Aplicar la operación de promedio.

Mostrar el resultado.

Variables		
Nombre	*Tipo*	*Uso*
`E1`	`Entero`	`Primera edad.`
`E2`	`Entero`	`Segunda edad.`
`E3`	`Entero`	`Tercera edad.`
`Prom`	`Entero`	`Resultado.`

Codificación

```
/*Promedio de 3 edades*/
#include<stdio.h>
#include<conio.h>

main()
{
 int e1,e2,e3;
 float prom;
 printf("Teclea la edad de la primera persona ");
 scanf("%d",&e1);
 printf("Teclea la edad de la segunda persona ");
 scanf("%d",&e2);
 printf("Teclea la edad de la tercera persona ");
 scanf("%d",&e3);
 prom=(e1+e2+e3)/3; /*tener siempre presente la precedencia
 de operadores*/
 printf("El promedio de edad de las 3 personas es %f", prom);
}
```

Explicación

La primera línea, encerrada entre /* y */, es un comentario, no es una instrucción. La primera parte de este programa es la inclusión del archivo de cabecera `stdio.h` para poder usar las funciones `printf` y `scanf`; también se incluye la biblioteca `conio.h` y aunque en este ejemplo no se utiliza ninguna función de ésta no afecta su ejecución.

Es muy fácil identificar cuál será el orden en que se ejecutarán estas instrucciones: se pide la primera edad, se almacena en `e1`; se solicita la segunda edad y se guarda en `e2`; posteriormente se pide la tercera edad y se almacena en `e3`; después se hace el cálculo y se asigna a `prom` (en esta misma línea hay un comentario, recuerde que éste es transparente para el compilador) y por último se imprime el promedio de las edades.

La secuenciación parece sencilla y en realidad lo es, sin embargo, se trata de una estructura de control tan importante que prevalece en todos los programas, por muy grandes o pequeños que sean. Y aquí lo realmente importante consiste en encontrar la secuencia exacta para que los programas funcionen de manera eficiente.

(*continúa*)

(*continuación*)

Ejecución

```
Teclea la edad de la primera persona 5
Teclea la edad de la segunda persona 15
Teclea la edad de la tercera persona 10
El promedio de edad de las 3 personas es 10.00
```

EJERCICIO 3.2 Encuentre el área de un trapecio

Descripción

Solicitar base menor, base mayor y altura del trapecio.

Aplicar la fórmula del trapecio.

Mostrar el resultado.

Variables		
Nombre	*Tipo*	*Uso*
B	*Real*	*Base menor.*
B	*Real*	*Base mayor.*
H	*Real*	*Altura.*
R	*Real*	*Resultado.*

Codificación

```
/*Area de un trapecio*/
#include<stdio.h>
#include<conio.h>

main()
{
  float b,B,h,r;
  clrscr();
  printf("Inserte la medida de la base menor\n");
  scanf("%f",&b);
  printf("Inserte la medida de la base mayor\n");
  scanf("%f",&B);
  printf("Inserte la altura de su trapecio");
  scanf("%f",&h);
```

```
  r=(b+B)*h/2;
  printf("El area de su trapecio es %.2f",r);
  getch();
  return 0;

}
```

Explicación

Aquí, además de `stdio.h` sería obligatorio incluir el archivo de cabecera `conio.h`, ya que en él se encuentran implementadas las funciones `clrscr()` y `getch()`.

Después de la declaración de variables se limpia la pantalla, ya que a continuación está la función `clrscr()`. Luego se solicitan base mayor, base menor y altura, que se almacenan en sus respectivas variables: `b,B,h`. Se asigna el resultado de la fórmula en la variable `r` y finalmente se muestra el resultado con dos decimales, ya que la cadena de tipo `%.2f` indica que el dato a imprimir es con dos decimales. En la última línea se encuentra la función `getch()` la cual espera que se presione una tecla; esto hace que el programa no termine abruptamente, sino que cuando ejecuta el último `printf()` se quedará en pausa o en espera hasta que se presione alguna tecla.

Ejecución

```
Inserte la medida de la base menor
3.2
Inserte la medida de la base mayor
5
Inserte la altura de su trapecio
7
```

El área del trapecio es 28.7.

EJERCICIO 3.3 Calcule el salario de un trabajador con el total de percepciones y deducciones

Descripción

Solicitar cantidad de horas trabajadas.

Solicitar el sueldo que se paga por cada hora trabajada.

(continúa)

(*continuación*)

Calcular el sueldo bruto.

Calcular el monto por concepto de deducciones.

Calcular el monto por concepto de percepciones.

Calcular el sueldo neto.

Mostrar los resultados.

Variables		
Nombre	*Tipo*	*Uso*
`H`	`Entero`	`Cantidad de horas trabajadas.`
`Sh`	`Real`	`Sueldo por hora trabajada.`
`Sb`	`Real`	`Sueldo bruto.`
`D`	`Real`	`Total de deducciones.`
`P`	`Real`	`Total de percepciones.`
`Sn`	`Real`	`Sueldo neto.`

Codificación

```
/*Salario de un trabajador*/
#include <stdio.h>
#include <conio.h>

main()
{
  float sn,sb,sh,d,p;
  int h;
  clrscr();
  printf("Escriba la cantidad de horas trabajadas:\n");
  scanf("%d",&h);
  printf("Escriba la paga x hora:\n");
  scanf("%f",&sh);
  sb=h*sh;
  d=sb*.12;
  p=sb*.15;
  sn=sb+p-d;
```

```
    printf("Tu sueldo bruto es %.2f,\n tus deducciones son %.2f,
    \n tus percepciones son %.2f \n el neto es %.2f",sb,d,p,sn);
    getch();
    return 0;

}
```

Explicación

Se solicita la cantidad de horas trabajadas además de lo que se paga por cada hora, y se almacenan en *h* y *sh* respectivamente. Se calcula el sueldo bruto y se asigna a *sb*; posteriormente se asignan a la variable *d* las deducciones, las cuales son el 12% del sueldo bruto; de igual forma en *p* se guardan las percepciones que corresponden al 15% del mismo salario bruto. Se calcula el sueldo neto que consiste en el sueldo bruto menos las deducciones más las percepciones y finalmente se muestran las variables con los resultados del sueldo neto desglosando los demás conceptos.

Ejecución

```
Escriba la cantidad de horas trabajadas:
20
Escriba la paga x hora:
40
Tu sueldo bruto es 800.00
tus deducciones son 96.00
tus percepciones son 120.00
el neto es 824.00
```

EJERCICIO 3.4 Encuentre el promedio de cuatro números

Descripción

Solicitar los 4 números a promediar.

Aplicar la operación para calcular el promedio.

Mostrar el resultado.

(*continúa*)

(*continuación*)

Variables		
Nombre	*Tipo*	*Uso*
`A`	`Entero o real`	`Primero a promediar.`
`B`	`Entero o real`	`Segundo a promediar.`
`C`	`Entero o real`	`Tercero a promediar.`
`D`	`Entero o real`	`Cuarto a promediar.`
`tot`	`Real`	`Resultado.`

Codificación

```
/*Programa para promediar 4 números*/
#include<stdio.h>
#include<conio.h>

main()
{
 float a,b,c,d,tot;
 clrscr();
 printf(" Inserte el 1er numero a promediar \n\n\a");
 scanf("%f",&a);
 printf(" Inserte el 2do numero a promediar \n\n\a");
 scanf("%f",&b);
 printf(" Inserte el 3er numero a promediar \n\n\a");
 scanf("%f",&c);
 printf(" Inserte el 4to numero a promediar \n\n\a");
 scanf("%f",&d);
 tot=(a+b+c+d)/4;
 printf("El promedio es %.2f",tot);
 getch();
 return 0;

}
```

Explicación

Este programa funciona esencialmente como el promedio de las tres edades explicado anteriormente; es decir, se solicitan los números y se almacenan en `a`,

`b`, `c` y `d` respectivamente; se calcula el promedio y se asigna a `tot`, posteriormente se muestra la variable `tot` con dos decimales.

La diferencia se vería en la presentación del programa, ya que cada vez que se solicita un número a promediar, avanzaría dos saltos de línea y además se escucharía un "`bip`".

Ejecución

```
Inserte el 1er numero a promediar
3
Inserte el 2do numero a promediar
5
Inserte el 3er numero a promediar
8
Inserte el 4to numero a promediar
6
El promedio es 5.50
```


EJERCICIO 3.5 Calcule el monto de las ventas del día de una pastelería

Descripción

Definir el precio de cada producto o tamaño de pastel.

Solicitar la cantidad de pasteles grandes, medianos y chicos que se vendieron.

Solicitar también la cantidad de panes vendidos.

Calcular la venta del día multiplicando la cantidad de cada producto por su precio.

Mostrar el resultado.

Variables		
Nombre	*Tipo*	*Uso*
`G`	`Constante real`	`Precio por pastel grande.`
`M`	`Constante real`	`Precio por pastel mediano.`
`CH`	`Constante real`	`Precio por pastel chico.`
`P`	`Constante real`	`Precio por pieza de pan.`
`G`	`Entero`	`Número de pasteles grandes vendidos.`

(*continúa*)

(*continuación*)

Variables		
Nombre	*Tipo*	*Uso*
M	*Entero*	*Número de pasteles medianos vendidos.*
Ch	*Entero*	*Número de pasteles chicos vendidos.*
P	*Entero*	*Número de piezas de pan vendidos.*
Tot	*Real*	*Total de venta.*

Codificación

```
/*VENTA DE PASTELES */
# include<conio.h>
# include<stdio.h>

#define G 150.00
#define M 100.00
#define CH 50.00
#define P 2.20

main()
{
  int g,m,ch,p;
  float tot;
  clrscr();
  printf("Introduzca el numero de pasteles GRANDES vendidos\n");
  scanf("%i",&g);
  printf("Introduzca el numero de pasteles MEDIANOS vendidos\n");
  scanf("%i",&m);
  printf("Introduzca el numero de pasteles CHICOS vendidos\n");
  scanf("%i",&ch);
  printf("Introduzca el numero de PANES vendidos\n");
  scanf("%i",&p);
  tot=(G*g)+(M*m)+(CH*ch)+(P*p);
  printf("Su venta total es de %.2f",tot);
  getch();
  return 0;

}
```

Explicación

Después de los archivos de cabecera se definen las constantes `G`, `M`, `CH`, `P` con sus respectivos valores, en los cuales se almacena el precio de los diferentes tamaños de pasteles. Se solicita la cantidad de pasteles vendidos grandes, medianos, chicos y panecillos; respectivamente se guardan en `g`, `m`, `ch` y `p`. Posteriormente se asigna el resultado del cálculo del total de ventas a `tot`, el cual consiste en la suma de los productos del precio de cada pastel por el de pasteles vendidos. Al final se muestra el resultado.

Ejecución

```
Introduzca el numero de pasteles GRANDES vendidos
9
Introduzca el numero de pasteles MEDIANOS vendidos
7
Introduzca el numero de pasteles CHICOS vendidos
4
Introduzca el numero de PANES vendidos
50
Su venta total es de 2360.00
```

EJERCICIO 3.6 Realice las cuatro operaciones básicas con dos números

Descripción

Pedir dos valores con los que se llevarán a cabo las cuatro operaciones.

Sumarlos, restarlos, multiplicarlos y dividirlos, mostrando cada resultado.

Variables		
Nombre	*Tipo*	*Uso*
`a`	`Entero o real`	`Primer operando.`
`b`	`Entero o real`	`Segundo operando.`

Codificación

```
/*MINICALCULADORA*/
#include<stdio.h>
#include<conio.h>
```

(continúa)

(*continuación*)

```
main()
{
  float a,b;
  clrscr();
  printf("\nTeclea tu primer numero ");
  scanf("%f",&a);
  printf("\nTeclea tu segundo numero ");
  scanf("%f",&b);
  printf("\nLos resultados son: %.2f %.2f %.2f
  %.2f",a+b,a-b,a*b,a/b);
  getch();
  return 0;

}
```

Explicación

Este ejemplo inicia solicitando dos valores que son guardados en `a` y `b`, posteriormente muestra los resultados, pero en esta ocasión no se utiliza una variable con el fin de almacenar esos resultados, sino que se imprimen directamente (con dos decimales), usando en la función `printf` las expresiones aritméticas de la suma, resta, multiplicación y división de los dos números que introduzca el usuario. Observe que para que aparezca el resultado de una expresión aritmética, ésta tiene que estar fuera de las comillas y debe estar acompañada o precedida de su respectiva cadena de tipo de dato.

Ejecución

```
Teclea tu primer numero 32
Teclea tu segundo numero 54
Los resultados son: 86.00 -22.00 1728.00 0.59
```

Resumen

En este capítulo se estudiaron las partes mínimas que un programa en C debe tener de acuerdo con la metodología de la programación estructurada.

La programación estructurada es una técnica para crear programas mediante ciertas reglas que permiten que un programa se pueda modificar, leer y mantener fácilmente. Su característica principal es el uso de estructuras de control.

La secuenciación es la estructura de control que se explica en este capítulo y es aquella en la que las instrucciones se ejecutan una después de otra en el orden en que están escritas.

Un programa debe tener una sola entrada y una sola salida, y por lo general consta de varias funciones o subprogramas.

Existen funciones internas, las cuales ya vienen implementadas en el lenguaje, además funciones externas, que son diseñadas por el programador.

Para utilizar las funciones que ofrece el lenguaje es necesario incluir el archivo de cabecera o biblioteca (librería) correspondiente con la directiva `#include`. Una biblioteca es un archivo que contiene a esos subprogramas llamados funciones. Cada biblioteca es una colección de funciones; una de las más utilizadas es `stdio.h`, que provee las funciones de entradas y salidas estándar.

Aunque un programa puede tener múltiples funciones, debe existir la función principal `—main—`; desde ésta se utilizan las funciones internas y externas.

Evaluación

I. Conteste las siguientes preguntas.

1. La _____ es una técnica que bajo ciertas reglas permite crear, modificar, leer y mantener programas fácilmente.
2. Una _____ es un subprograma con una tarea predeterminada.
3. ¿Qué tipo de funciones se pueden encontrar en el lenguaje C? Explique la diferencia entre ambas.
4. Al conjunto de archivos donde se encuentran implementadas las funciones se le denomina _____.
5. ¿Cuál de los siguientes es ejemplo de un archivo de cabecera?
   a) `printf` b) `include` c) `conio.h` d) `define`
6. Si desea realizar operaciones matemáticas como raíz cuadrada o potencia, por ejemplo, se debe incluir la biblioteca llamada _____.

7. La biblioteca que se requiere para utilizar la función `printf` es _____.

8. ¿Cuál es el nombre de la función, a partir de la cual inicia la ejecución?
a) `void` b) `scanf` c) `string.h` d) `main`

9. ¿Es indispensable para su ejecución que un programa tenga comentarios?
a) cierto b) falso
Explique por qué _____.

10. ¿Para qué sirve la instrucción `return 0`?

11. Las instrucciones más utilizadas llamadas directivas de preprocesamiento, las cuales se ejecutan antes del proceso de traducción son _____ e inician con el símbolo _____.

12. Las _____ sirven para guiar el orden en que se ejecutan las instrucciones en un programa.

II. Resuelva el siguiente ejercicio, escriba lo que imprime el siguiente programa.

```
/* Impresion de un cheque */
# include <stdio.h>
# include <conio.h>
main()
{
  char NombreEmp[31];
  int HorasTrab;
  float CuotaHora,Sueldo;
  clrscr();
  gotoxy(18,2);
  printf("SUELDO DE EMPLEADO CON CHEQUE POR IMPRESORA");
  gotoxy(19,5); printf("--------------------------------\n");
  gotoxy(19,6); printf("³           ³");
  gotoxy(19,7); printf("³  CAPTURA DE DATOS   ³");
  gotoxy(19,8); printf("³           ³");
  gotoxy(19,9); printf("--------------------------------\n");
  gotoxy(19,10); printf("³           ³");
  gotoxy(19,11); printf("³ NOMBRE:       ³");
```

```
gotoxy(19,12); printf("³           ³");
gotoxy(19,13); printf("³ HORAS TRABAJADAS:     ³");
gotoxy(19,14); printf("³           ³");
gotoxy(19,15); printf("³ CUOTA POR HORA:      ³");
gotoxy(19,16); printf("³           ³");
gotoxy(19,17); printf("--------------------------------\n");
gotoxy(29,11); gets(NombreEmp);
gotoxy(39,13); scanf("%d",&HorasTrab);
gotoxy(37,15); scanf("%f",&CuotaHora);
Sueldo = CuotaHora* HorasTrab;
printf("-----------------------------------------------\n");
printf("³    BANCOMER S.A. DE C.V.    ³\n");
printf("³                             ³\n");
printf("³ GUADALAJARA, JAL, A 14 DE FEBRERO DE 2008. ³\n");
printf("³                             ³\n");
printf("³ PAGUESE A LA ORDEN DE: %-30s ³\n",NombreEmp);
printf("³                             ³\n");
printf("³ LA CANTIDAD DE: $ %-15.2f    ³\n",Sueldo);
printf("³                             ³\n");
printf("³ CTA. NUM. 123456  FIRMA:      ³\n");
printf("-----------------------------------------------\n");
printf("PRESIONE <CUALQUIER TECLA> PARA CONTINUAR...");
getch();
return 0;

}
```

Ejercicios propuestos

I. Codifique programas que hagan lo siguiente.

1. Calcular el área y el volumen de un cilindro.
2. Recibir una cantidad de segundos y mostrar su equivalente en el formato hrs.: min: seg.
3. Calcular el número con 10 operaciones de esta serie 4/1−4/3+4/5−4/7+4/9 …
4. Obtener, pidiendo las cantidades invertidas, los porcentajes correspondientes a cada una de tres personas que decidieron invertir conjuntamente, con respecto al total.

5. Dibujar un cuadro de asteriscos de 4 × 4.

6. Leer un nombre del teclado y escribirlo en ASCII.

II. Escriba el contenido de cada variable al ejecutarse el programa.

1.

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

main()
{
  int a=5,b=3,c=10,d=55;
  c=a+b*b-6;
  d=c-10%4*3/6;
  b=b*d+c;
  b=15;
  printf ("%d\n%d\n%d\n%d\n",a,b,c,d);
  getch();
  return 0;
}
```

a _____

b _____

c _____

d _____

2.

```
#include <stdio.h>
#include <conio.h>
#define M 3

main()
{
  int a=5,b=8,c;
  c=4*a%b;
  c=c-M;
```

```
  b=a+c-M;
  a=b*M;
  printf ("%d\n%d\n%d\n",a,b,c);
  getch();
  return 0;

}
```

a _____

b _____

c _____

3.

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

main()
{
  int a=5,b=3,c=10,d=55;
  c=a+b*b-6;
  d=c-10%4*3/6;
  b=b*d+c;
  b=15;
  printf ("%d\n%d\n%d\n%d\n",a,b,c,d);
  getch();
  return 0;

}
```

a _____

b _____

c _____

d _____

4.

```
#include <stdio.h>
#include <conio.h>
#define M 3

main()
{
  int a=5,b=8,c;
  c=4*a%b;
  c=c-M;
  b=a+c-M;
  a=b*M;
  printf ("%d\n%d\n%d\n",a,b,c);
  getch();
  return 0;
}
```

a _____

b _____

c _____

CAPÍTULO 4

Estructuras de control selectivas

Aquí se mencionan las diferentes formas de manejar una selección, clasificando la estructura por su uso común.

Las estructuras de control selectivas son aquellas que evalúan una expresión, usualmente una condición booleana, y a partir del resultado permiten tomar decisiones entre una, dos o más opciones; a esto se le conoce como selección condicional. Existen tres tipos de estructuras selectivas: **selección simple**, **selección doble** y **selección múltiple**.

Una condición booleana es una expresión que puede tener como resultado sólo el valor de verdadero o de falso.

La condición puede utilizar datos de tipo entero, real o carácter y se forman generalmente utilizando los operadores relacionales, por ejemplo a > b, 5 == b.

4.1 Estructura selectiva simple (`if`)

Es aquella que después de evaluar una condición determina su valor, que es verdadero o falso, y sólo si el resultado de la condición es verdadero se realizará la instrucción o instrucciones definidas para la condición, su sintaxis es la siguiente:

```
1.
   if (condición 1 )
    instrucción 1;
2.
   if (condición 1 )
           {
    instrucción 1;
    instrucción 2;
    instrucción 3;
           }
```

El alcance sintáctico predeterminado para una estructura de control es de una instrucción, como se aprecia en la primera forma; cuando se requiere que se ejecute más de una, las instrucciones deberán agruparse mediante llaves, tal como se puede ver en la segunda forma.

EJEMPLO 4.1 Determinar si un alumno aprobó un curso a partir del promedio que obtuvo de sus tres calificaciones de los parciales que se hicieron durante el semestre

```
#include <stdio.h>
#include <conio.h>
main()
{
 float cal1, cal2, cal3, prom;
 printf("Dame la calificacion del primer examen parcial");
 scanf("%f",&cal1);
 printf("Dame la calificacion del segundo examen parcial");
 scanf("%f",&cal2);
 printf("Dame la calificacion del tercer examen parcial");
 scanf("%f",&cal3);
```

```
 prom = (cal1 + cal2 + cal3) / 3;
 if (prom >= 60)

 printf ("Aprobo");

 getch();
 return 0;
}
```

Se puede observar que la condición se establece utilizando la variable *prom* que almacena el promedio del alumno, de tal modo que si el promedio calculado es mayor o igual que 60 la condición es verdadera y sólo entonces se realizará la instrucción asignada que es imprimir que el alumno aprobó; de lo contrario, el programa no realizará nada y simplemente terminará.

EJEMPLO 4.2 Imprimir si un número es positivo, negativo o cero

```
/*Positivo, negativo o cero*/
#include<stdio.h>
#include<conio.h>

main()
{
 int num;
 printf("Indica si el numero es positivo, negativo o cero\n");
 printf("Dame el \n");
 scanf("%i",&num);
 if(num==0)
 printf("Es cero\n");
 if(num>0)
 printf("Es positivo\n");
 if(num<0)
 printf("Es negativo\n");
 getch();
 return 0;
}
```

En este ejemplo, el usuario introducirá un número y en la pantalla aparecerá si éste es positivo, negativo o cero. Una vez que el usuario ya introdujo el número,

se almacena en la variable *num* y se evalúa la primera condición; si la variable *num* es igual que cero, entonces se imprimirá *Es cero*; después revisa la siguiente condición y si la variable *num* es mayor que cero, se imprimirá ***Es positivo*** y por último se evalúa la tercera condición y si la variable *num* es menor que cero se imprimirá *Es negativo*.

Independientemente de la condición que se cumpla, el programa evaluará las tres condiciones, pero se imprimirá el texto sólo donde la condición sea verdadera.

Ejercicios resueltos

EJERCICIO 4.1 Leer un número por el teclado y evaluar si es par o impar

Descripción

Introducir un número.

Evaluar si el número dividido entre dos tiene de residuo cero o uno.

Imprimir si el número es par o impar.

Variables		
Nombre	*Tipo*	*Uso*
n	*Entero*	*Valor a revisar*

Codificación

```
/*Par o impar*/
#include<stdio.h>
#include<conio.h>

main()
{
 int n;
 clrscr();
 printf( "Par o impar.\n\n" );
 printf( "Escribe el numero: " );
 scanf("%d",&n);
 if(n % 2 == 0)
  printf("El numero %d es par.",n);
```

```
 if (n % 2 != 0)
  printf("El numero %d es impar.",n);
 getch();
 return 0;
}
```

Explicación

Se introduce un número entero. Si al dividirse entre dos su residuo es cero, se cumple la primera condición, se imprime que el número es par y pasa a la siguiente condición. Si el número leído se divide entre dos y su residuo es diferente de cero, la condición no se cumple, por lo que no se ejecuta la siguiente instrucción y termina el programa.

Si introducimos un número impar, y al dividirse entre dos su residuo no es cero, la primera condición no se cumple y no sucede nada; enseguida se evalúa la siguiente condición; si al dividirse entre dos, su residuo es diferente de cero, se imprime que es impar y termina el programa.

Ejecución

1.
```
Par o impar.
Escribe el numero 8
El numero 8 es par.
```

2.
```
Par o impar.
Escribe el numero 15
El numero 15 es par
```

4.2 Selectiva doble (`if – else`)

Es aquella que permite evaluar una condición booleana y elegir entre dos opciones. Si la condición es verdadera, ejecutará la instrucción que se encuentra a continuación del `if`, pero si la condición es falsa se ejecutará la instrucción que se encuentra a continuación del `else`. Por lo tanto, se seleccionan las instrucciones que se encuentran a continuación del `if` o las que están después de `else`, pero no ambas. La sintaxis es:

```
1.
  if (condición )
   instrucción;
     else
   instrucción;
2.
  if (condición )
        {
   instrucción 1;
   instrucción 2;
        }
     else
        {
   instrucción 3;
   instrucción 4;
        }
```

Recuerde que para cada estructura, si existe **más de una instrucción**, éstas se deberán agrupar con las llaves. En la forma (2) anterior se agrupan tanto para el `if` como para el `else`.

EJEMPLO 4.3 Determinar si un alumno aprobó o reprobó un curso a partir del promedio que obtuvo en sus tres calificaciones parciales durante el semestre y mostrar la calificación

```
/*Aprobo o reprobo*/
#include <stdio.h>
#include <conio.h>
main()
{
 float cal1, cal2, cal3, prom;
 printf("Dame la calificacion del primer examen parcial");
 scanf("%f",&cal1);
 printf("Dame la calificacion del segundo examen parcial");
 scanf("%f",&cal2);
 printf("Dame la calificacion del tercer examen parcial");
 scanf("%f",&cal3);
 prom = (cal1 + cal2 + cal3) / 3;
 if (prom >= 60)
```

```
 printf ("Aprobo con %f",prom);
 else
 printf ("Reprobo con %f", prom);
 getch();
 return 0;
}
```

En este ejemplo nuevamente se calcula el promedio del alumno, el resultado de la variable *prom* será el valor que se utilice en la condición. Si el contenido de la variable *prom* es mayor o igual que 60 (condición verdadera) se imprime *Aprobo* y su respectivo promedio, de lo contrario *prom* se encuentra entre 0 y 59 (condición falsa), por lo que se imprimirá *Reprobo* con su promedio respectivo.

EJEMPLO 4.4 Convertir kilómetros a metros

```
/*Elegir cuál conversión realizar si de kilómetros a metros o viceversa*/
#include<stdio.h>
#include<conio.h>
main ()
{
 float z,y;
 int op;
 printf("Conversion de kms a mts o mts a kms\n");
 printf("Si deseas hacer la conversion de kms a mts elige la
   opcion 1 y de mts a kms elige la opcion 2\n");
 scanf("%d",&op);
 if(op==1)
 {
 printf("Dame los kilometros a convertir\n");
 scanf("%f",&z);
 y=z*1000;
 printf(" la conversion a metros es %f\n",y);
 }
 else
 {
 printf("Dame los metros a convertir\n");
 scanf("%f",&y);
 z=y/1000;
```

(continúa)

(*continuación*)

```
 printf("La conversion a kilometros es %f\n",z);
 }
getch();
return 0;
}
```

En este ejemplo, el usuario debe elegir la opción que desea realizar oprimiendo el número 1 o 2; esto se almacena en la variable `op`. Posteriormente el programa realizará las instrucciones correspondientes evaluando una estructura de control selectiva doble. Si la variable `op` es igual que uno se realizarán las siguientes instrucciones: aparecerá en la pantalla `Dame los kilometros a convertir` y una vez que el usuario introduzca la cantidad aparecerá en la pantalla `La conversion a metros es`, pero si la condición no es verdadera, entonces aparecerá en la pantalla `Dame los metros a convertir` y enseguida `La conversion a kilometros es` y el programa concluirá.

Nota: Cabe señalar que al elegir la opción 1, se realizará la conversión de kilómetros a metros, pero al elegir la opción 2 u oprimir cualquier tecla diferente a 1 se realizará la conversión de metros a kilómetros.

Ejercicios resueltos

EJERCICIO 4.2 Indicar si el año en que naciste fue bisiesto

Descripción

Introducir el año en que naciste.

Si el año introducido dividido entre cuatro da un residuo de cero, el año en que naciste fue bisiesto; de lo contrario, el año en que naciste no lo fue.

Variables		
Nombre	*Tipo*	*Uso*
`anio`	`Entero`	`Año de nacimiento`

Codificación

```
/* AñO BISIESTO */
#include<stdio.h>
#include<conio.h>
```

```
main(){
 int anio;
 clrscr();
 printf( "Indica si el anio en que naciste fue bisiesto\n\n" );
 printf( "En que anio naciste " );
 scanf( "%d", &anio );
 if( anio % 4 == 0 )
  printf( "El anio en que naciste es bisiesto\n\n" );
 else
  printf( "El anio en que naciste NO es bisiesto\n\n" );

 getch();
 return 0;
}
```

Explicación

Se ingresa el año en que naciste, si al dividirse entre cuatro su residuo es cero, la condición se cumple y se imprime `El anio en que naciste es bisiesto.`

Si el año introducido al dividirse entre cuatro produce un residuo distinto de cero, la condición no se cumple y se imprime `El anio en que naciste NO es bisiesto.`

Ejecución

1.
```
Indica si el anio en que naciste fue bisiesto.
En que anio naciste 1980.
El anio en que naciste es bisiesto.
```

2.
```
Indica si el anio en que naciste fue bisiesto.
En que anio naciste 1985.
El anio en que naciste NO es bisiesto.
```

EJERCICIO 4.3 Convertir grados de temperatura

Descripción

Seleccionar una opción.

Evalúa la condición.

(continúa)

(*continuación*)

Realiza el cálculo.

Muestra el resultado.

Variables		
Nombre	*Tipo*	*Uso*
`Opc`	`Entero`	`Opción de conversión.`
`g`	`Real`	`Grados.`
`con`	`Real`	`Conversión.`

Codificación

```
/*øF - øC o øC a øF */
#include<stdio.h>
#include<conio.h>

main()
{
 float g, con;
 int opc;
 clrscr();
 printf( "Conversiones de grados de temperatura" );
 printf( "\n[1]øF - øC \n[2]øC - øF\n\n" );
 printf( "Selecciona una opcion:" );
 scanf( "%d",&opc );
 if( opc == 1 )
 {
  printf( "Introduce los øF = " );
  scanf( "%f", &g );
  con = (g-32)/1.8;
  printf( "%føF = %føC", g, con );
 }
 else
 {
  printf( "Introduce los øC = " );
  scanf( "%f", &g );
  con = g * 1.8 + 32;
  printf( "%føC = %føF", g, con );
```

```
 }

 getch();
return 0;
}
```

Explicación

Se pide seleccionar un tipo de conversión; una vez elegida, se solicita la temperatura para realizar la operación de conversión, y finalmente se muestra el resultado.

Ejecución

1.
```
Conversiones de grados de temperatura.
[1]øF - øC
[2]øC - øF
Selecciona una opcion 1
Introduce los øF = .
øF = øC
```

2.
```
Conversiones de grados de temperatura.
[1]øF - øC
[2]øC - øF
Selecciona una opcion 2
Introduce los øC = .
øC = øF
```

4.3 Selectiva doble anidada

Es aquella estructura que dentro del alcance de una condición tiene otra condición; en otras palabras, en un `if - else` se encuentra otro `if - else`. Cada condición será evaluada en el orden en el que va apareciendo: si la condición 1 es verdadera se ejecuta la primera instrucción y ya no se revisan las demás condiciones, de lo contrario se evalúa la siguiente condición —la 2— y si ésta es verdadera realizará su instrucción y así sucesivamente. El `else` final que contiene la última instrucción será ejecutado si ninguna de las condiciones anteriores fue

verdadera. Esta estructura es más útil cuando se tienen tres o más opciones. Es posible utilizar tantos anidamientos como se requiera. Su sintaxis es la siguiente:

```
if (condición 1 )
  instrucción 1;
     else
if (condición 2)
instrucción 2;
     else
instrucción 3;
```

EJEMPLO 4.5 Indicar si el número leído es positivo, negativo o cero

```
/*Positivo, negativo o cero*/
#include<stdio.h>
#include<conio.h>
main()
{
 int num;
 printf("Indica si el numero es positivo, negativo o cero\n");
 printf("Dame el numero\n");
 scanf("%d",&num);
 if(num==0)
 printf("Es cero\n");
 else
 if(num>0)
  printf("Es positivo\n");
 else
  printf("Es negativo\n");
 getch();
 return 0;
}
```

Una vez almacenado el número en la variable *num*, se pasa a la siguiente instrucción donde se evalúa si *num* es igual que cero y en caso de cumplirse la condición se imprimirá *Es cero*. Si se cumplió la primera condición se sale del anidamiento y el programa termina; de lo contrario, se evaluará la siguiente condición: si *num* es mayor que cero se imprimirá *Es positivo*, se sale del anidamiento y el programa

termina. De lo contrario, se ejecutará la última instrucción donde se visualizará en la pantalla `Es negativo`.

EJEMPLO 4.6 Leer dos números y si son iguales multiplicarlos; si el primero es mayor que el segundo, que se resten; si el primero es menor que el segundo, que se sumen

```
#include <stdio.h>
#include <conio.h>
main()
{
 int num1,num2,resul;
 printf("Dame el primer ");
 scanf("%d",&num1);
 printf("Dame el segundo ");
 scanf("%d",&num2);
 if (num1==num2)
 {
 resul = num1 * num2;
printf ("La multiplicacion de los numeros es %d",resul);
 }
else
 if (num1 > num2)
 {
     resul = num1 - num2;
  printf ("La resta de los numeros es %d", resul);
 }
 else
 {
  resul = num1 + num2;

     printf ("La suma de los numeros es %d", resul);
 }
 getch();
 return 0;
}
```

Inicia pidiendo los dos valores, a continuación se compara si son iguales, si esto es verdadero, se realiza la multiplicación de los números y ya no se evalúan las

siguientes condiciones saliendo del anidamiento, si la condición 1 fue falsa se evalúa la condición 2 y se compara si el primer número es mayor que el segundo, si esto se cumple se restan los valores y se sale del anidamiento. En caso de que la condición 2 sea falsa, se ejecutan las instrucciones a continuación del último `else`.

Ejercicios resueltos

EJERCICIO 4.4 Indicar el tipo de triángulo introducido

Descripción

Introducir los tres lados del triángulo.

Comparar si los tres lados son iguales o si dos de ellos son iguales.

Imprimir el tipo de triángulo.

Variables

Nombre	*Tipo*	*Uso*
l1	*Entero*	*Lado 1 triángulo.*
l2	*Entero*	*Lado 2 triángulo.*
l3	*Entero*	*Lado 3 triángulo.*

Codificación

```
/*Clasificación de triángulos*/
#include<stdio.h>
#include<conio.h>

 main()
{
 int l1, l2, l3;
 clrscr();
 printf("Clasifica un triangulo\n\n");
 printf("Escriba los lados del triangulo: " );
 scanf("%d %d %d", &l1, &l2, &l3);
 printf("\n");
 if( l1 == l2 && l2 == l3 )
      printf( "El triangulo es Equilatero" );
```

```
else
 if( l2 == l3 || l2 == l3 || l3 == l1 )
      printf( "El triangulo es Isosceles" );
 else
      printf( "El triangulo es Escaleno" );

getch();
return 0;
}
```

Explicación

Se introducen los valores de los tres lados del triángulo. Se compara si son iguales en cuyo caso se cumple la primera condición y se imprime *El triangulo es Equilatero*. En caso contrario, se compara si dos lados son iguales; si lo son, es verdadera la condición y se imprime *El triangulo es Isosceles*. Si la segunda condición no se cumple, se imprimirá *El triangulo es Escaleno*.

Ejecución

1.
```
Clasifica un triangulo.
Escribe los lados del triangulo 7 7 7.
El triangulo es Equilatero.
```
2.
```
Clasifica un triangulo.
Escribe los lados del triangulo 5 3 9.
El triangulo es Escaleno.
```

4.4 Selectiva múltiple (switch - case)

Es aquella estructura que permite elegir entre dos o más opciones, *switch* evalúa la expresión que se encuentra dentro de los paréntesis y el resultado se compara con valores alternativos.

El *switch* en la expresión lleva implícito el operador igual (==), por lo que compara si la expresión es igual a alguna de las opciones. Por lo tanto no se puede comparar utilizando otro operador relacional.

El tipo de dato de la expresión sólo puede ser entero o carácter; por lo tanto, las opciones deberán coincidir con el tipo de dato de la expresión.

`switch` compara el valor de la expresión con cada una de las opciones en el orden en que se encuentran. Cada opción se representa con la palabra reservada `case`, por lo tanto habrá tantos `case` como opciones. Una vez que encuentra la igualdad de la expresión con una opción se realizarán las instrucciones que están a continuación del `case` hasta encontrar un `break`; si no encuentra ningún valor igual a la expresión, realizará la instrucción asignada al `default`, si éste existe.

Se deberá utilizar la palabra reservada `break` al termino de cada `case` para interrumpir la estructura y no revisar las siguientes opciones.

Como se mencionó anteriormente, el `default` se ejecutará cuando la expresión no coincida con ninguna opción. Sin embargo, será decisión del programador incluirla o no en su programa, ya que éste es opcional. La sintaxis es:

```
switch (expresion)
                {
                    case 1: instruccion 1;
                            break;
                  case 2: instruccion 2;
                            break;
                               .
                               .
                  case n: instruccion n;
                            break;
                      default:
                            instruccion n +1;
}
```

EJEMPLO 4.7 Indicar un día de la semana y que el programa escriba el número de día que le corresponde

```
#include <stdio.h>
#include <conio.h>
 main()
{
 char dia;
 printf("Elige un dia de la semana:\n Lunes = L\n Martes = M\n
   Miercoles = I\n Jueves = J\n Viernes = V\n Sabado
```

```
  = S\n Domingo = D");
scanf("%c",&dia);
switch(dia)
{
case 'L': printf ("# 1");
     break;
case 'M': printf ("# 2");
     break;
case 'I': printf ("# 3");
     break;
case 'J': printf ("# 4");
     break;
case 'V': printf ("# 5");
     break;
case 'S': printf ("# 6");
     break;
case 'D': printf ("# 7");
     break;
 }
 getch();
 return 0;
}
```

El usuario elige en un menú un nombre de día de la semana; automáticamente, aparece en la pantalla el número de día de la semana que le corresponde.

En este ejemplo, el tipo de dato de la expresión es carácter. A continuación se puede observar que la instrucción `default` es opcional, y en este ejemplo no se utilizó.

EJEMPLO 4.8 Realizar la operación que se elige del menú visualizado

```
/*Calculadora*/
#include<stdio.h>
#include<conio.h>
 main()
{
int x,y,op;
float z;
printf("Calculadora basica elige la operacion a realizar\n");
```

(continúa)

(*continuación*)

```
printf("1=Suma, 2=Resta, 3=Multiplicacion, 4=Division\n");
scanf("%d",&op);
switch(op)
{
 case 1:printf("Suma\n");
  printf("Teclea el primer \n");
  scanf("%i",&x);
  printf("Teclea el segundo \n");
  scanf("%i",&y);
  z=x+y;
  printf("La suma es%f",z);
  break;

 case 2:printf("Resta\n");
  printf("Teclea el primer \n");
  scanf("%i",&x);
  printf("Teclea el segundo \n");
  scanf("%i",&y);
  z=x-y;
  printf("la resta es%f",z);
  break;

case 3:printf("Multiplicacion\n");
  printf("Teclea el primer \n");
  scanf("%i",&x);
  printf("Teclea el segundo \n");
  scanf("%i",&y);
  z=x*y;
  printf("La multiplicacion es%f",z);
  break;

case 4:printf("Division\n");
  printf("Teclea el primer \n");
  scanf("%i",&x);
  printf("Teclea el segundo \n");
  scanf("%i",&y);
  z=x/y;
  printf("La division es%f",z);
  break;
```

```
 }
getch();
return 0;
}
```

Este programa produce una calculadora básica y despliega cuatro opciones; la opción elegida por el usuario se almacena en la variable `op`, y será la expresión que el `switch` emplee para comparar con cada una de las opciones siguientes; cuando `op` encuentre una igualdad en alguna opción, se realizarán las instrucciones que se encuentran en el `case`.

Por ejemplo, si el usuario elige la opción 3, en la variable `op` se almacenará el 3 y empezará a compararse con cada uno de los `case`. Cuando se evalúe la opción 3, el usuario habrá elegido multiplicar y mostrará `Teclea el primer`, después `Teclea el segundo` y para finalizar aparecerá `La multiplicacion es....`

Ejercicios resueltos

EJERCICIO 4.5 Elegir una figura geométrica y calcular su área

Descripción

Elegir una figura geométrica a partir del menú.

Solicitar los valores.

Mostrar el área.

Variables		
Nombre	*Tipo*	*Uso*
`n1`	`Real`	`Operando 1.`
`n2`	`Real`	`Operando 2.`
`r`	`Real`	`Resultado.`
`op`	`Carácter`	`Opción operación.`

Codificación

```
/*CALCULADORA*/
#include<stdio.h>
#include<conio.h>
```

(continúa)

(*continuación*)

```
main()
{
float n1, n2, r;
char op;
clrscr();
printf( "Figuras geometricas\n\n" );
printf( "Elija su figura ( 1. triangulo\n 2. rectangulo\n 3.
  cuadrado,\n 4. circulo ): \n\n" );
scanf( "%c",&op);
switch( op )
{
 case '1': printf("\ndame la base y la altura");
    scanf("%f%f",&n1,&n2);
    printf("\n el area es %.2f",n1*n2/2);
    break;
 case '2': printf("\ndame la base y la altura");
    scanf("%f%f",&n1,&n2);
    printf("\n el area es %.2f",n1*n2);
    break;
  case '3': printf("\ndame el lado");
    scanf("%f",&n1);
    printf("\nel area es %.2f",n1*n1);
    break;
  case '4': printf("\ndame el radio");
    scanf("%f",&n1);
    printf("\nel area es %.2f",3.1416*n1*n1);
    break;
   default: printf( "Opcion incorrecta" );
 }
getch();
return 0;
}
```

Explicación

El dato importante a conocer es el de la figura geométrica. El operador que se introduce en el ejemplo es 1, que corresponde al área del triángulo; a continuación, la variable selectora se compara con la opción 1, al ser iguales, se solicita el valor de la base y la altura, después se muestra el resultado.

Ejecución

```
Figuras geometricas
Elija su figura ( 1. triangulo
 2. rectangulo
 3. cuadrado
 4. circulo ):

1
dame la base y la altura 2 3
el area es  3.00
```

EJERCICIO 4.6 Imprimir el salario real de un trabajador

Descripción

Introducir las horas trabajadas.

Introducir el puesto que desempeña.

Indicar el salario real; si el trabajador gana más de $8,000 a la semana reducir 20% de su salario.

Variables		
Nombre	*Tipo*	*Uso*
op	Carácter	Puesto.
sal	Real	Salario total.
sdes	Real	Salario descuento.
h	Entero	Número de horas.

Constantes		
Nombre	*Tipo*	*Uso*
D	Entero	Precio hora director.
G	Entero	Precio hora gerente.
S	Entero	Precio hora supervisor.
EV	Entero	Precio hora ejecutivo.
SEC	Entero	Precio hora secretaria.

(continúa)

(*continuación*)

Codificación

```
#include<stdio.h>
#include<conio.h>
#define D 600
#define G 450
#define S 300
#define EV 150
#define SEC 80

 main(){
 float sdes, sal;
 int h;
 char op;
 clrscr();

 printf( "NOMINA\n\n" );
 printf( "Horas trabajadas en la semana: " );
scanf( "%d", &h );
printf( "Indica el puesto que desempeñas: " );
printf ( "D = Director\n G = Gerente\n S = Supervisor\n E =
  Ejecutivo de ventas\n T = Secretaria \n");
scanf( "%c", &op );
switch(op)
{
 case 'D': sal = D * h;
    break;
 case 'G': sal = G * h;
    break;
 case 'S': sal = S * h;
    break;
 case 'E': sal = EV * h;
    break;
 case 'T': sal = SEC * h;
    break;
}
if ( sal > 8000 )
{
 sdes = sal * .8;
```

```
 printf("Tu salario con el 20 por ciento de descuento es:
    $ %.2f ",sdes);
 }
 else
  printf ("Tu salario es: $ %.2f",sal);
 getch();
 return 0;
}
```

Explicación

En este programa se solicitan dos datos al usuario: la cantidad de horas trabajadas y el puesto. A continuación se calcula el salario bruto; y si éste es mayor que $8,000 se hará un descuento del 20%. Finalmente se muestra el salario neto.

Ejecución

1.

```
NOMINA
Horas trabajadas en la semana: 2
Indica el puesto que desempeñas:
D = Director
G = Gerente
S = Supervisor
E = Ejecutivo de ventas
T = Secretaria
D
Tu salario es: $1200.00
```

2.

```
NOMINA
Horas trabajadas en la semana: 30
Indica el puesto que desempeñas:
D = Director
G = Gerente
S = Supervisor
E = Ejecutivo de ventas
T = Secretaria
S
Tu salario con el 20 por ciento de descuento es: $7200.00
```

EJERCICIO 4.7 Indicar el signo zodiacal a partir de una fecha

Descripción

Introducir día y mes de nacimiento.

Mostrar el signo zodiacal que le corresponda.

Variables		
Nombre	*Tipo*	*Uso*
`dia`	`Entero`	`Dia de nacimiento.`
`mes`	`Entero`	`Mes de nacimiento.`

Codificación

```
/*ANIO BISIESTO Y SIGNO ZODIACAL*/
#include<stdio.h>
#include<conio.h>

 main(){
 int dia, mes;
 clrscr();
 printf( "Se mostrara tu signo zodiacal\n\n" );
 printf( "Cuando naciste? (dd/mm) " );
 scanf( "%d/%d", &dia, &mes );
 printf( "\nTu signo zodiacal es: " );
 switch( mes ){
  case 1: if( dia <= 20 ) printf( "Capricornio" );
       else printf( "Acuario" );
       break;
  case 2: if( dia <= 20 ) printf( "Acuario" );
       else printf( "Piscis" );
       break;
  case 3: if( dia <= 20 ) printf( "Piscis" );
       else printf( "Aries" );
       break;
  case 4: if( dia <= 20 ) printf( "Aries" );
```

```
        else printf( "Tauro" );
        break;
case 5: if( dia <= 20 ) printf( "Tauro" );
        else printf( "Geminis" );
        break;
case 6: if( dia <= 20 ) printf( "Geminis" );
        else printf( "Cancer" );
        break;
case 7: if( dia <= 20 ) printf( "Cancer" );
        else printf( "Leo" );
        break;
case 8: if( dia <= 20 ) printf( "Leo" );
        else printf( "Virgo" );
        break;
case 9: if( dia <= 20 ) printf( "Virgo" );
        else printf( "Libra" );
        break;
case 10: if( dia <= 20 ) printf( "Libra" );
        else printf( "Escorpion" );
        break;
case 11: if( dia <= 20 ) printf( "Escorpion" );
        else printf( "Sagitario" );
        break;
case 12: if( dia <= 20 ) printf( "Sagitario" );
        else printf( "Capricornio" );
        break;
}
getch();
return 0;
}
```

Explicación

Se solicita *dia* y *mes* de nacimiento. En la variable *mes*, se almacena el número de mes y a continuación se buscará la igualdad en algún *case*, en caso de encontrarla se evalúa el día, y dependiendo del resultado se muestra el signo.

(*continúa*)

(*continuación*)

Ejecución

```
Se mostrara tu signo zodiacal.

Cuando naciste? (dd/mm) 5 4
Tu signo zodiacal es Aries
```

Resumen

Las estructuras de control selectivas son aquellas que evalúan una condición booleana y a partir del resultado permiten tomar decisiones entre una, dos o más opciones; a esto se le conoce como selección condicional. Existen tres tipos de estructuras selectivas: selección simple, selección doble y selección múltiple.

La estructura de control selectiva simple es aquella que después de evaluar una condición determina su valor, que es verdadero o falso, y sólo si el resultado de la condición es verdadero se realizará la instrucción o instrucciones definidas para la condición `if`.

La estructura selectiva doble es aquella que permite evaluar una condición booleana y elegir entre dos opciones. Si la condición es verdadera ejecutará las instrucciones a continuación del `if`, pero si es falsa se ejecutará la instrucción que se encuentra a continuación del `else`.

La estructura de control selectiva doble anidada es aquella estructura que dentro del alcance de una condición tiene otra condición; en otras palabras en un `if - else` se encuentra otro `if - else`. Esta estructura es útil cuando se tienen tres o más opciones. Es posible utilizar tantos anidamientos como se requiera.

La estructura de control selectiva múltiple es aquella que permite elegir entre dos o más opciones. `switch` evalúa la expresión que se encuentra dentro de los paréntesis y el resultado de ésta se compara con valores alternativos. El tipo de dato de la expresión sólo puede ser entero o carácter; por lo tanto, las opciones deberán coincidir con el tipo de dato de la expresión. Se deberá utilizar la palabra reservada `break` al término de cada `case` para interrumpir la estructura e impedir que las siguientes opciones e instrucciones sigan ejecutándose.

Evaluación

I. Conteste las siguientes preguntas.

1. ¿Cómo se define la estructura de control selección?

2. ¿Cuántas opciones de respuesta puede tener una condición booleana?

3. ¿Es posible escribir una selección dentro de otra?

4. ¿Es válido evaluar una condición de selección y si es falsa evaluar otra condición?

5. En el `switch`, ¿es válido escribir varias opciones `case` y al final de ese conjunto escribir `break`;?

6. La instrucción `default`, ¿es indispensable en el `switch`?

7. Es la estructura selectiva que se utiliza para ejecutar una acción cuando la condición es verdadera y otra acción cuando la condición es falsa:

a) Selectiva simple
b) Selectiva doble anidada
c) Selectiva doble
d) Selectiva múltiple

8. Estructura de control que, si no encuentra ningún valor igual a la expresión, realiza la instrucción o instrucciones asignadas por `default` (si ésta existe).

a) Selectiva simple
b) Selectiva doble
c) Selectiva múltiple
d) Selectiva doble anidada

9. Estructura de control que, después de evaluar una condición, si su valor es verdadero realiza una o más acciones.

a) Selectiva simple
b) Selectiva doble
c) Selectiva múltiple
d) Selectiva doble anidada

10. Estructura de control que, dentro de una condición, tiene otra condición a evaluar. Se utiliza cuando hay tres o más opciones.

a) Selectiva simple
b) Selectiva doble
c) Selectiva múltiple
d) Selectiva doble anidada

II. Escriba un programa que muestre lo siguiente.

Simule el tradicional juego de piedra, papel o tijera utilizando las estructuras de control selectivas. Incluya la librería `time` y `stdlib`, además del comando `srand`.

En este juego de azar participan dos personas buscando un ganador. Las reglas son las siguientes:

- Tijera corta papel gana
- Papel envuelve a piedra gana
- Piedra rompe tijera gana
- Con cualquier opción diferente pierde

Ejercicios propuestos

I. Codifique el siguiente programa.

1. Mostrar los signos zodiacales, validando que el mes introducido esté entre 1 y 12, y los días entre 1 y 31, según el mes.

II. Diga cuál es el resultado de la ejecución de los siguientes programas.

1.

```
/*Compra de camisas*/
#include <stdio.h>
#include <conio.h>

main()
{
 int num_camisas;
 float prec,tot_comp,tot_pag;
 printf ("CAMISAS POLO\nIngresa el precio de la camisa ");
 scanf ("%f",&prec);
 printf ("Ingresa el numero de camisas a comprar ");
 scanf ("%d",&num_camisas);
 tot_comp=num_camisas*prec;
 if (num_camisas>=3)
 {
 tot_pag=tot_comp-tot_comp*.2;
 }
 else
 {
 tot_pag=tot_comp-tot_comp*.1;
```

```
 }
 printf ("El total a pagar es de $%f",tot_pag);
 getch();
 return 0;
}
```

2.

```
/*Descuento en el almacén*/
#include <stdio.h>
#include <conio.h>

main()
{
 float compra,desc,tot_pag;
 printf ("LINDA ALMACEN\nIngresa el total de tu compra ");
 scanf ("%f",&compra);
 if (compra>1000)
 {
 desc=compra*.2;
 }
 else desc=0;
 tot_pag=compra-desc;
 printf ("Total a pagar:\n$%f\nGRACIAS POR TU
   PREFERENCIA",tot_pag);
 getch();
 return 0;
}
```

3.

```
/*Capital con intereses*/
#include <stdio.h>
#include <conio.h>
main()
{
 float p_int,cap,tot_i,capf;
 char op;
 printf ("BANCO NACIONAL\nIngresa tu capital ");
```

```
scanf ("%f",&cap);
printf ("Ingresa la tasa de intereses ");
scanf ("%f",&p_int);
tot_i=cap*p_int;
if (tot_i>7000)
{
     printf ("Deseas reinvertir tu cap ital?\nS=Si\nN=No ");
     scanf ("%s",&op);
     switch (op)
     {
     case 'S':capf=cap+tot_i;
        printf ("TU INVERSION SERA DE $%f\nGRACIAS POR TU
PREFERENCIA",capf);
        break;
     case 'N':printf ("TU CAPITAL ESTA SEGURO\nGRACIAS POR TU
PREFERENCIA");
     }
     }
else printf ("TU CAPITAL ESTA SEGURO\nGRACIAS POR TU PREFERENCIA");
getch();
return 0;
}
```

III. Complete los siguientes programas.

1.

```
/*Hospital*/
#include <stdio.h>
#include <conio.h>
#define e1 25
#define e2 16
#define e3 20
#define e4 32

main()
{
 int tipoenf,edad,dias,costot;
 printf ("HOSPITAL CERCA DEL CIELO\nIndica tu
```

```
enfermedad\n1=Intoxicacion\n2=Gripa\n3=Alergia\n4=Diabetes ");
scanf ("___",&tipoenf);
printf ("Indica cuantos dias estuviste hospitalizado ");
scanf ("%d",&dias);
printf ("Indica tu edad ");
scanf ("%d",&edad);
switch (_____)
{
  case 1:costot=dias*e1;
  break;
  case 2:costot=dias*___;
  break;
  case 3:costot=dias*e3;
  _____;
  case 4:costot=dias*e4;
}
if ((edad>=14)___(edad<=22))
{
  costot=costot*1.1;
 }
 printf ("El costo de tu tratamiento y estancia es de $ %d",costot);
 getch();
 return 0;
}
```

2.

```
/*Llantas marca X*/
#include <stdio.h>
#include <conio.h>

main()
{
 _____ n_ll;
 float tot;
 printf ("Llantas X\n_____ ");
 scanf ("%d",&n_ll);
 if (n_ll<5)
 {
```

```
tot=n_ll*300;
printf ("El precio de cada llanta es de $300.00\nTotal a
pagar:\n$____\nGRACIAS POR TU PREFERENCIA",tot);
}
if ((n_ll>=5)&&(__________))
{
tot=n_ll*250;
printf ("El precio de cada llanta es de $250.00\nTotal a
pagar:\n$%f\nGRACIAS POR TU PREFERENCIA",tot);
}
if (n_ll>10)
{
tot=___________;
printf ("El precio de cada llanta es de $200.00\nTotal a
pagar:\n$%f\nGRACIAS POR TU PREFERENCIA",tot);
}
getch();
return 0;
}
```

3.

```
/*Horas extras*/
#include <stdio.h>
#include <conio.h>

main()
{
 int ht;
 float pph,tp,he,pe,pd,pt;
 printf ("EMPRESA PATITO\nIngresa el pago por hora ");
 scanf ("%f",_____);
 printf ("Ingresa el numero de horas trabajadas ");
 scanf ("%d",_____);
 if (ht<=40)
 {
  tp=ht*pph;
 }
 _____
```

```
he=ht-40;
if (he<=8)
{
pe=he*pph*2;
}
else
{
pd=8*pph*2;
pt=(he-8)*____*3;
pe=pd+pt;
}
_________________
printf ("El total a pagar en el sueldo es:\n$%f",tp);
getch();
return 0;
}
```

CAPÍTULO 5

Estructuras de control repetitivas

En este capítulo se analizan las formas de realizar una repetición así como sus diferentes usos y aplicaciones.

Una estructura iterativa permite repetir una acción; la repetición es controlada por una expresión que es una condición booleana. Hay tres formas de expresar las repeticiones o los ciclos: `for`, `while` y `do-while`.

El cuerpo del ciclo está constituido dentro de la estructura repetitiva; la repetición puede ser definida o indefinida. La iteración es definida cuando se conoce de antemano el número de repeticiones a ejecutar; por ejemplo, si se desea sumar cinco números, se conoce de antemano que se repetirá la acción de sumar cinco ocasiones y, para estos casos, es mejor utilizar la estructura de control **`for`**.

La iteración es indefinida cuando no se conoce de antemano el número de veces que se repetirá alguna acción; por ejemplo, cuando se pregunta al

usuario si desea repetir alguna instrucción y éste puede responder sí o no. En este caso no se conoce cuántas veces se repetirá porque esto depende de la respuesta del usuario. Para las repeticiones indefinidas es mejor utilizar las estructuras ***while*** y ***do-while***.

Es posible usar las estructuras de control *while*, *do-while* y *for* casi en forma indiferente.

5.1 Estructura repetitiva while

En esta estructura, la repetición se realizará tantas veces como se indique mientras se cumpla una condición. La cantidad de repeticiones puede ser definida o indefinida. La representación de la estructura es la siguiente:

```
expresion 1;
while (expresion 2)
   {
   instruccion 1;
   expresion 3;
   }
```

donde:

expresion 1	siempre será el valor de inicio de la variable de control (asignación).
expresion 2	es la condición booleana.
expresion 3	es la forma en que cambia la variable de control (asignación).
instruccion 1	instrucciones a ejecutar

EJEMPLO 5.1 Imprimir los números enteros del 1 al 10

```
#include<stdio.h>
#include<conio.h>
main()
{
 int c;
 clrscr();
 c=1;
 while(c<=10)
 {
  printf(" %d ",c);
  c=c+1;
```

```
    }
    getch();
    return 0;
}
```

La variable de control es `c`, que también servirá para generar los números. Se inicializa en 1, ya que es el primer valor a imprimir. Dentro del ciclo se imprime `c` y se incrementa de uno en uno.

El ciclo repetitivo está controlado por `c`; la repetición se efectuará mientras el valor del contador sea menor o igual que 10. Por ello es una repetición definida.

En este ejercicio, la variable de control es un **contador**, y se le llama así a la variable que sirve para incrementar de uno en uno. El contador debe tener las siguientes características: su valor de inicio es 1, y se incrementa o disminuye de uno en uno, aunque en ocasiones se utiliza otra proporción fija (de cinco en cinco, de tres en tres, etc.). En la condición se debe evaluar el valor del contador, para determinar cuándo debe finalizar el ciclo.

También es necesario agrupar con llaves las instrucciones `printf(" %d ",c);` y `c = c+1;` ya que de cumplirse la condición es necesario efectuar las dos instrucciones.

Al igual que en otras estructuras de control, el alcance de una condición predeterminada es una instrucción; por tal razón si existe más de una al repetirse éstas deben agruparse con llaves.

5

EJEMPLO 5.2 Sumar los números enteros del 1 al 5 e imprimir el resultado

```
#include<stdio.h>
#include<conio.h>
main()
{
 int I, S;
 clrscr();
 I=1; S=0;
 while(I<=5)
 {
 S = S+I;
 I = I+1;
```

(continúa)

(*continuación*)

```
}
printf("La suma de los numeros del 1 al 5 es %d ",S);
getch();
return 0;
}
```

Este programa utiliza una repetición mediante el uso del contador para indicar el número de ocasiones que se ejecuta el ciclo. La letra `I` servirá para generar los números del 1 al 5, se debe inicializar el valor de `I` y de `S` antes de la condición, `S` guardará la suma de los números generados por `I`, y ésta se incrementará en uno, el ciclo se repetirá mientras `I` sea menor o igual que 5. Se muestra la suma fuera de la condición, ya que el resultado se imprimirá sólo una vez.

Aquí también se utiliza un **acumulador** —una variable que permite sumar o acumular mediante otra operación una serie de números— que debe iniciarse en cero; de lo contrario comenzará con el valor almacenado en la memoria anteriormente (basura).

EJEMPLO 5.3 Hallar el producto de varios números positivos introducidos por teclado y terminar el proceso cuando se contesta con una letra diferente a *s*

```
#include<stdio.h>
#include<conio.h>
main()
{
 int p,num;
 char resp='s';
 clrscr();
 p=1;
 printf(" Dame un numero");
 scanf("%d",&num);
 while(resp=='s')
 {
 p =p*num;
 printf(" \n dame otro numero ");
 scanf("%d",&num);
 printf(" \n Ingresar otro numero s/n ");
 scanf("%c",&resp);
```

(*continúa*)

```
 }
 printf(" el total de la multiplicacion es %d ",p);
 getch();
 return 0;
}
```

En este ejemplo, la variable de control es `resp`, una forma de iteración indefinida. Aquí `resp` es una **bandera** o **centinela**, un tipo de variable empleado cuando el ciclo es indefinido, o sea que la repetición se hará de acuerdo al valor contenido, que no depende del número de iteraciones. El valor de la bandera debe ser diferente a los datos utilizados.

La variable `p` se utiliza como acumulador y guarda el resultado de la multiplicación de los números; se inicializa con 1 antes de entrar al ciclo (es importante que `p` inicie en uno para no modificar la multiplicación del primer número). Fuera del ciclo se solicita el primer número, y ya dentro del ciclo se pide otro número a multiplicar, se lee, se multiplica y se almacena. Enseguida pregunta si se desea introducir otro número, lo que repite hasta que el contenido de `resp` sea diferente a `s`. Al salir del ciclo, se muestra el resultado de la acumulación del producto.

EJEMPLO 5.4 Producir una tabla de multiplicar e imprimirla en la pantalla utilizando la estructura de control `while`

```
#include<stdio.h>
#include<conio.h>
main()
{
 int i,n; clrscr();
 printf("DAME EL NUMERO DE TABLA : ");
 scanf("%d",&n);
 i=1;
 while(i<=10)
 {
      printf("\n %d*%d=%d ",n,i,n*i);
   i++;
 }
 getch();
 return 0;
}
```

Se declaran dos variables: la `n` para indicar el número de la tabla que se va a elaborar y la `i` para realizar el ciclo de 1 a 10 para la multiplicación; `n` entonces guarda el número de la tabla. El ciclo que va de 1 a 10 muestra el número `n`, el número `i` y el resultado de multiplicar `n*i`.

Este ejemplo se mostrará también con las demás estructuras de control repetitivas.

Es importante observar que en todos estos ejemplos de la estructura de control `while`, no se escribe punto y coma al final de la condición, ya que esto indica que la estructura de control termina en ese punto y no repite las instrucciones. A continuación se muestra el programa de la tabla de multiplicar, pero con un punto y coma al final de la condición. ¿Qué imprimirá el ejemplo?

EJEMPLO 5.5 Tabla de multiplicar con un error de lógica

```
#include<stdio.h>
#include<conio.h>
main()
{
 int i,n; clrscr();
 printf("DAME EL NUMERO DE TABLA DE MULTIPLICAR: ");
 scanf("%d",&n);
 i=1;
 while(i<=10);
   {
       printf("\n %d*%d=%d ",n,i,n*i);
    i++;
   }
 getch();
 return 0;
}
```

El compilador no marcará error, ya que no es error de sintaxis, pero no terminará la ejecución del programa ("entra en ciclo") porque no cambia el valor de la `i` contenido en la condición. La forma correcta no lleva el punto y coma al final de `while`.

Ahora se compara el comportamiento de estos tres fragmentos de programa, que aunque son similares producen diferente resultado.

```
1)
c=10;
while (c<21)
{
 printf(" %d ",c);
 c=c+2;
}

2)
c=10;
while (c<21)
{
 c=c+2;
 printf(" %d ",c);
}

3)
c=10;
while (c<21)
 printf(" %d ",c);
 c=c+2;
```

En el caso 1 se imprimirá `10 12 14 16 18 20`, ya que las instrucciones repetidas son imprimir y luego incrementar `c` en dos. En el caso 2 se imprimirá `12 14 16 18 20`, ya que primero se indica el incremento de la variable `c` y después se imprime. Finalmente, en el caso 3 `10 10 10 10...` se imprimirá de forma indefinida ("entra en ciclo"), ya que se repetirá solamente la primera instrucción después de la condición porque no hay llaves que agrupen las dos instrucciones.

EJEMPLO 5.6 Sumar los números pares y multiplicar los números impares hasta que la suma sea mayor que 50 y el producto sea mayor que 150

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

 int num,suma=0,prod=1;
 clrscr();
```

(continúa)

(*continuación*)

```
while(suma<=50 || prod<=150)
{
printf("\n \t DAME EL NUMERO ENTERO: ");
scanf("%d",&num);
if(num%2==0)
  suma=suma+num;
 else
  prod=prod*num;
}
printf("\n \t LA SUMA ES: %d " ,suma);
printf("\n \t EL PRODUCTO ES : %d ",prod);
getch();
return 0;
}
```

En este ejemplo se usan dos variables de control que son dos acumuladores, dado que la iteración depende de dos condiciones: que *sum* sea menor o igual que 50 y que *prod* sea menor o igual que 150. Para unir la dos se utiliza el operador || (or) que funciona de la siguiente forma: evalúa y si una de las condiciones es verdadera se ejecuta de nuevo el ciclo, el ciclo termina cuando ambas condiciones sean falsas.

Una vez capturado un número en la variable *num*, se pregunta si al realizarse la división de *num* entre dos el residuo es igual que cero; si la condición es verdadera se suma *num* a suma, de lo contrario se multiplica *num* por *prod*. Y esto se repite mientras el contenido de *sum* sea menor o igual que *50 o prod* sea menor o igual que *150*.

Para finalizar el tema, debemos señalar que las condiciones pueden utilizar como variable de control contadores o acumuladores. El *contador* es una variable que se incrementa o disminuye en cantidades fijas en cada iteración (de 2 en 2, de 5 en 5, etc.). El *acumulador* es una variable que se incrementa o disminuye en cantidades diferentes en cada iteración. La *bandera* sirve para indicar estados de verdadero o falso de la condición.

Ejercicios resueltos

EJERCICIO 5.1 Encontrar cuatro múltiplos de un número cualquiera

Descripción

Leer un número.

Leer números arbitrarios.

Verificar que el número sea múltiplo del primero.

Si es múltiplo imprimirlo.

Variables		
Nombre	*Tipo*	*Uso*
num	*Entero*	*Número del que se deben obtener múltiplos.*
numC	*Entero*	*Número a evaluar.*
cont	*Entero*	*Cuenta los múltiplos.*

Codificación

```
#include<stdio.h>
#include<conio.h>
main()
{
 int num, numC, cont=0;
 clrscr();
 printf(" Dame un numero para buscar sus multiplos ");
 scanf("%d",&num);
 while(cont<=4)
 {
 printf(" \n\tDame un numero");
 scanf("%d",&numC);
 if(numC%num==0)
  {
  printf("%d es multiplo ",numC);
  cont++;
  }
 }
 getch();
 return 0;
}
```

Explicación

Se solicita un número y se guarda en *num*, que servirá para buscar sus múltiplos. A continuación se analiza la condición en la que se pregunta si *cont* es menor o igual que 4; *cont* almacena la cuenta del número de múltiplos encontrados.

(*continúa*)

(continuación)

Se guarda un número en `numC`, luego se evalúa si el residuo de `numC` dividido entre `num` es igual que cero; si es verdadero se muestra el múltiplo y se incrementa `cont`. Se repite esta operación hasta que el contador `cont` llegue a 4.

Ejecución

```
Dame un numero para buscar sus multiplos
5
7
6
6 es multiplo
9
154
154 es multiplo
8
8 es multiplo
10
10 es multiplo
```

5.2 Estructura repetitiva do-while

En esta estructura `do-while`, la condición de continuación del ciclo se prueba al final del mismo. Funciona de manera similar a la estructura `while`; la diferencia es que una evalúa al inicio del ciclo y la otra al final. En esta estructura es indispensable escribir las llaves, aunque pudiera parecer innecesario utilizarlas. También se debe notar que al final de la condición `do-while` se escribe punto y coma. La representación de la estructura es la siguiente:

```
expresion 1;
do
{
instruccion 1;
expresion 3;
} while(expresion 2);
```

donde:

`expresion 1` es la asignación de inicio.
`expresion 3` es la asignación de variación.

instruccion 1 son las instrucciones a ejecutar.
expresion 2 debe ser una expresión booleana.

La estructura tiene los mismos elementos que utiliza la estructura *while*. Como se mencionó anteriormente, estas estructuras funcionan de forma similar.

El *do-while* tiene una aplicación muy usual: cuando se requiere repetir por lo menos una vez un programa. Observe el siguiente ejemplo:

EJEMPLO 5.7 Obtener el promedio de una determinada cantidad de números leídos desde el teclado

```
#include<stdio.h>
#include<conio.h>
main()
{
 float num,cuenta,sum;
 clrscr();
 sum=0;cuenta=0;
 do
 {
 printf("\n dame un numero (para detener -1) ");
 scanf("%f",&num);
 sum=sum+num;
 cuenta=cuenta+1;
 }while(num!=-1);
 printf("\nEl promedio de los %.0f numeros es %.2f ",cuenta-1,sum-num/
(cuenta-1));
 getch();
 return 0;
}
```

El programa utiliza dos variables: *sum* que sirve para sumar los números que se van ingresando (acumulador) y *cuenta* para contar la cantidad de números ingresados (contador). Se van solicitando los números y se suman; cuando escriba *-1* se detiene el ingreso de éstos y en ese momento *num* funciona como bandera. Para que la operación sea correcta se debe restar el –1 a la variable *sum* y también se debe restar 1 a la variable *cuenta*.

EJEMPLO 5.8 Calcular el pago a realizar según los litros de gasolina

```
#include<stdio.h>
#include<conio.h>
#define precio 5
main()
{
 float l;
 char R;
 clrscr();
 do
 {
 printf(" cuantos litros son ");
 scanf("%f",&l);
 printf("\n el pago es de %.2f ",l*precio);
 printf("\n Otro calculo s/n ");
 scanf("%c",&R);
 }while(R=='s');
 getch();
 return 0;
}
```

Este programa controla la repetición mediante el centinela (o bandera) `R`. Recuerde que lo importante con respecto al uso del centinela es que debe estar definido de forma que no se confunda con otros datos.

En una constante llamada `precio` se almacenó el valor por litro. Se solicita el número de `litros` y se realiza el cálculo del pago. Después de la operación se pregunta si se desea realizar otro cálculo, almacenando la respuesta en la variable `R`. Como se aprecia, no se conoce de antemano cuántas veces se repetirá el programa, pero se garantiza que al menos una vez se ejecutará el cuerpo del ciclo.

EJEMPLO 5.9 Imprimir en pantalla la tabla de multiplicar de un número tecleado por el usuario, utilizando la estructura de control do-while

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

```
int i,n; clrscr();
printf("DAME EL NUMERO DE LA TABLA DE MULTIPLICAR: ");
scanf("%d",&n);
i=1;
do {
      printf("\n %d*%d=%d ",n,i,n*i);
   i++;
  } while(i<=10);
getch();
return 0;
}
```

Aquí se muestra *DAME EL NUMERO DE LA TABLA DE MULTIPLICAR* y genera una del 1 al 10. Se declaran dos variables: la *n* para indicar el número de la tabla que se va a elaborar y la *i* para realizar el ciclo de 1 a 10. Conforme avanza el ciclo se imprime el resultado de multiplicar *n*i*. Observe las diferencias en el uso de la estructura *while* y *do-while* para este ejemplo.

Ejercicios resueltos

EJERCICIO 5.2 Adivinar en un máximo de cinco oportunidades un entero comprendido entre 1 y 100. En cada ciclo la computadora debe decir si el que se captura es mayor o menor que el que generó automáticamente

Descripción

Generar un número aleatorio.

Solicitar un número al usuario.

Compararlos.

Mostrar el resultado.

Variables		
Nombre	*Tipo*	*Uso*
c	*Entero*	*Contador de iteraciones.*
nusuario	*Entero*	*Número generado por el usuario.*
numcom	*Entero*	*Guarda el número generado.*

(*continúa*)

Codificación

(*continuación*)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
main()
{
 int c=0,nusuario,numcom; clrscr();
 randomize()
 numcom=1+random(100);
 do {
      printf("\n Dame un numero entero: ");
   scanf("%d",&nusuario);
   if(nusuario>numcom)
     printf("\n tu numero es mayor al mio");
   else
     printf("\n tu numero es menor al mio");
     c=c+1;
  }
 while( nusuario!=numcom && c<5);
 if (nusuario==numcom)
  printf("\n ¡Felicidades! Lo lograste en %d intentos",c);
 else
  printf(" Mejor suerte para la proxima");
 getch();
 return 0;
}
```

Explicación

Aquí se utiliza la biblioteca `stdlib.h`, que permite el uso de las funciones `randomize()` y `random();` la primera inicializará el generador de números pseudoaleatorios, con una semilla que va en función de la hora actual, lo que permite que no se repitan las mismas secuencias de números aleatorios. La segunda función produce un pseudoaleatorio en el rango de 0 a x-1.

Por tal razón se escribe primero la generación de un número pseudoaleatorio y enseguida con `random(100)` se genera el número entre 0 y 99, por lo que agregamos 1 para que quede el valor de `numcom` entre 1 y 100. Uno de los operadores que se utiliza es `!=` que significa diferente y se lee: si `nusuario` es

diferente de *numcom*. Observemos que la condición está compuesta por dos enunciados, unidos por el operador `&& (and)`, que deben ser verdaderos para que se repita el ciclo.

Al momento de ser iguales *nusuario* y *numcom* se imprime la palabra *felicidades* y el número de intentos; en caso contrario imprime la leyenda *"Mejor suerte para la proxima"*.

Ejecución

(Ejemplo si se genera aleatoriamente el 11).

```
Dame un numero entero: 50
tu numero es mayor al mio
Dame un numero entero: 5
tu numero es menor al mio
Dame un numero entero: 21
tu numero es mayor al mio
Dame un numero entero: 10
tu numero es menor al mio
Dame un numero entero: 11
¡Felicidades! Lo lograste en 5 intentos
```


Ejercicios resueltos

EJERCICIO 5.3 Leer números desde el teclado y sumar los primeros cinco impares

Descripción

Pedir un número.

Verificar si es impar y sumarlo.

Mostrar resultado.

Variables		
Nombre	*Tipo*	*Uso*
`suma`	`Entero o real`	`Guarda la suma de los números impares.`
`num`	`Entero`	`Es el número leído del teclado.`
`cont`	`Entero`	`Cuenta la cantidad de números impares.`

(*continúa*)

(*continuación*)

Codificación

```
/* Sumar cinco números impares, leídos desde el teclado. */
#include<stdio.h>
#include<conio.h>

main()
{
 int suma=0, num, cont=0 ;
 clrscr();
 do{
 printf("\n Dame un numero ");
 scanf("%d",&num);
 if(num%2==1)
  {
  suma+=num;
  cont++;
  }
 }while(cont<5);
 printf("\n\n\t La suma de 5 numeros impares es %d ",suma);
 getch();
 return 0;
}
```

Explicación

Se inicializan las variables *suma* y *cont* en 0. La estructura repetitiva *do-while* permitirá repetir la ejecución mientras *cont* sea menor que 5.

La repetición consiste en pedir un número y almacenarlo en *num*, se evalúa si el residuo de la división del contenido de *num* entre dos es uno: si es verdadero, el número se acumula en la variable *suma* y el contador se incrementa. Se evalúa si *cont* es menor que 5. Se repetirán las instrucciones hasta que *cont* llegue a 5.

Ejecución

```
Dame un numero 2
Dame un numero 5
Dame un numero 3
Dame un numero 1
```

```
Dame un numero 6
Dame un numero 9
Dame un numero 4
Dame un numero 1

   La suma de 5 numeros impares es 29
```

5.3 Estructura repetitiva for

La estructura de control *for* se utiliza generalmente cuando la repetición está definida. Esta estructura maneja todos los detalles de la repetición controlada por contador. La representación de la estructura repetitiva es la siguiente:

```
for ( expresion 1; expresion 2; expresion 3)
                instruccion 1
```

donde:

expresion 1	es el nombre de la variable de control y su valor de inicio.
expresion 2	es la que evalúa la condición de control.
instruccion 1	son las instrucciones que han de repetirse.
expresion 3	instrucción de incremento o decremento de la variable de control.

La *expresion 1* y *expresion 3* siempre serán asignaciones, mientras que *expresion 2* siempre tendrá que ser una expresión booleana. Al igual que en las anteriores estructuras repetitivas, las partes de la estructura *for* son similares.

Las estructuras repetitivas pueden ser equivalentes en ciertos casos. Lo ideal es usar *for* cuando la repetición es definida, y las otras dos cuando la repetición es indefinida. *while* generaliza la repetición, y las otras dos estructuras son casos particulares del uso de ella, y facilitan la repetición.

EJEMPLO 5.10 Imprimir en pantalla los primeros 15 números positivos enteros en orden decreciente

```
#include<stdio.h>
#include<conio.h>
main()
 {
 int i;
 clrscr();
```

(continúa)

(*continuación*)

```
for(i=15;i>0;i--)
 printf("%d ",i);
getch();
return 0;
}
```

La variable de control es `i` y además servirá para imprimir los `s`. El número con que inicia, en este caso, es 15. A continuación se evalúa la expresión 2 para verificar la continuación del ciclo, que cumple la condición de ser mayor a 0. Finalmente se evalúa la expresión 3 para determinar la forma en que se disminuye la variable; en este caso, el decremento es de 1. Se continúa de esta manera hasta que `i` vale 0.

EJEMPLO 5.11 Imprimir todas las letras del alfabeto de forma inversa

```
#include<stdio.h>
#include<conio.h>
main()
{
 char letra;
 clrscr();
 printf("Estas son las letras del alfabeto: ");
 for(letra='Z';letra>='A';letra--)
  printf("%c\t",letra);
 getch();
 return 0;
 }
```

Este programa genera las letras del alfabeto, iniciando con la 'Z' y terminando con la 'A'. El lenguaje C, reconoce los caracteres del alfabeto por su código ASCII.

Se declara la variable `letra`, de tipo carácter, que inicia con el valor de Z.
La condición indica que será válida mientras `letra` sea un valor mayor o igual que A, y se disminuye de uno en uno.

EJEMPLO 5.12 Imprimir en pantalla la tabla de multiplicar de un número utilizando la estructura de control for

```
#include<stdio.h>
#include<conio.h>
main()
{
 int i,n; clrscr();
 printf("DAME EL NUMERO DE TABLA DE MULTIPLICAR : ");
 scanf("%d",&n);
 for(i=1;i<=10; i++)
      printf("\n %d*%d=%d ",n,i,n*i);
 getch();
 return 0;
}
```

Como se observa, en comparación con las otras estructuras de control repetitivas, en una sola línea se controla el ciclo; ya que se inicializa la variable se verifica la condición y la forma de incrementar la variable de control.

Se pregunta al inicio el número de la tabla a multiplicar, se evalúa la estructura y se genera la misma del 1 al 10. Se declaran dos variables *n* para indicar el número de la tabla que se va a hacer y la *i* para realizar el ciclo de 1 a 10 para la multiplicación, y dentro del ciclo se muestra el resultado de multiplicar *n*i*.

Es importante mencionar que si se requiere la repetición de más de una línea es necesario escribir entre llaves las instrucciones.

En la estructura de control *for* se puede omitir cualquiera de las tres expresiones e incluso las tres. Si se omite la expresión1 es porque se inicializó la variable de control en alguna otra parte del programa, si se omite la expresion2, se supone que la condición es verdadera, creando un ciclo infinito, y si se omite la expresión3, será porque el incremento se da dentro del bucle.

Resumiendo, la definición de *while*, *do-while* y *for* puede ser equivalente en ciertos casos. A continuación se comparan las tres.

```
1.
for(expresion1; expresion2; expresion3)
instrucción 1;

2.
expresion1;
while (expresion2)
{
instruccion 1;
expresion3;
}

3.
expresion1;
do {
instruccion 1;
expresion3;
} while (expresion2);
```

Ejercicios resueltos

EJERCICIO 5.4 Crear un marco en la pantalla utilizando asteriscos en las coordenadas (1,1), (1,80), (80,24) y (1,24)

Descripción

Dibujar una línea horizontal en la parte superior de la pantalla.

Dibujar una línea vertical en el lado derecho de la pantalla.

Dibujar una línea horizontal en la parte inferior de la pantalla.

Dibujar una línea vertical en el lado izquierdo de la pantalla.

Variables		
Nombre	*Tipo*	*Uso*
`I`	`Entero`	`Guarda el contador.`

Codificación

```
//marco de la pantalla con asteriscos

#include <stdio.h>
#include <conio.h>
#include <dos.h>
main()
{
 int i;
 clrscr();
 for (i=1; i<=80; i++)
 {
      gotoxy(i,1);
      printf("*");
      delay(10);
 }
 for (i=1; i<=24; i++)
 {
      gotoxy(80,i);
      printf("*");
      delay(20);
 }
 for (i=80; i>=1; i--)
 {
      gotoxy(i,24);
      printf("*");
      delay(10);
 }
 for (i=24; i>=1; i--)
 {
      gotoxy(1,i);
      printf("*");
      delay(20);
 }
 getch();
 return 0;
}
```

(continúa)

(*continuación*)

Explicación

En este caso no se leen valores desde el teclado ya que se conocen las coordenadas del margen: el extremo superior izquierdo tiene las coordenadas 1,1 y el inferior derecho 80,24; además, el símbolo a dibujar ya se definió y es un asterisco. Los ciclos van de 1 a 80 para la parte superior e inferior, y de 1 a 24 para los lados.

Ejecución

```
**********************
*                    *
*                    *
*                    *
***********************
```

Nota: La imagen se mostrará según la dimensión y resolución del monitor.

Resumen

En este capítulo se revisaron las opciones del lenguaje C para ejecutar varias veces una instrucción. La iteración consiste en repetir una o varias instrucciones, y es controlada con una expresión que usualmente es una condición booleana.

Cuando se conoce de antemano cuántas veces se realizará la repetición se le llama iteración definida; cuando no se conoce el número de repeticiones se le llama iteración indefinida.

En lenguaje C hay tres formas de realizar las repeticiones y éstas tienen tres partes: el valor de inicio de la variable de control, la que indica cómo va cambiando el valor y la constituida por la condición o valor de verdad. Según la estructura que se utiliza es el orden en que aparecen, en algunos casos se puede omitir una, dos o incluso las tres partes, dependiendo de lo que se desea realizar.

A la variable de control se le llama también `contador` cuando cuenta el número de iteraciones y se incrementa de uno en uno; y `acumulador` cuando permite sumar una cantidad constante en cada iteración. Cuando la variable almacena un valor booleano (falso o verdadero) se le llama `bandera`.

Evaluación

I. Describa los siguientes conceptos.

1. ¿Qué es un ciclo o repetición?
2. ¿Qué es un ciclo definido?
3. ¿Qué es un ciclo indefinido?
4. ¿Qué estructuras de control hay para la repetición en la programación estructurada?
5. ¿Para qué se usa un contador?
6. ¿Para qué se usa un acumulador?
7. ¿Cuándo se hace un ciclo infinito?
8. ¿En qué caso se recomienda usar `while` y cuando `do-while`?
9. ¿Qué sucede si al finalizar el encabezado del `for` se coloca ; (punto y coma)?
10. ¿Qué sucede si al finalizar el encabezado del `while` se coloca ; (punto y coma)?
11. ¿Al finalizar el encabezado de `do-while` se debe escribir ; (punto y coma)?

II. Resuelva el siguiente ejercicio.

1. Realice con un símbolo un espiral, iniciando en el centro del monitor, y hasta que se llene éste; se debe controlar el ancho y alto de cada vuelta, así como el tamaño máximo a cubrir en el monitor.

```
* * * *
* * * *
* * * *
    * *
```

Ejercicios propuestos

I. Codifique los siguientes programas.

1. Imprimir los números del 1 al 100.
2. Imprimir la serie de 0 1 1 2 3 5 8 … n (inicia con 0 1, y el siguiente número es la suma de los dos anteriores) hasta un límite definido por el usuario.

3. Realizar una división y repetir la operación hasta que uno de los números sea 0.
4. Imprimir los colores usando los números del 1 al 100 con la función `textcolor()`.
5. Mostrar un "*" rebotando por la pantalla hasta presionar una tecla.
6. Mostrar la suma de los números entre 10 y 125.
7. Escribir la suma de la serie 1/2 + 1/3 + 1/4 + ... + 1/20.
8. Escribir un programa que muestre el mayor de una serie de números introducidos por el teclado.
9. Calcular la suma de una serie de números leídos por teclado.
10. Escribir un programa que muestre el mayor, el menor y la media de una serie de números introducidos por el teclado.
11. Mostrar el resultado de sumar los números pares e impares que hay entre 10 y 125.
12. Mostrar la suma de 10 números enteros consecutivos.
13. Imprimir un rango de números enteros solicitando el inicio y fin de los valores.
14. Mostrar el cuadrado de los primeros cinco números naturales.
15. Mostrar los números primos entre el 1 y el 125.
16. Imprimir todos los números inferiores a uno introducido por el teclado.
17. Imprimir en pantalla los múltiplos de 5 que hay entre 1 y 1000.
18. Multiplicar entre sí los números entre 1 y 10 y mostrar el resultado.
19. imprimir el factorial de los números comprendidos entre el 2 y el 15.
20. Imprimir un triángulo isósceles.
21. Imprimir los múltiplos de un número entre 1 y 10, menores que 125.

II. Complete las líneas en blanco.

1. Sumar los 100 primeros números enteros positivos.

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

```
main()
{
 int _______,_______;
 suma=0;
 clrscr();
 for(contador=1; _________ ;contador++)
 suma=suma+contador;
 printf("La suma es :_____",suma);
 getch();
 return 0;
}
```

2. Elevar *X* a una potencia *Y*.

```
#include<stdio.h>
#include<conio.h>
main()
 {
 int cont,x,y,res;
 clrscr();
 res=_____;
 printf("\nEscriba el No. que se elevara a una potencia: ");
 scanf("%d",_____);
 printf("\nA que potencia elevara el numero %d: ",x);
 scanf("%d",&y);
 for(_____;cont<=y;cont++)
       res=res*x;
 printf("\nEl %d elevado a la potencia %d es: %d ",_______);
 getch();
 return 0;
 }
```

3. Calcular el factorial de un número.

```
#include<stdio.h>
#include<conio.h>
main()
{
 int cont, x;
```

```
long ____;
clrscr();
printf(" factorial del numero..? ");
scanf("_____",&x);
Fact =1;
for(cont=1;cont<=x;_____)
fact=cont*fact;
printf("\n factorial de %d = %d", x,fact);
getch();
return 0;
}
```

4. Imprimir los números del 100 al 0.

```
#include<stdio.h>
#include<conio.h>
main()
{
____ c;
clrscr();
c=100;
while(_____)
{
printf("%d ",c);
_____
}
getch();
return 0;
}
```

5. Imprimir la suma de los 100 primeros números.

```
#include<stdio.h>
#include<conio.h>
main()
{
 int c,suma;
 c=____
 _____=0;
```

```
clrscr();
while(c<=100)
{
_____=suma+c;
_____=c+1;
}
printf("la suma de los primeros cien numeros es %d ",suma);
getch();
return 0;
}
```

6. Imprimir los números impares hasta el 100 e imprimir cuántos impares hay.

```
#include<stdio.h>
#include<conio.h>
main()
{
_____ c,son;
clrscr();
 c=____;
son=____;
while(c<100)
{
printf(" %d",c);
c=_____;
son=son+1;
}
printf(" \nEl numero de impares es %d ", son);
getch();
return 0;
}
```

7. Imprimir los números del 1 al 125. Calcular la suma de todos los números pares por un lado y, por otro, la de todos los impares.

```
#include<stdio.h>
#include<conio.h>
main()
```

```
{
 int i, sumapar, sumaimpar;
 clrscr();
 sumapar=____;
 sumaimpar=_____;
 i=1;
 do
 {
 if(i%2==0)
  sumapar=sumapar+____;
 else
  sumaimpar=sumaimpar+i;
 i=i+1;
 }while(i<=125);
 printf("\nLa suma de los numeros ________ es %d ",sumapar);
 printf("\nLa suma de los numeros _______ es %d ",sumaimpar);
 getch();
 return 0;
 }
```

III. Describa lo que imprime cada programa.

1.

```
#include<stdio.h>
#include<conio.h>
main()
{
 float d,n;
 int c=0;
 clrscr();
 printf("Numero de digitos de una cifra\n\n");
 gotoxy(10,2);printf("Numeros de digitos de una cifra");
 gotoxy(10,4);printf("Dame una cifra: ");
 scanf("%f",& n);
 do
 {
 d=n/10;
 n=n/10;
```

```
 c++;
 }while(d>=1);
 gotoxy(10,6);printf("El numero de cifras es %d",c);
 getch();
 return 0;
}
```

2.

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

main()
{
 int num;
 clrscr();
 printf("mostrar el cuadrado de un numero leido del teclado.");
 printf("se detiene con -1\n\n");
do
 {
 printf(" \ndame un numero ");
 scanf("%d",&num);
 printf(" %d ",num*num);
 }while(num!=-1);
 getch();
 return 0;
}
```

3.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
main()
{
 int h,m,s;
 clrscr();
 printf(" simular un reloj digital\n\n");
 for (h=0;h<24;h++)
```

```
 for(m=0;m<=59;m++)
 for(s=1;s<=59;s++)
 {
  gotoxy(15,10);
  printf(" %d:%d:%d ",h,m,s);
  delay(1000);
  sound(15);
 }
 getch();
 nosound();
 return 0;
}
```

4.

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

main()
{

 int c,n,r,x=20;
 clrscr();
 printf("Convertir un numero de base 10 a base 2\n\n");
 printf("dame un numero ");
 scanf(" %d",&n);
 do
 {
 c=n/2;
 r=n%2;
 gotoxy(x,5);
 printf("%d",r);
 n=c;
 x--;
 }while (c>=1);
 getch();
 return 0;
}
```

5.

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

main()
{
 char y='b',x;
 while (y<='j')
 {
  x=y+3;
  printf ("%c\n",x);
  y=y+1;
 }
 getch();
 return 0;
}
```

6.

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

main()
{
 int y=1,x;
 while (y<=10)
 {
  x=y*y;
  printf ("%d\n",x);
  y=y+2;
 }
 getch();
 return 0;
}
```

7.

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

```
main()
{
 int num=10;
 while (num!=1)
 {
  if (num%2==0)
  {
  num=num/2;
  }
  else num=num*3+1;
  printf ("%d\n",num);
 }
 getch();
 return 0;
}
```

CAPÍTULO 6

Datos de tipo estructurado.

Arreglos

En este capítulo se revisan los datos estructurados (arreglos) que son un conjunto de elementos del mismo tipo de datos; además se estudiarán los arreglos que contienen cadenas de caracteres y sus operaciones más comunes.

6.1 Arreglo o array

Un arreglo, o *array*, es un conjunto de elementos del mismo tipo de datos almacenados en memoria continua.

Existen diferentes tipos de arreglos: unidimensionales, bidimensionales, tridimensionales, etcétera.

6.2 Vectores o arreglos unidimensionales

Un arreglo unidimensional, también llamado lista o vector, contiene un conjunto de variables del mismo tipo. Su declaración, como la de cualquier variable, requiere de un nombre y un tipo de dato; además se debe agregar el número de elementos que contendrá.

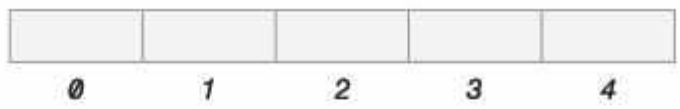
```
<Tipo dato> <identificador> <[número de elementos]>;
```

Por ejemplo, si se requiere una lista para anotar las calificaciones de cinco alumnos, el arreglo será de tipo entero, podría llamarse `LisCalif` (que es un nombre arbitrario como el de cualquier variable), y será de tamaño 5. Su declaración quedaría así:

```
int LisCalif[5];
```

El acceso a un elemento se lleva a cabo mediante el nombre del arreglo y un índice que señala una posición específica en el arreglo.

En C, la enumeración de elementos se inicia en 0, de tal forma que el índice en `LisCalif` tendrá valores de 0 a 4 y se podría ver así:



Para almacenar un valor en una celda en particular, es necesario escribir el nombre del arreglo y un índice que indique la posición en que se almacena el dato.

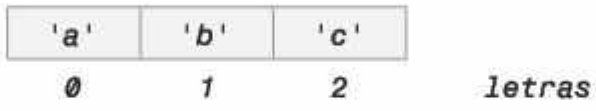
```
LisCalif[0]=10, LisCalif[1]=5, . . . LisCalif[4]=4;
```

Observe que el número de la celda es independiente del contenido. En el índice es posible utilizar variables, constantes o expresiones aritméticas, siempre que sean de tipo entero, para hacer referencia a la posición de un elemento como:

```
LisCalif[i+2];    LisCalif[i*10].
```

Recuerde que el resultado de la operación que hace referencia al índice debe ser siempre un número entero. En el siguiente ejemplo se muestra un arreglo de tipo carácter al que se le asignan los valores de a, b y c.

```
char letras[]={'a','b','c'}
```



En el siguiente ejemplo tenemos un arreglo con números de tipo real, y la forma en que se asignan los valores leídos del teclado:

```
float peso[4];
scanf("%f",&peso[1]);
```

Si se va a asignar un valor desde el teclado, se utiliza la instrucción `scanf` de forma similar a cualquier variable.

```
scanf("%d",&LisCalif[0]);
```

Para imprimir el contenido de una celda, se procede de manera similar a la de cualquier variable, teniendo cuidado de indicar el índice:

```
printf("  %d  ",LisCalif[0]);
```

Si se manipularan todos los elementos de un arreglo, es frecuente utilizar una estructura repetitiva; la más usual es el ciclo `for`, ya que los índices del arreglo aumentan de uno en uno. Por ejemplo, cuando desea imprimir el contenido del arreglo o cuando se suman todos los elementos, o también cuando se va a inicializar, como en el ejemplo siguiente:

```
for (i=0;i<=4;i++)
      LisCalif[i]=0;
```

En este caso se inicializó el arreglo con valores de 0; también es posible inicializar con valores distintos desde la declaración, como:

```
int LisCalif[5]={10, 5, 8, 9, 4};        o
Int LisCalif[]={10, 5, 8, 9, 4};
```

Si se omite el tamaño del arreglo, éste se definirá por el número de valores de inicialización; inclusive se pueden definir los elementos del arreglo ya declarado. Para imprimir todos los elementos de un arreglo usando el ciclo `for`:

```
for (i=0;i<=4;i++)
      printf(" %d",  LisCalif[i]);
```

En caso de que el índice `i` esté fuera del límite, el compilador de C no marcará error, pero puede causar un fallo en el programa.

EJEMPLO 6.1 Leer y almacenar siete estaturas y mostrarlas en forma tabular

```
#include<stdio.h>
#define TAMANIO 7
main()
{
  float estatura[TAMANIO];
  int i;
  for(i=0;i<=6;i++)
   {
      printf(" Dame la estatura %d ", i+1);
      scanf("%d",&estatura[i]);
   }
  printf("   num          estatura \n");
for(i=0;i<TAMANIO;i++)
      printf("  %d           %f \n", i+1,estatura[i]);
}
```

En este ejemplo se utiliza una directiva de preprocesador, que es una variable simbólica llamada `TAMANIO` de valor 7; en la función `main` se define un arreglo de siete elementos de números reales y el índice `i` de tipo entero. Mediante el ciclo `for` se solicita la estatura `i+1`: se escribió el índice así para imprimir el texto `"Dame la estatura 1"`, y se almacena en la posición `estatura[0]`. Todo esto se repite hasta que el valor de `i` llega a 6. Una vez almacenadas las estaturas se imprimen, con otro ciclo `for`, en forma de lista.

Ejercicios resueltos

EJERCICIO 6.1 Imprimir el contenido de las posiciones 0, 3 y 4 del arreglo vec

Descripción

Inicializar arreglo.

Imprimir los números de las posiciones indicadas.

Variables		
Nombre	*Tipo*	*Uso*
`vec`	`Vector real`	`Almacena los números.`

Codificación

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

main()
{
  int vec[5]={3,5,7,9,2};
  clrscr();
  printf("\n posicion 0 %d ",vec[0]);
  printf("\n posicion 3 %d ",vec[3]);
  printf("\n posicion 4 %d ",vec[4]);
  getch();
  return 0;
}
```

Ejecución

```
posicion 0 3
posicion 3 9
posicion 4 2
```

Explicación

Este ejemplo tiene los elementos asignados al inicio, así que sólo hay que mostrar el contenido de cada una de las posiciones indicadas.

EJERCICIO 6.2 Buscar un número en un arreglo e indicar en qué posición se encuentra

Descripción

Pedir el número a buscar.

Realizar la búsqueda en todo el arreglo.

Imprimir si el número se encontró y la posición.

(*continúa*)

(*continuación*)

Variables		
Nombre	*Tipo*	*Uso*
vec	Vector real	Almacena números para buscar entre ellos.
i	Entero	Variable de control para la estructura repetitiva.
n	Entero	Número a buscar.
b	Entero (booleano)	Almacena el valor de la bandera.
p	Entero	Indica la posición en que se encontró (si es que se encuentra).

Codificación

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

main()
{
  int vec[5]={3,5,7,9,2}, i, n, b=0,p;
  clrscr();
  printf("Teclea el valor a buscar  ");
  scanf("%i",&n);
  for(i=0;i<5;i++)
     if(vec[i]==n)
       {
         b=1;
         p=i;
       }
  if(b==1)
     printf("\nEl numero %i si se encuentra en la posicion
           %i",n,p);
  else
     printf("\nEl numero %i no se encuentra",n);
getch();
return 0;
}
```

Ejecución

```
Teclea el valor a buscar  9
El numero 9 si se encuentra en la posicion 3
```

Explicación

Solicita el número a buscar, que se amacena en la variable *n*. Se compara si el primer elemento del vector es *n*; si no es así, se revisan los siguientes elementos.

Cuando la condición es verdadera, *b* toma el valor de 1, *p* toma el valor de la posición. Al salir del ciclo se revisa si *b* contiene 1; si es cierto, se imprime que sí se encontró y en qué posición.

EJERCICIO 6.3 Imprimir el contenido de las posiciones pares de una lista y su suma

Descripción

Definir la lista.

Verificar la posición.

Si es par, imprimir y sumar.

Mostrar la suma.

Variables		
Nombre	*Tipo*	*Uso*
lista	*Entero*	*Almacena los valores del vector.*
suma	*Entero*	*Almacena la suma de los números contenidos en las posiciones pares.*
i	*Entero*	*Para recorrer el arreglo.*

Codificación

```
#include<stdio.h>
#include<conio.h>
#define C 10
main()
{
  int lista[C]={1,12,3,12,0,4,5,7,9,2};
  int suma=0, i=0;
  clrscr();
  printf("\n Elementos en posiciones pares\n ");\
  do{
```

(continúa)

(*continuación*)

```
    if(i%2==0)
      {
         printf("\n %d ",lista[i]);
         suma+=lista[i];
      }
    i++;
    }while(i<C);
  printf("\n la suma de esos numeros es  %d ",suma);
  getch();
  return 0;
}
```

Explicación

Una vez definida la lista, se inicializan en cero las variables que contendrán la suma y la que recorren los vectores `suma` e `i`, respectivamente. Se verifica si el valor de `i` es par, y de ser así se muestra y se suma. A continuación, independiente del valor del índice, éste se incrementa hasta recorrer todo el arreglo. Al final se muestra la suma de los números de la posición par.

Ejecución

```
Elementos en posiciones pares
1
0
3
5
9
la suma de esos numeros es 18
```

EJERCICIO 6.4 Leer elementos, imprimir, sumar y contar los elementos de una posición par; si el número contenido es impar indicar la posición

Descripción

Pedir la cantidad de números a almacenar la primera ocasión.

Evaluar si la posición es par.

Si es verdadero, evaluar si el número es impar.

Si se cumple lo anterior, imprimir el número y la posición.

Variables		
Nombre	Tipo	Uso
listaC	Entero	Almacena los números del arreglo.
suma	Entero	Suma los elementos impares.
c	Entero	Cuenta los números impares.
i	Entero	Recorre el arreglo.
n	Entero	Almacena la cantidad de números a leer.

Codificación

```
/*Leer elementos, imprimir, sumar y contar los elementos de
la posicion par, si el numero es impar e indicar la posicion.
*/
#include<stdio.h>
#include<conio.h>
#define C 10
int main()
{
int listaC[C];
int suma=0,c=0,i=0,n;
clrscr();
printf("\n Cuantos elementos se leeran (no mas de 10)\t ");
scanf("%d",&n);
for(i=0;i<n;i++)
   scanf("%d",&listaC[i]);
clrscr();
printf("\tposicion\tnumero\n");
i=0;
do{

   if(i%2==0)
     if(listaC[i]/2*2!=listaC[i])
     {
        printf("\n\t\t %d\t%d ",i,listaC[i]);
        suma+=listaC[i];
        c++;
      }
   i++;
   }while(i<n);
```

(continúa)

(*continuación*)

```
  printf("\n son %d y la suma de esos numeros es  %d ",c,suma);
  getch();
  return 0;
}
```

Explicación

Se pregunta la cantidad de números a almacenar en `n` en el arreglo. A continuación se leen y se almacenan en `listaC`. Se inicializa de nuevo `i` para recorrer el arreglo y verificar: primero si la posición es par, si es verdadero se evalúa si el número es impar (una forma de verificar si el número es impar es dividir entre dos y luego multiplicarlo por 2: si es igual al original es par, de lo contrario es impar). Si es verdadero se imprime la posición en que se encuentra y el número; además se hace la suma del mismo y se cuenta.

Finalmente muestra la cantidad de números impares en posición par y la suma de ellos.

Ejecución

```
Cuantos elementos se leeran (no mas de 10) 5
1
2
2
3
5

Posicion        numero
0                1
4                5

son 2 y la suma de esos numeros es  6
```

EJERCICIO 6.5 Realizar en un arreglo las siguientes operaciones: agregar un elemento, borrar un elemento, buscar un elemento de la lista

Descripción

Solicitar la cantidad de números a almacenar.

Leer números.

Solicitar el número a buscar.

Realizar la búsqueda.

Solicitar un número a borrar.

Borrar el número moviendo los elementos.

Pedir un número a insertar en la última posición.

Agregar el número.

Variables		
Nombre	*Tipo*	*Uso*
`lista[MAX]`	`Entero`	`Almacena la lista de números.`
`n`	`Entero`	`Almacena la cantidad de elementos almacenados.`
`buscar`	`Entero`	`Almacena el elemento a buscar.`
`borrar`	`Entero`	`Almacena el elemento a borrar.`
`i`	`Entero`	`Recorre el arreglo.`
`b`	`Entero`	`Bandera para la búsqueda del elemento.`

Codificación

```
/* En un arreglo buscar un elemento, borrar elemento, agregar. */
#include <stdio.h>
#include <stdlib.h>
#include <conio2.h>
#define MAX 100
#define p printf
#define s scanf

main()
{
  int lista[MAX];
  int n,buscar,borrar,i,b;
  do
  {
      p("Numero de elementos a almacenar  ");
      s("%d",&n);
  }while(n<0||n>MAX);
  for(i=0;i<n;i++)
```

(continúa)

(*continuación*)

```
  {
      p("elemento ");
      s("%d",&lista[i]);
  }
 clrscr();
  p("\n\n");
  for(i=0;i<n;i++)
     p(" %d ",lista[i]);
/* búsqueda */
  i=0;b=0;
  p("\n numero a buscar ");
  s("%d",&buscar);
  while(i<n&&b==0)
  {
      if(buscar==lista[i])
         b=1;
      else
         i++;
  }
  if(b==1)
      p("elemento encontrado en %d \n",i);
  else
      p("elemento no encontrado\n");
//borrar, se busca el elemento y se escribe el último elemento en su
lugar, disminuir n
/* búsqueda del número a borrar */
  i=0;b=0;
  p("\n numero a borrar ");
  s("%d",&buscar);
  while(i<n&&b==0)
  {
      if(buscar==lista[i])
         b=1;
      else
         i++;
  }
  if(b==1)
      {
          lista[i]=lista[n-1];
```

```
            n--;
        }
  p("\n\n");
  for(i=0;i<n;i++)
     p(" %d ",lista[i]);

/*insertar al final */
  p("\n numero a insertar ");
  s("%d",&buscar);
  lista[n]=buscar;
  n++;
  p("\n\n");
  for(i=0;i<n;i++)
     p(" %d ",lista[i]);

  system("PAUSE");
  return 0;
}
```

Explicación

Pregunta cuántos elementos serán almacenados en el arreglo y se almacena el valor en la variable `n`, se leen y almacenan en `lista`. A continuación se pregunta el número a buscar y se almacena en la variable `buscar`. Se realiza un ciclo para comparar con cada uno de los elementos del arreglo; en caso de ser iguales se cambia el valor de la bandera `b` a 1, y se sale del ciclo; al salir del ciclo se pregunta si la bandera `b` es 1. Se imprime `elemento encontrado` y si no, lo contrario.

Para borrar se pregunta el número y se busca; si se encuentra, se mueven los elementos, iniciando del último de la lista y hasta la posición `lista[i]`. La cantidad de elementos `n` se disminuye. Finalmente se solicita el número que se agrega y se almacena en `buscar`. Se incrementa el número de elementos `n` y se almacena el elemento en la posición `i`. Al terminar las operaciones se muestra el arreglo con los elementos.

Ejecución

```
Numero de elementos a almacenar   6
elemento 3
elemento 1
```

(continúa)

(*continuación*)

```
elemento 6
elemento 7
elemento 4
elemento 2

3 1 6 7 4 2

numero a buscar 7
elemento encontrado en 3

numero a borrar 6

3 1 7 4 2

numero a insertar 58

3 1 7 4 2 58
```

A continuación se muestra el ejercicio anterior con el uso de funciones con parámetros para cada operación.

```
/* En una matriz buscar un elemento, borrar elemento, agregar, usando
funciones y una variable global.
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio2.h>
#define MAX 100
#define p printf
#define s scanf

int n=-1;

void llenar(int l[])
{
 int i;
 do
 {
  p("numero de elementos a almacenar  ");
  s("%d",&n);

 }while(n<0||n>MAX);
```

```
 for(i=0;i<n;i++)
  {
    p("elemento ");
    s("%d",&l[i]);
  }
}

void mostrar(int l[])
{
  int i;
  for (i=0;i<n;i++)
    printf("  %d",l[i]);
}

void buscar(int b, int l[])
{
 int i=0,bandera=0;
 while(i<n&&bandera==0)
  {
      if(b==l[i])
         bandera=1;
      else
         i++;
  }
  if(bandera==1)
      p("elemento encontrado en %d \n",i);
  else
      p("elemento no encontrado\n");
}

void agregar(int num, int l[])
{
  l[n]=num;
  n++;
}

void borrar(int pos, int l[])
{
  int i;
```

```
  for (i=pos;i<n;i++)
   l[i]=l[i+1];
  n--;
}

int menu()
{
 int opc;
 clrscr();
 printf(" menu\n 1. llena\n 2. muestra\n 3. busca\n 4. agregar\n 5.
elimina \n 6. salir\n\t ");
 s("%d",&opc);
 return opc;
}

int main(int argc, char *argv[])
{
  int lista[MAX];
int buscado,b;
int opc,num,pos;
clrscr();
do{
 opc=menu();
 switch(opc)
 {
  case 1:// llenar el arreglo
          llenar(lista);
          break;
  case 2: mostrar(lista);
          break;
  case 3: //búsqueda
          p("\n numero a buscar ");
          s("%d",&buscado);
          buscar(buscado,lista);
          break;
  case 4: //agregar
          printf(" numero a agregar a la lista  ");
          s("%d",&num);
          agregar(num,lista);
```

```
            break;
    case 5://borrar
            p(" posicion a eliminar ");
            s("%d",&pos);
            borrar(pos,lista);
            break;
    case 6: printf("  adios ");
            break;
   }
   getch();
  }while(opc!=6);
}
```

En este ejemplo, en la función principal se encuentra el menú de las operaciones y de ahí se invoca cada una de ellas; esto se repite mientras no se escoja la opción número 6.

6.3 Matrices o arreglos bidimensionales

Los arreglos bidimensionales son también llamados tablas o matrices. Cada posición de un arreglo bidimensional tiene dos índices: el primero indica el número de renglón y el segundo el número de columna en que se encuentra el elemento. La forma convencional de declarar un arreglo bidimensional es:

<Tipo dato> <identificador> <[número de renglones]> <[número de columnas]>

```
int matriz1[3][2];
```

En un arreglo de números enteros de tres renglones y dos columnas (3 × 2) se podrían ver así los índices:

0,0	0,1
1,0	1,1
2,0	2,1

Cuando se alude a la matriz completa, como durante la asignación o la impresión de valores, en general se hace de manera similar a cuando se alude a un arreglo

unidimensional, cuidando solamente que se haga referencia a los índices de renglón y columna. La inicialización se realiza de la siguiente manera:

```
int matriz[3][2]={1,2,3,4,5,6};  o
int matriz[3][2]={
                                  {1,2}
                                  {3,4}
                                  {5,6}
                  };
```

El segundo caso muestra que en realidad un arreglo bidimensional es un arreglo de vectores. Para almacenar un valor desde el teclado:

```
scanf(" %d",&matriz[0][0]);
```

A la acción de manipular todos los elementos de una matriz se le puede llamar recorrido, y éste se lleva a cabo utilizando dos ciclos `for` anidados, ya que es necesario recorrer las columnas para cada una de las filas. Por ejemplo, para asignar a todos los elementos de la matriz un valor desde el teclado:

```
for(i=0;i<=2;i++)
 for(j=0;j<=1;j++)
       scanf("%d",&matriz[i][j]);
```

Para cada valor de `i` (que en este caso representa las filas) se recorrerán todos los valores de `j` (que son las columnas), ya que cada renglón tiene dos columnas en la matriz de ejemplo.

El elemento primordial de una matriz está constituido por sus índices, porque éstos determinan el orden en el que se debe ejecutar el ciclo `for`. Para imprimir un valor:

```
printf("%d",matriz[i][j]);
```

Para imprimir todos los valores del arreglo en el mismo orden en que se leyeron los datos, según el ejemplo anterior:

```
for(i=0;i<=2;i++)
 for(j=0;j<=1;j++)
       printf("%d",matriz[i][j]);
```

Si se invierten los índices se crea el mismo efecto de recorrer la `i`, que son las columnas, más rápidamente que los renglones con `j`:

```
for(j=0;j<=2;j++)
 for(i=0;i<=1;i++)
       printf("%d\t",matriz[j][i]);
```

Ahora, recorrer por columna, podría ser así:

```
for(i=0;i<=1;i++)
 for(j=0;j<=2;j++)
       printf("%d",matriz[j][i]);
```

o

```
       printf(" %d\t",matriz[i][j]);
```

Para imprimir todos los valores fila por fila:

```
for(i=0;i<=2;i++)
 for(j=0;j<=1;j++)
       printf("%d\t",matriz[i][j]);
```

y para imprimir los valores columna por columna puede ser así:

```
for(i=0;i<=2;i++)
 for(j=0;j<=1;j++)
       printf("%d",matriz[j][i]);
```

o también

```
for(j=0;j<=2;j++)
 for(i=0;i<=1;i++)
       printf("%d",&matriz[i][j]);
```

Ejercicios resueltos

EJERCICIO 6.6 Almacenar números entre 1 y 25, generados aleatoriamente, en una tabla de 3 × 2 renglones y columnas

Descripción

Indicar la función de números aleatorios.

Almacenar los números generados.

Mostrar la matriz.

(continúa)

(*continuación*)

Variables		
Nombre	*Tipo*	*Uso*
`tabla`	`Entero, real`	`Almacena los números.`
`i`	`Entero`	`Recorre filas.`
`j`	`Entero`	`Recorre columnas.`

Codificación

```
/* Tabla con números aleatorios entre 1 y 25 */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

main()
{
 int tabla[3][2], i,j;
 clrscr();
 randomize();
 for(i=0;i<3;i++)
   for(j=0;j<2;j++)
     tabla[i][j]=1+random(25);
 for(i=0;i<3;i++)
   {
     for(j=0;j<2;j++)
       printf("\t%d",tabla[i][j]);
     printf("\n");
   }
 getch();
 return 0;
}
```

Explicación

Se declara la matriz indicando el número de renglones y de columnas; a continuación con los dos ciclos se recorren las posiciones de la misma almacenando un número aleatorio. Como se explicó antes, `random` genera números entre 0 y 24 y se le suma uno para que sean entre 1 y 25. Al finalizar se muestra la matriz.

Ejecución

```
13
4
13
24
```

EJERCICIO 6.7 Realizar la suma de dos matrices, con una dimensión de hasta 4 × 4

Descripción

Definir el tamaño de la primera matriz.

Definir el tamaño de la segunda matriz.

Verificar que sean de la misma dimensión.

Llenar la matriz, primero una y luego la segunda.

Sumar las dos matrices.

Mostrar el resultado de la suma.

Variables		
Nombre	*Tipo*	*Uso*
`posX, posY, pX_1, pX_2`	`Entero`	`Posición del cursor para imprimir.`
`band_1`	`Entero`	`Bandera.`
`m1, m2, m3`	`Entero`	`Matrices a sumar y a almacenar.`
`I, j`	`Entero`	`Índices para renglón y columna.`
`col_1,ren_1,col_2,ren_2`	`Entero`	`Dimensión de las matrices.`

Constantes	
Nombre	*Uso*
`Columna`	`Número de columnas.`
`Renglón`	`Número de renglones.`

Codificación

```
/* Suma de matrices */
#include<stdio.h>
```

(continúa)

(*continuación*)

```
#include<conio.h>
#define columna 4
#define renglon 4

main()
{
  int m1[renglon][columna],m2[renglon][columna],m3[renglon][columna];
  int i,j,col_1,ren_1,col_2,ren_2,posX,posY,pX_1,pX_2;
  int band_1;
  clrscr();
  do
  {
    printf("\n\t Matriz 1, dame el numero de renglones y columnas ");
    scanf("%d %d",&ren_1,&col_1);
    printf("\n\t Matriz 2, dame el numero de renglones y columnas ");
    scanf("%d %d",&ren_2,&col_2);
    if(ren_1==ren_2&&col_1==col_2)
              band_1=0;
   else
   {
      band_1=1;
      printf("numeros de datos invalidos");
   }
 }while(band_1);
 clrscr();
 pX_1=0;
 textcolor(5);
 gotoxy(6,7);
/* llenar la primera matriz */
 cprintf("matriz 1");
 posX=5;posY=9;
 for(i=0;i<ren_1;i++)
 {
  for(j=0;j<col_1;j++)
   {
     gotoxy(posX,posY);
     printf(" ");
     scanf("%d",&m1[i][j]);
     posX+=2;
```

```
    }
    posX=5;
    posY+=2;
  }
 pX_2=35;
 gotoxy(40,7);
/* llenar la segunda matriz */
cprintf("matriz 2");
posX=40;posY=9;
for(i=0;i<ren_2;i++)
  {
  for(j=0;j<col_2;j++)
     {
     gotoxy(posX,posY);
     printf(" ");
     scanf("%d",&m2[i][j]);
     posX+=2;
    }
posX=40;
posY+=2;
}
/* sumar  */
for(i=0;i<ren_1;i++)
   for(j=0;j<col_1;j++)
      m3[i][j]=m1[i][j]+m2[i][j];
posX=30;posY=30;
gotoxy(28,28);
textcolor(5);
/* mostrar suma  */
cprintf(" Resultado");
for(i=0;i<ren_1;i++)
{
 for(j=0;j<col_1;j++)
  {
    gotoxy(posX,posY);
    printf("%d\t",m3[i][j]);
    posX+=3;
  }
 posX=30;
```

(continúa)

(*continuación*)

```
    posY+=3;
   }
   getch();
   return 0;
}
```

Explicación

Se solicita el número de renglones y columnas de la primera matriz; a continuación se solicita para la segunda, se verifica que la dimensión sea igual y en caso contrario pregunta de nuevo las dimensiones. Ya que es válida la dimensión se solicitan los datos; para cada una de las matrices, el cursor se coloca en la posición que muestra la dimensión de la matriz.

A continuación realiza la suma y muestra la matriz resultante conservando la dimensión.

Ejecución

```
Matriz 1, dame el numero de renglones y columnas 2 3
Matriz 2, dame el numero de renglones y columnas 2 3
matriz 1                                    matriz 2
1  2  3                                     1  1  1
4  5  6                                     1  1  1
                    Resultado
                    2  3  4
                    5  6  7
```

Suma de matrices usando funciones

```
/* Suma de matrices, utilizando funciones */
#include<stdio.h>
#include<conio.h>
#define columna 4
#define renglon 4

void escribe(int m1[renglon][columna],int ren_1,int col_1)
{
  int i,j,posX,posY;
  posX=30;posY=30;
```

```
  gotoxy(28,28);
  textcolor(5);
  cprintf(" Resultado");
  for(i=0;i<ren_1;i++)
  {
   for(j=0;j<col_1;j++)
    {
      gotoxy(posX,posY);
      printf("%d\t",m1[i][j]);
      posX+=3;
    }
   posX=30;
   posY+=3;
  }
}

void llenar(int m[renglon][columna],int ren,int col,int pX)
{
  int i,j,posX,posY;
  posX=5+pX;posY=9;
  for(i=0;i<ren;i++)
  {
   for(j=0;j<col;j++)
    {
      gotoxy(posX,posY);
      printf(" ");
      scanf("%d",&m[i][j]);
      posX+=2;
    }
    posX=5+pX;
    posY+=2;
 }
}

void sumar(int m1[renglon][columna],int m2[renglon][columna],int
m3[renglon][columna],int ren_1,int col_1)
{
  int i,j;
  for(i=0;i<ren_1;i++)
```

```
        for(j=0;j<col_1;j++)
          m3[i][j]=m1[i][j]+m2[i][j];
  escribe(m3,ren_1,col_1);
}

main()
{
  char opr;
  int m1[renglon][columna],m2[renglon][columna],m3[renglon][columna];
  int i,j,col_1,ren_1,col_2,ren_2,posX,posY,pX_1,pX_2;
  int band_1;
  clrscr();
  do
    {
    printf("\n\t Matriz 1, dame el numero de renglones y columnas ");
    scanf("%d %d",&ren_1,&col_1);
   printf("\n\t Matriz  2, dame el numero de renglones columnas ");
   scanf("%d %d",&ren_2,&col_2);
   if(ren_1==ren_2&&col_1==col_2)
     band_1=0;
   else
    {
    band_1=1;
    printf("numeros de datos invalidos");
    }
  }while(band_1);
  clrscr();
  pX_1=0;
  textcolor(5);
  gotoxy(6,7);
  cprintf("matriz 1");
  llenar(m1,ren_1,col_1,pX_1);
  pX_2=35;
  gotoxy(40,7);
  cprintf("matriz 2");
  llenar(m2,ren_2,col_2,pX_2);
  sumar(m1,m2,m3,ren_1,col_1);
  getch();
  return 0;
}
```

6.4 Arreglo de caracteres y cadena de caracteres

Generalmente se dice que es arreglo de caracteres cuando lo que se almacenó son caracteres y no existe el carácter nulo al final (`/0`). Cuando el arreglo de caracteres termina con el carácter nulo se llama cadena de caracteres. Ambos funcionan de forma similar a los arreglos numéricos, partiendo de la base de que cada carácter ocupa normalmente un byte.

'H'	'O'	'L'	'A'
0	1	2	3

Éste es un arreglo de caracteres como se vería almacenado en memoria. La declaración de este arreglo es:

```
char saludo[4];
```

y la asignación será:

```
saludo [0] = 'H';
saludo [1] = 'O';
saludo [2] = 'L';
saludo [3] = 'A';
```

Desde el teclado se define así:

```
scanf("%c",&saludo[0]);
```

Para imprimir un carácter de la cadena se puede utilizar `printf`; lo que muestra en pantalla el carácter contenido en la celda con posición 0.

```
printf("%c",saludo[0]);
```

Otra forma de leer y mostrar caracteres es mediante las funciones que a continuación se describen; el ejemplo 6.3 muestra una aplicación de las mismas.

Función `getchar()` Permite leer una cadena carácter por carácter.

Función `putchar()` Esta función es opuesta a `getchar(); putchar()` escribe en la salida un carácter.

EJEMPLO 6.2 Contar el número de ocasiones que aparece la letra 'a' en una línea

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

(*continúa*)

(*continuación*)

```
main()
{
   int CuentaCar=0;
   char Cadena;
   while (Cadena!=EOF)
        {
          Cadena=getchar();
          if (Cadena=='a')
             ++CuentaCar;
        }
   printf("\n Aparecieron %d  veces la letra 'a' ",CuentaCar);
   getch();
   return 0;
}
```

Con `getchar` se leen los caracteres; para salir de la lectura de caracteres se presionan las teclas Ctrl+Z, que es el fin de línea. Una vez leído un carácter se compara con la letra `a` y si es igual se suma con `CuentaCar`. Al final se muestra el resultado.

EJEMPLO 6.3 Imprimir en pantalla, en mayúsculas, un nombre que fue leído en minúsculas

```
#include<stdio.h>
#include<conio.h>
#include(ctype.h)
main()
{
   char Cadena;
   printf("\n  Escriba su nombre:  ");
   while (Cadena!=EOF)
    {      Cadena=getchar();
           putchar(toupper(Cadena));
    }
   getch();
   return 0;
}
```

Se declara una biblioteca nueva denominada `ctype` que permite analizar caracteres y hacer conversiones. Contiene, entre otras, la función `toupper`, que convierte un carácter escrito en minúscula en uno escrito en mayúscula.

El programa leerá un nombre, carácter por carácter hasta que oprima Ctrl+Z. Ya que se usó la función `getchar`, convierte a mayúscula cada carácter conforme se lee y al final lo imprime.

Funciones *getch() y getche()* Estas funciones no se incluyen en el estándar de C, pero se incorporan en la mayoría de los compiladores.

Las dos funciones permiten leer un carácter del teclado: `getch()` no lo visualiza en pantalla, mientras que `getche()` lo lee y lo imprime en pantalla. Estas funciones se encuentran en la biblioteca `conio.h`.

Cadena de caracteres

Las cadenas de caracteres suelen tener un carácter al final que indica la terminación de la cadena (se conoce como carácter nulo, y en código ASCII es 0). En C, la longitud de un arreglo de cadena de carácter se define por el tamaño de la cadena más uno, para almacenar el carácter nulo. Para leer una cadena se puede utilizar `scanf`, anotando sólo el nombre del arreglo, y para mostrarlo se utiliza `printf`.

```
char A[5];
scanf("%s",&A);
```

H	O	L	A	'\0'
0	1	2	3	4

Para imprimir la cadena de caracteres completa se requiere sólo el nombre del campo y se imprimirá desde el inicio hasta encontrar un fin de línea (`\0`) o un retorno de carro (`\n`).

```
printf("%s",A);
```

Para el caso en que se desee leer una cadena de caracteres como **Gabriela Márquez**, si se utiliza la función `scanf` se almacenará sólo la palabra Gabriela, ya que esta instrucción deja de leer al encontrar un espacio en blanco. Para esos casos es preferible utilizar la función `gets()`.

La **función *gets()*** permite leer una cadena, incluyendo espacios en blanco, terminando con el carácter de fin de línea (enter). La **función *puts()*** imprime una cadena de caracteres, incluyendo el carácter de fin de línea, así que el apuntador señala el inicio del siguiente renglón.

EJEMPLO 6.4 Almacenar un nombre en el arreglo denominado `Mi Nombre`

```
#include<stdio.h>
#include<conio.h>
main()
{
   char MiNombre[50];
   printf("\n  Escriba su nombre:  ");
   gets(MiNombre);
   printf("\n\n  Hola, mi nombre es ");
   puts (MiNombre);
   getch();
   return 0;
}
```

Funciones para manejo de cadenas

El C estándar contiene la biblioteca *string.h*, que permite utilizar las funciones de manipulación de cadenas de caracteres más usuales, como calcular la longitud de una cadena, comparar alfabéticamente dos cadenas, concatenar cadenas, etcétera.

Función *strcpy* Copia la cadena apuntada por *cadena2* a *cadena1*, incluyendo el carácter nulo. Su sintaxis es:

```
char strcpy(char cadena1, char cadena2);
```

La función toma el contenido de *cadena2* y lo almacena en *cadena1*. *Cadena1* debe ser de tamaño igual o mayor a *cadena2*, ya que almacena completamente hasta el carácter de terminación de *cadena2*.

Función *strncpy* Copia una cantidad determinada de caracteres de una cadena y los almacena en un arreglo. Su sintaxis es:

```
char strncpy(char cadena1, char cadena2, int n);
```

Toma *n* caracteres escritos en *cadena2* y los almacena en *cadena1*. Puede ser que no se copia el carácter de terminación de la *cadena2*.

Función *strcat* Toma una cadena y la almacena en un arreglo, al final de lo que el arreglo tenga escrito. Su sintaxis es:

```
char strcat(char cadena1, char cadena2)
```

Toma el contenido de *cadena2* y lo deposita en *cadena1*, a partir del carácter de fin de la *cadena1*, sobrescribiendo ese carácter.

Función *strncat* Toma cierta cantidad de caracteres de una cadena y la almacena en un arreglo, al final de lo que el arreglo tenga escrito. Su sintaxis es:

```
char strncat(char cadena1, char cadena2, int n)
```

Toma el número de caracteres definido en *n*, de *cadena2* y lo deposita en la *cadena1*, a partir del carácter de fin de *cadena1*, sobrescribiendo el carácter de fin de *cadena1*.

Función *strcmp* Compara dos cadenas de caracteres alfabéticamente, devuelve 0 si son iguales, un número negativo si la primera es menor que la segunda o un número positivo si la primera es mayor a la segunda. Su sintaxis es:

```
int strcmp( const char cadena1, const char cadena2)
```

Toma el contenido de *cadena1* y lo compara con el contenido de *cadena2*.

Función *strlen()* Devuelve el número de caracteres de una cadena, hasta el que antecede al carácter nulo, de terminación de cadena.

Ejercicios resueltos

EJERCICIO 6.8 Leer un arreglo de caracteres con la instrucción scanf y mostrar lo capturado

Descripción

Leer cada uno de los caracteres.

Verificar si se lee un fin de cadena de carácter.

Mostrar la cadena de caracteres leída.

Variables		
Nombre	*Tipo*	*Uso*
cad	*Carácter*	*Almacena la cadena leída.*
car	*Carácter*	*Almacena un carácter.*
i	*Entero*	*Recorre el arreglo.*

(*continúa*)

(*continuación*)

Codificación

```
/* Captura de una cadena de caracteres con scanf */

#include<stdio.h>
#include<conio.h>

void main()
  {

  char cad[20],car;
  int i=0;
  clrscr();
  printf("\nIntroduzca una cadena de caracteres (no mas de 18) \n");
  do
 {
 scanf("%c",&car);
 if (car!='\n')
  {
  cad[i]=car;
  i++;
  }

}while(car!='\n');

cad[i]='\0'; /* Se añade el fin de cadena */
printf("\n La cadena capturada es: %s", cad);
getch();
 }
```

Explicación

Se lee un carácter, se almacena en el vector *cad*, en la posición 0 se incrementa el índice, se evalúa la condición y si el carácter leído es diferente de fin de nueva línea, se repite el ciclo. Al finalizar el ciclo se coloca el fin de cadena y a continuación se muestra la cadena de caracteres leída.

Ejecución

```
Introduzca una cadena de caracteres (no mas de 18)
Gaby
La cadena capturada es Gaby
```

EJERCICIO 6.9 Leer un arreglo de caracteres y mostrar la longitud de la cadena leída, carácter a carácter, con `strlen`, y con `sizeof` finalmente mostrar la posición en memoria del arreglo

Descripción

Leer la cadena hasta que se oprima fin de línea.

Mostrar cada uno de los resultados.

Variables		
Nombre	*Tipo*	*Uso*
`cad`	`Carácter`	`Almacena la cadena leída.`
`car`	`Carácter`	`Almacena un carácter.`
`i`	`Entero`	`Almacena la longitud del arreglo.`
`j`	`Entero`	`Recorre el arreglo.`

Codificación

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 20

main()
  {
  char cad[MAX],car;
  int i=0,j;
  clrscr();
  printf("\nIntroduzca una cadena de caracteres (no mas de 18) \n");
  do
           {
           scanf("%c",&car);
           if (car!='\n')
             {
             cad[i]=car;
             i++;
             }
           }while(car!='\n');
        cad[i]='\0'; //Colocando fin de cadena
```

(continúa)

(*continuación*)

```
  clrscr();
 /* imprimir ASCII */
  printf("\n La cadena capturada es: %s", cad);
  printf("\n  ASCII correspondiente\n");
  for(j=0;j<i;j++)
     printf("\t %d",cad[j]);
  printf("\n La longitud de la cadena, contada con i es: %d", i);
  printf("\n La longitud de la cadena medida con strlen es: %d",
        strlen(cad));
  printf("\n La longitud del arreglo cad es: %d", sizeof(cad));
  printf("\n La direccion donde se encuentra la cadena capturada es:
        %d",&cad);
  getch();
}
```

Explicación

Se lee la cadena1 carácter a carácter, se agrega el carácter nulo y se muestra la cadena de caracteres según el número ASCII que le corresponda a cada letra. A continuación se muestra la longitud según el contador (que incluye el espacio en blanco), la longitud medida con la instrucción `strlen` (número de caracteres de la cadena) y el tamaño del arreglo con la instrucción `sizeof` (número de bytes de todo el arreglo). Finalmente se muestra la dirección de memoria donde se inició el almacenamiento de la cadena.

Ejecución

```
Introduzca una cadena de caracteres (no mas de 18)
gaby marquez
La cadena capturada es
gaby marquez
103    97    98    121    32    109    97    114    113    117    101
122
La longitud de la cadena, contada con i es: 12
n La longitud de la cadena medida con strlen es: 12
La longitud del arreglo cad es 20
La direccion de la cadena capturada es: 2293584
```

EJERCICIO 6.10 Leer un nombre y contar el número de ocasiones que aparece la letra seleccionada

Descripción

Leer nombre.

Leer letra a contar.

Mostrar resultado.

Variables		
Nombre	*Tipo*	*Uso*
`total`	`Entero`	`Cuenta el número de incidencias.`
`i`	`Entero`	`Recorre el arreglo de caracteres.`
`nombre`	`Arreglo caracteres`	`Almacena el nombre.`

Codificación

```
/* Muestra menú para captura de nombre y cuenta letras */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
  {
  int total,i;
  char nombre[40],letra;
  clrscr();
  printf("\nDame el nombre\n");
  nombre[0]='\0';
  gets(nombre);
  fflush(stdin);
  printf("\nCual letra se va a contar\n");
  scanf("%c",&letra);
  total=0;
  for(i=0;i<strlen(nombre);i++)
     {
     if (letra==nombre[i])
       total++;
```

(continúa)

(*continuación*)

```
    }
  printf("\nLa letra - %c - se encontro %i veces en el nombre:
        %s\n",letra,total,nombre);
  getch();
  return 0;
}
```

Explicación

Se solicita un nombre y a continuación se limpia la entrada estándar; la cadena es almacenada en la variable `nombre`, después se lee en `letra` el carácter a contar; la variable `total` llevará la suma de las ocasiones que se encuentre. En el cuerpo del ciclo se encuentra la instrucción `strlen`, que devuelve el número de caracteres leídos. Al final se muestra el carácter y el número de ocurrencias, así como el nombre. Es importante mencionar que se consideran diferentes las letras mayúsculas y las minúsculas.

Ejecución

```
Dame el nombre
gabriela marquez
Cual letra se va a contar
a
La letra - a - se encontro 3 veces en el nombre: gabriela marquez
```

EJERCICIO 6.11 Distinguir entre un número y una letra; si es letra, además distinguir si es mayúscula o minúscula

Descripción

Leer el carácter.

Evaluar si es número.

Si no es número, evaluar si es letra.

Si es letra, evaluar si es mayúscula.

Variables		
Nombre	*Tipo*	*Uso*
`car`	`Carácter`	`Almacena el carácter a evaluar.`

Codificación

```
/*Identifica si un carácter es una letra del alfabeto o un dígito */

#include <stdio.h>
#include <ctype.h>
#include <conio.h>

main()
{
  char car;
  printf("\nCapture un caracter\n");
  scanf("%c",&car);
  if (isdigit(car))
    printf("\nEl caracter capturado es un digito\n");
  else {
       if (isalpha(car))
          printf("\nEl caracter capturado es una letra del alfabeto");
       if (isupper(car))
          printf(" y es mayuscula\n");
       else
          printf(" y es minuscula\n");
          }
  getch();
  return 0;
}
```

Explicación

Se captura el carácter y se compara con la instrucción `isdigit`. Si es un dígito entre cero y nueve se muestra el texto `"El caracter capturado es un digito"`; de lo contrario, se compara de nuevo para verificar si es letra, y de serlo escribe que es una letra. A continuación compara si es mayúscula y lo indica; de lo contrario escribe que es minúscula.

Ejecución

```
Capture un caracter
S
El caracter capturado es una letra del alfabeto y es mayuscula
```

Resumen

Se llama arreglo al conjunto de elementos o variables del mismo tipo de dato. Los arreglos pueden ser de una o de varias dimensiones.

El arreglo unidimensional se conoce también como vector o lista, y se define mediante un tipo de dato, un identificador y la cantidad de elementos entre corchetes.

Para tener acceso a un elemento se utiliza un índice, que indica la posición del arreglo a consultar; también se usa para asignar valores.

El arreglo bidimensional es llamado matriz o tabla; en cada posición tiene dos índices: uno para indicar la fila y otro para la columna.

Es importante escribir siempre el índice a continuación del identificador para tener acceso a una posición del arreglo, ya sea para leer un valor o para asignarlo.

El arreglo puede almacenar caracteres. La diferencia entre formar un arreglo de caracteres y formar una cadena de caracteres, consiste en que en el segundo caso se agrega el carácter nulo `\0` al final.

En la biblioteca `string.h` se encuentran funciones para la manipulación de cadenas de caracteres, como unir dos cadenas de caracteres, comparar dos cadenas o copiar una cadena sobre otra.

Evaluación

I. Conteste las siguientes preguntas.

1. ¿Qué es un arreglo?

2. El índice se utiliza para indicar la _____ en un arreglo.

3. ¿Con qué número inicia la numeración de un arreglo?

4. ¿Puede un arreglo almacenar datos de tipo básico?

5. ¿Puede un arreglo almacenar datos de tipo estructurado?

6. ¿Es posible inicializar el arreglo cuando se declara?

7. ¿Cómo se escribe para asignar un valor en un arreglo?

8. ¿Es posible que el arreglo tenga una, dos o más dimensiones?
9. ¿Cuál es la diferencia entre arreglo de caracteres y cadena de caracteres?
10. ¿Es posible comparar dos cadenas de caracteres y decir cuál tiene mayor longitud?
11. ¿Cómo se ordenan los caracteres?

II. Muestre el resultado del siguiente ejercicio.

Escriba un programa que almacene una lista de cinco nombres, diga cuál nombre es el mayor y cuántos caracteres tiene.

Ejercicios propuestos

I. Codifique los siguientes programas.

1. Leer e imprimir el nombre completo de un alumno, con su carrera y código.
2. Hacer un programa que permita leer cierta cantidad de nombres de asignaturas que debe llevar un alumno.
3. Definir un programa que permita leer un texto de 20 líneas.
4. Contar el número de vocales que aparecen en una cadena de caracteres.
5. Escribir un programa que permita leer la longitud de una cadena de caracteres.
6. Eliminar los duplicados en un arreglo ordenado (por ejemplo, 1 2 2 3 4 5 5 5 6 7 8 9 9 10 11).
7. Almacenar en tres arreglos paralelos información de proveedores: en uno el nombre del proveedor; en otro, la ciudad en que trabaja, y en el tercero el número de artículos que vendió. Realizar las siguientes operaciones: mostrar un proveedor por artículo, borrar un proveedor, insertar un nuevo proveedor.
8. Leer los elementos de un arreglo de 3 × 6 y mostrar los elementos de cada renglón en una columna.

9. Leer una fecha, pedir un número de días que se suma a esa fecha, mostrar la fecha final.
10. Leer una frase y escribir cada palabra en un renglón.
11. Ordenar cinco cadenas de texto.

II. Complete las líneas en blanco.

```
/*Unir dos arreglos ordenados */
#include <stdio.h>
#define MAX 100
#define p printf
#define s scanf
int main(int argc, char *argv[])
{
  int A[MAX],B[MAX],unidos[MAX],lA,lB,lC,i,j,k;
  /* llenar los arreglos */
  p("\n cuantos elementos en el arreglo A ");
s("%d",&lA);
for(i=0;i<lA;i++)
    s("%d",&A[i]);
p("\n cuantos elementos en el arreglo B ");
s(_____);
for(i=0;i<lB;i++)
    s("%d",&B[i]);
/* comprobar longitud de arreglos */
lC=lA+lB;
if(_____)
{
   p(" demasiados elementos");
   exit (0);
}
/* insertar */
i=0;j=0;k=0;
while(i<lA&&j<lB)
{
    if(A[i]<B[j])
       {
         unidos[___]=A[___];
```

```
            i++;
          }
       else
          { unidos[k]=B[j];
            j++;
          }
       k++;
}
if(i==lA)
   while(j<lB)
   {
      unidos[k]=B[j];
      j++;
      k++;
   }
else
   while(i<lA)
   {
      unidos[k]=A[i];
      i++;
      k++;
   }
  /* mostrar resultado */
  for(i=0;i<lC;i++)
        p("\n\n %d ",unidos[___]);
  system("PAUSE");
  return 0;
}
```

III. Describa lo que imprime el siguiente programa.

1.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void imprimeNcaracteres(int cant,char titulo[])
  {
  int i;
  printf("\n");
```

```
  for(i=0;i<cant;i++)
    printf("%c\t",titulo[i]);
  printf("\n");
  }

int main()
{
  int n;
  char frase[20];
  printf("\nEscriba el titulo de un libro\n");
  gets(frase);
  printf("\nEscriba la cantidad de caracteres a imprimir de esa
frase\n");
  scanf("%i",&n);
  imprimeNcaracteres(n,frase);
  getch();
  return 0;
}
```

CAPÍTULO 7

Datos de tipo estructurado

En este capítulo se hablará del tipo de datos estructurado y de la definición y manejo de apuntadores.

7.1 Estructura

La estructura o registro es un tipo de datos estructurado que agrupa una o más variables, ya sea de un solo tipo o de diferentes tipos de datos, y además éstos pueden ser a su vez datos simples o datos estructurados. Una estructura se declara de la siguiente manera:

```
struct identificador_estructura
        {
        Tipo_dato identificador_campo1;
        Tipo_dato identificador_campo2;
```

```
.
.
.
};
```

Donde

identificador_estructura es el nombre de la estructura; en el cuerpo del mismo (dentro de las llaves) se encuentran los campos que contendrán la estructura, cada uno con su respectivo tipo de dato. Se muestra un ejemplo a continuación:

```
struct agenda
{
   char nombre[30];
   int edad;
};
```

Para declarar variables de tipo agenda sería:

```
struct agenda alumno, profesor;
int a;
char b,c;
```

Se declararon dos variables, *alumno* y *profesor*, siguiendo las mismas reglas que se han empleado para declarar una variable simple. También se pueden declarar a continuación de la definición:

```
struct agenda
{
   char nombre[30];
   int edad;
}alumno,profesor;
```

En este ejemplo, la estructura agenda tiene dos campos llamados también miembros: *nombre* es una cadena de caracteres y *edad*, un entero.

Para asignar valores en la estructura se puede proceder de la siguiente manera:

```
alumno.nombre= "Juan Perez";
alumno.edad=15;
```

En donde se hace referencia al identificador de la variable, seguido del operador punto (.) que es el operador de miembros de estructura. O bien, si se desea ingresar los datos desde el teclado:

```
scanf("%s",&alumno.nombre);
scanf("%i",&alumno.edad);
```

para imprimir los datos leídos:

```
printf("%s",alumno.nombre);
printf("%i",alumno.edad);
```

A continuación se muestra un ejemplo que maneja esta estructura, y se declaran dos variables.

EJEMPLO 7.1 Leer y mostrar los datos de un alumno y un profesor; el ejemplo utiliza la definición de estructura como una variable local

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

main()
{
 struct agenda
  {
  char nombre[30];
  int edad;
  }alumno,profesor;
 clrscr();

/*inicializar */
 alumno.nombre[0]='0';
 alumno.edad=0;
 profesor.nombre[0]='0';
 profesor.edad=0;

/*asignación */
 printf(" dame el nombre del alumno ");
 scanf("%s",&alumno.nombre[0]);
 printf(" dime la edad ");
 scanf("%d",&alumno.edad);
 printf(" dame el nombre del profesor ");
 scanf("%s",&profesor.nombre);
 printf(" dime la edad ");
```

(continúa)

(*continuación*)

```
  scanf("%d",&profesor.edad);

/*imprimir */
  clrscr();
  printf("\nnombre\t\tedad\n");
  printf(" %s\t%d\n",alumno.nombre,alumno.edad);
  printf(" %s\t%d\n",profesor.nombre,profesor.edad);
  getch();
  return 0;
}
```

En el ejemplo se declaran dos variables, `profesor` y `alumno`, que tienen los mismos campos: uno que es el `nombre` como cadena de caracteres y otro es la `edad` de tipo entero; se inicializan los datos, se solicitan y se muestran.

EJEMPLO 7.2 Mostrar el nombre y la estatura de un alumno (manejo de una estructura como variable global)

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

struct agenda
   {
   char nombre[30];
   float estatura;
   };

main()
{

struct agenda  alumno;
clrscr();

/* inicializar */
  alumno.nombre[0]='0';
  alumno.estatura=0;

/* asignación */
```

```
  printf(" dame el nombre del alumno ");
  scanf("%s",&alumno.nombre[0]);
  printf(" dime la estatura ");
  scanf("%f",&alumno.estatura);

/* imprimir */
  clrscr();
  printf("\nnombre\t\t\n");
  printf(" %s\t%.2f\n",alumno.nombre,alumno.estatura);
  getch();
  return 0;
}
```

En este ejemplo se declara la estructura fuera de la función; el manejo es similar al ejemplo anterior, aunque en este caso el primer campo de la estructura, `nombre>` es una cadena de caracteres y `estatura` es un valor de tipo real. Se inicializa, se asignan valores y se muestran.

EJEMPLO 7.3 Solicitar y mostrar el nombre y tres calificaciones para cada alumno; pueden incluirse hasta 10 alumnos (arreglo de estructuras)

```
#include <stdio.h>
#include <conio.h>
#define MAX 10

main()
{
  struct
   {
   char nombre[30];
   int  calificacion[3];
   }alumno[MAX];

   clrscr();
  int i,j,n;

/* inicializar */
```

(continúa)

(*continuación*)

```
  for(i=0;i<MAX;i++)
     { alumno[i].nombre[0]='\0';
       for (j=0;j<3;j++)
              alumno[i].calificacion[j]=0;
     }

/* agregar elementos */
  i=0;
  printf(" \n cuantos elementos se insertaran \t");
  scanf("%d",&n);
  while(i<n)
     {
     printf(" \ndame el nombre %d ",i+1);
     scanf("%s",&alumno[i].nombre[0]);
     for (j=0;j<3;j++)
        {
        printf(" dame la  calificacion %d ",j+1);
        scanf("%d",&alumno[i].calificacion[j]);
        }
     i++;
     }

/* imprimir */
  printf("\n\tnombre\t\tcalificacion 1\tcalificacion
        2\tcalificacion 3\n\n");
  for(i=0;i<n;i++)
     {
     printf("\t %s ",alumno[i].nombre);
     for(j=0;j<3;j++)
       {
          printf("\t%10d",alumno[i].calificacion[j]);
       }
     printf("\n");
     }
  getch();
  return 0;
}
```

La estructura consta de dos campos: *nombre* de tipo cadena de caracteres y un arreglo de tres enteros para almacenar en *calificaciones*, las calificaciones respectivas; todo será almacenado en un arreglo *alumnos* de 10 elementos. Cada posición del arreglo alumnos almacena una estructura.

El acceso a la variable se lleva a cabo colocando el nombre de la misma, la posición y a continuación el nombre del campo de la estructura a usar.

En el ejemplo se omitió el nombre de la estructura, que para estos ejemplos es opcional; en otras palabras, no afecta si se omite cuando la declaración de las variables se realiza inmediatamente después.

Estructuras anidadas

El campo de una estructura puede, a su vez, ser otra estructura que ya haya sido definida.

```
struct nombre_com
{
   char nombre[30];
   char apellidopat[15];
   char apellidomat[15];
 };

struct agenda
   {
   struct nombre_com nombrec;
   int  calificacion[3];
   float estatura;
   }alumno[MAX];
```

La primera estructura *nombre_com* consta de tres elementos tipo cadena de caracteres y está contenida en la estructura *agenda*, que además de *nombre_com*, tiene los campos *calificaciones*, que es un arreglo de enteros y *estatura*, de tipo real.

Para asignar valores en la estructura anidada se procede de la siguiente manera:

```
alumno[0].nombrec.nombre = "Juan";
alumno[0].nombrec.apellidopat = "Perez";
```

```
alumno[0].nombrec.apellidomat = "Garcia";
alumno[0].edad=15;
alumno[0].estatura=1.85;
```

o bien, si se desea ingresar los datos por teclado:

```
scanf("%s",&alumno[0].nombrec.nombre);
scanf("%s",&alumno[0].nombrec.apellidopat);
scanf("%s",&alumno[0].nombrec.apellidomat);
scanf("%i",&alumno[0].edad);
scanf("%f",&alumno[0].estatura);
```

para imprimir los datos leídos:

```
printf("%s",alumno[0].nombre_c.nombre);
printf("%s",alumno[0].nombre_c.apellidopat);
printf("%s",alumno[0].nombre_c.apellidomat);
printf("%i",alumno[0].edad);
printf("%.2f",alumno[0].estatura);
```

EJEMPLO 7.4 Mostrar nombre, tres calificaciones y estatura para un alumno; el nombre está separado por nombre, apellido paterno y apellido materno

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

main()
{
  int j;
  struct nombre_com
   {
   char nombre[30];
   char apellidopat[15];
   char apellidomat[15];
   };

  struct agenda
   {
```

```
   struct nombre_com nombrec;
   int calificacion[3];
   float estatura;
   }alumno;


/* inicializar */
  alumno.nombrec.nombre[0]='\0';
  alumno.nombrec.apellidopat[0]='\0';
  alumno.nombrec.apellidomat[0]='\0';
  for (j=0;j<3;j++)
                alumno.calificacion[j]=0;
  alumno.estatura=0;

/* agregar un elemento */
  printf(" \ndame el nombre ");
  scanf("%s",&alumno.nombrec.nombre);
  printf(" \napellido paterno ");
  scanf("%s",&alumno.nombrec.apellidopat);
  printf(" \napellido materno  ");
  scanf("%s",&alumno.nombrec.apellidomat);
  for (j=0;j<3;j++)
            {
            printf(" dame la  calificacion %d ",j+1);
            scanf("%d",&alumno.calificacion[j]);
            }
  printf(" dame la estatura");
  scanf("%f",&alumno.estatura);

/* imprimir */
  printf("\nnombre\t\tcalificacion 1\tcalificacion 2\tcalificacion
        3\testatura\n\n");
  printf(" %s ",alumno.nombrec.nombre);
  printf(" %s ",alumno.nombrec.apellidopat);
  printf(" %s ",alumno.nombrec.apellidomat);
  for(j=0;j<3;j++)
       {
       printf("\t%8d",alumno.calificacion[j]);
       }
```

(continúa)

(*continuación*)

```
    printf("\t%.2f",alumno.estatura);
    getch();
    return 0;
}
```

En el ejemplo se maneja una estructura *nombre_com* con tres campos de cadena de caracteres: nombre y apellidos. A continuación, la estructura *agenda* cuenta con tres campos: uno de tipo *struct nombre_com*, otro es un arreglo de enteros para las calificaciones y otro de tipo real para la estatura. Al finalizar la estructura se declaró una variable alumno.

Se inicializan los campos, se solicitan datos y se muestran; según el dato que se modifica se indica la trayectoria: se inicia con la variable, sigue el nombre del campo de la estructura y después el nombre del campo contenido. El resto de los campos se manipula de forma similar.

Uso de typedef (definir el nombre de un tipo de dato)

La palabra *typedef* permite asignar un identificador a un tipo de dato primitivo (sinónimo) o de un dato estructurado; el nombre sirve para declarar variables de ese tipo. Este manejo es práctico cuando se usan estructuras, ya que en cualquier parte del programa se podrán declarar variables del tipo de la estructura; además facilita la portabilidad, ya que si cambia el tipo de dato será necesario modificar sólo los tipos declarados.

La definición debe estar declarada antes de ser usada fuera de las funciones. La sintaxis es:

```
typedef tipo de dato identificador;
```

por ejemplo:

```
typedef int Entero;
```

declarar una variable de tipo *int* ahora se llama también *Entero*:

```
Entero Num;
```

para una estructura

```
typedef struct{
            int numero;
```

```
            Entero n;
            }Tipo;
Tipo  numeros;
```

Es buena práctica iniciar el identificador con una letra mayúscula para indicar que se trata de un sinónimo de un tipo de dato.

EJEMPLO 7.5 Usar un nuevo tipo de dato, almacenar en una estructura un dato de tipo `int` y otro de tipo `float`; mostrar en pantalla

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

typedef int Entero;

typedef struct{
            float numReal;
            Entero n;
           }Tipo;

main()
{
  Entero a;
  Tipo b;
  clrscr();
  a=10;
  b.n=1;
  b.numReal=1.10;
  printf(" %d  %d  %f", a,b.n,b.numReal);
  getch();
  return 0;
}
```

Como se puede ver, se declara un nuevo tipo *int* llamado *Entero*, y a continuación un nuevo tipo llamado *Tipo* que contiene un número de tipo *float* y un número de tipo *Entero*.

En el programa principal se declara la variable `a` de tipo `Entero` y la `b` de tipo `Tipo`. Se le asigna valor y se muestran.

Ejercicios resueltos

EJERCICIO 7.1 (*Versión* 1) Almacenar en una estructura el nombre y los apellidos de una persona, y crear otra estructura que contenga el nombre, además de tres calificaciones; almacenar esto en un arreglo, inicializarlo, agregar datos y mostrarlos, usando un tipo de dato definido por el usuario

Descripción

Definir los nuevos tipos de datos y las estructuras.

Inicializar los elementos.

Ingresar datos.

Mostrarlos.

Variables		
Nombre	*Tipo*	*Uso*
`i,j,n`	`Entero`	`Índices para recorrer el arreglo.`
`lAlumno[MAX]`	`Estructura`	`Arreglo de datos tipo estructura.`
`Nombre_com`	`Estructura`	`Estructura para el nombre completo.`
`Alumno`	`Estructura`	`Estructura con todos los datos.`

Codificación

```
/* uso de typedef (tipo de dato de variable) */

#include <stdio.h>
#include <conio.h>
#define MAX 10

typedef struct
   {
   char nombre[30];
   char apellidopat[15];
```

```
   char apellidomat[15];
   }Nombre_com;

typedef  struct
   {
   Nombre_com nombrec;
   int calificacion[3];
   float estatura;
   }Alumno;

main()
{
  int i,j,n;
  Alumno lAlumno[MAX];


/* inicializar */
  for(i=0;i<MAX;i++)
     {
     lAlumno[i].nombrec.nombre[0]='\0';
     lAlumno[i].nombrec.apellidopat[0]='\0';
     lAlumno[i].nombrec.apellidomat[0]='\0';
     for (j=0;j<3;j++)
          lAlumno[i].calificacion[j]=0;
          lAlumno[i].estatura=0;
     }

/* agregar elementos */
  i=0;
  printf(" \n Cuantos elementos se insertaran \t");
  scanf("%d",&n);
  while(i<n)
     {
     printf(" \ndame el nombre %d ",i+1);
     scanf("%s",&lAlumno[i].nombrec.nombre);
     printf(" \napellido paterno %d ",i+1);
     scanf("%s",&lAlumno[i].nombrec.apellidopat);
     printf(" \napellido materno %d ",i+1);
     scanf("%s",&lAlumno[i].nombrec.apellidomat);
```

(continúa)

(*continuación*)

```
    for (j=0;j<3;j++)
            {
            printf(" dame la  calificacion %d ",j+1);
            scanf("%d",&lAlumno[i].calificacion[j]);
            }
    printf(" dame la estatura");
    scanf("%f",&lAlumno[i].estatura);
    i++;
    }

/* imprimir */
  printf("\nnombre\t\tcalificacion 1\tcalificacion 2\t
        calificacion 3\testatura\n\n");
  for(i=0;i<n;i++)
    {
    printf(" %s ",lAlumno[i].nombrec.nombre);
    printf(" %s ",lAlumno[i].nombrec.apellidopat);
    printf(" %s ",lAlumno[i].nombrec.apellidomat);
    for(j=0;j<3;j++)
      {
      printf("\t%8d",lAlumno[i].calificacion[j]);
      }
    printf("\t%.2f",lAlumno[i].estatura);
    printf("\n");
    }
  getch();
  return 0;
}
```

Explicación

Se define el struct *Nombre_com*, que contiene los campos: *nombre*, *apellidopat*, *apellidomat* de tipo carácter. A continuación se define el *struct Alumno* que contiene en un campo el *struc nombre_com* (ya definido) además de los campos *calificacion*, que es un arreglo de tres elementos, y *estatura*, que es de tipo real. Al final está la variable alumno, un arreglo de estos *structs*.

El primer paso fue inicializar todos los elementos del arreglo alumno, con el fin de línea o con 0, según corresponda a una cadena de caracteres o a un número.

A continuación se pregunta cuántos elementos se almacenarán en el arreglo y el número se almacena en *n*; se piden los datos y finalmente se muestran en pantalla.

Ejecución

```
Cuantos elementos se insertaran   2
dame el nombre 1  Juan
apellido paterno 1  Perez
apellido materno 1 Lopez
 dame la  calificacion 1 99
 dame la  calificacion 2 78
 dame la  calificacion 3  86
 dame la estatura 1.79

dame el nombre 2  Luis
apellido paterno 2  Ibarra
apellido materno 2 Gomez
 dame la  calificacion 1 85
 dame la  calificacion 2 97
 dame la  calificacion 3  68
 dame la estatura  2 1.59

nombre      calificacion 1        calificacion 2
calificacion 3    estatura
Juan perez  lopez                   99                    78
86                      1.79
Luis Ibarra Gomez                   85                    97
68                      1.59
```

EJERCICIO 7.2 (*Versión 2*) Almacenar en una estructura el nombre y los apellidos de una persona, y crear otra estructura que contenga el nombre, además de tres calificaciones; almacenar esto en un arreglo. Inicializar el arreglo, agregar datos y mostrarlos, usando un tipo definido por el usuario. (Solución usando funciones y apuntadores)

```
/* estructuras anidadas usando apuntadores */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

(*continúa*)

(*continuación*)

```
struct t_nombre{
              char nom[10];
              char apem[10];
              char apep[10];

              };

struct t_datos{
              struct t_nombre nombrec;
              int edad;

              };

void leerNombre(struct t_datos *nn);

void imprimirNombre(struct t_datos *nn);

main()
{
 struct t_datos n[3];
 clrscr();

 leerNombre(&n[0]);
 imprimirNombre(&n[0]);
 getch();
}

void leerNombre(struct t_datos *nn)
{
 printf(" \n nombre ");
 scanf(" %s",nn->nombrec.nom);
 printf(" \n apell mat ");
 scanf(" %s",nn->nombrec.apem);
 printf(" \n ape pat ");
 scanf(" %s",nn->nombrec.apep);
 printf(" \n edad ");
 scanf(" %d",&nn->edad);
 getch();
}
```

```
void imprimirNombre(struct t_datos *nn)
{
 printf(" nombre: %s",nn->nombrec.nom);
 printf(" ap paterno: %s",nn->nombrec.apep);
 printf(" ap materno: %s",nn->nombrec.apem);
 printf(" edad: %d",nn->edad);

}
```

EJERCICIO 7.3 (*Versión 3*) Almacenar en una estructura el nombre y los apellidos de una persona, y crear otra estructura que contenga el nombre, además de tres calificaciones; almacenar esto en un arreglo. Inicializar el arreglo, agregar datos y mostrarlos. (Solución usando un tipo de dato definido por el usuario)

```
/* manejo de estructuras anidados con arreglos */

#include <stdio.h>
#include <conio.h>
#define MAX 10

typedef struct{
              char nombre[20];
             char apellp[10];
             char apellm[10];
            }nomc;

typedef struct {
               nomc nom;
               char direccion[15],tel[10];
               }persona;

typedef struct {
               persona pers;
               int n;
               } registrop;
```

(continúa)

(*continuación*)

```
main()
{
      registrop datos[MAX];
      int i;
      clrscr();
      printf("\n datos de alumnos \n");
      printf(" cuantos alumnos se ingresan ");
      scanf("%d",&datos[0].n);
      fflush(stdin);
      for (i=0;i<datos[0].n;i++)
        {
        printf("\n dame el nombre ");
        gets(datos[i].pers.nom.nombre);
        printf("\n dame el apellido paterno ");
        gets(datos[i].pers.nom.apellp);
        printf("\n dame el apellido materno ");
        gets(datos[i].pers.nom.apellm);
        printf("\n dame la direccion ");
        gets(datos[i].pers.direccion);
        printf("\n dame el teléfono ");
        gets(datos[i].pers.tel);
        }
      /* mostrar */
      clrscr();
      printf("\t nombre\t apellidos \tdireccion\t\ttelefono\n");
      for (i=0;i<datos[0].n;i++)
          {
          printf("\t %s\t",datos[i].pers.nom.nombre);
          printf("\t %s\t",datos[i].pers.nom.apellp);
          printf("\t %s\t",datos[i].pers.nom.apellm);
          printf("%s\t",datos[i].pers.direccion);
          printf("%s\n\n",datos[i].pers.tel);
          }
      getch();
      return 0;
}
```

7.2 Apuntadores

Un apuntador es una variable que contiene una dirección de memoria como valor almacenado. Es decir, el apuntador almacena la dirección de una variable, y esa variable contiene un valor.

El apuntador debe declararse indicando el tipo de dato al que apunta e inicializarse antes de ser utilizado.

Los operadores de un apuntador son `&` y `*`. El símbolo `&` es un operador unario y regresa la dirección de la variable apuntada; el operador `*` de indirección o de referencia devuelve el valor contenido en la variable apuntada.

El lenguaje C usa los apuntadores en la manipulación de datos simples, arreglos, estructuras y funciones; en este punto se explican los datos simples y la aritmética.

Al momento de ser declarado, al apuntador se le indica el tipo del dato al que apunta. El apuntador es un tipo de dato simple que siempre se asocia a otro tipo que puede ser simple o estructurado. Por ejemplo, apuntar a `float`, apuntar a estructura, apuntar un tipo definido por el usuario, etcétera.

La declaración es similar a la de cualquier variable, sólo que se antepone un asterisco antes del identificador.

```
<Tipo dato> *<identificador>;
```

Donde *Tipo dato*, es el tipo de dato, que puede ser simple o estructurado; `*` (asterisco) indica que la variable almacena una dirección de memoria; *identificador* es el nombre asignado a la variable.

```
int *apint;
float *apfloat;
```

Ahora se declara una variable de tipo `float` y otra tipo `int`:

```
int dato1;
float dato2;
```

Para asignar las direcciones de las variables a los apuntadores:

```
apint=&dato1;
apfloat=&dato2;
```

Entonces `apint` almacena la dirección de `dato1` y `apfloat` almacena la dirección donde se encuentra almacenada la variable `dato2`. Para modificar el valor de `dato1` y `dato2`, mediante los apuntadores

```
*apint=3;
*apfloat = 3.2;
```

Ahora `dato1` almacena el valor 3, y `dato2` almacena 3.2. Es importante asignar a los apuntadores la dirección de las variables antes de hacer operaciones. Veamos un ejemplo.

EJEMPLO 7.6 Definir dos variables y dos apuntadores a esas variables; asignar valores a las variables usando los apuntadores

```
/* 1. manejo de apuntadores
 septiembre 2010 */

#include <stdio.h>
#include <conio.h>

main()
{

  /* declaración de variables y apuntador a ese tipo de variable */
  int dato1;
  int *apint;

  float dato2;
  float *apfloat;

  printf ("\n Manejo de apuntadores 1\n");

  clrscr();
  /* Asignando la dirección a los apuntadores*/
  apint = &dato1;
  apfloat = &dato2;

  /* Asignando valores a las variables */
  *apint = 3;
  *apfloat = 3.2;
```

```
 printf ("\n dato1 %d  direccion dato 1 %d contenido apuntador %d
          ", dato1,&dato1, apint);
printf ("\n dato2 %.2f  direccion dato 2 %d contenido apuntador %d
         ", dato2,&dato2, apfloat);

 /* modificando las variables */
 *apint = 45;
 *apfloat = 51.3;

 printf ("\n dato1 %d  direccion dato 1 %d contenido apuntador %d
          ", dato1,&dato1, apint);
 printf ("\n dato2 %.2f  direccion dato 2 %d contenido apuntador
        %d  ", dato2,&dato2, apfloat);

 getch();
 return 0;
}
```

En este ejemplo se asignan los valores y se muestra el contenido de las variables y su dirección, además del contenido de los apuntadores. `dato1` queda con el valor de 3 y `dato2` con el valor de 3.2. En la segunda impresión, se observa que el valor de los datos se modificó: mediante la manipulación del apuntador, `dato1` tiene ahora un valor de 45 y `dato2` un valor de 51.3 (la dirección de las variables se mantiene).

Es posible realizar operaciones aritméticas usando apuntadores, como obtener el residuo de dividir entre 2 el `dato1`, y a la variable `dato2` sumarle 10.

```
*aptint = 3;
*apfloat = 3.2;
*apint = *apint % 2;
*apfloat = *apfloat+10;
```

Con esto, `dato1` almacena un 1 que es el residuo y `dato2` almacena 13.2. Es importante observar que un apuntador es una dirección en memoria y es un valor entero, pero un apuntador no es un entero.

Para operaciones de incremento o decremento se puede proceder así:

```
*apint = ++(*apint) ;
*apfloat = --(apfloat);
```

Resultando en *dato1* el valor 2 y en *dato2* el valor 12.2.

EJEMPLO 7.7 Realizar operaciones aritméticas con apuntadores y mostrar el resultado. La primera variable inicializa con el valor de 3, luego se obtiene el residuo de dividir entre 2 y finalmente se incrementa en 1. La segunda variable, de tipo `float`, inicia con el valor 3.2, después se le suma 10 y finalmente se reduce en uno

```
/* 2. aritmetica de apuntadores  */

#include <stdio.h>
#include <conio.h>

main()
{

  /* declaración de variables y apuntador a ese tipo de variable */
  int dato1;
  int *apint;

  float dato2;
  float *apfloat;

  printf ("\n Aritmética de apuntadores 2\n");

  clrscr();
  /* Asignando la dirección a los apuntadores*/
  apint = &dato1;
  apfloat = &dato2;

  /* Asignado valores a las variables y aplicando residuo y suma*/
  *apint = 3;
  *apfloat = 3.2;

  printf ("\n\n dato1 valor inicial   %d  ", *apint);
  printf ("\n dato2 valor inicial   %.2f", *apfloat);

  *apint = *apint % 2;
```

```
    *apfloat = *apfloat + 10;

    printf ("\n\n dato1 residuo 2    %d  ", *apint);
    printf ("\n dato2 sumado 10   %.2f  ", *apfloat);

    /* operador de incremento y decremento */
    *apint = ++(*apint);
    *apfloat = --(*apfloat);

    printf ("\n\n dato1 incrementado   %d  ", *apint);
    printf ("\n dato2 decrementado   %.2f  ", *apfloat);

    getch();
    return 0;
}
```

El resultado de este ejemplo se ve así:

```
dato1  valor inicial    3
dato2  valor inicial    3.2

dato1  residuo 2        1
dato2  sumando 10     13.20

dato1  incrementado     2
dato2  decrementado   12.20
```

Los apuntadores a datos de tipo estructura se manipulan fácilmente, ya que se sustituye el operador punto por el de flecha.

La operación (*apint)=10; es similar a apint->10;

EJEMPLO 7.8 Leer en una estructura anidada los datos del nombre completo y la edad, accediendo mediante un apuntador a estructura

```
/*  manejo de estructuras con apuntadores  */

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

(continúa)

(*continuación*)

```
#include <stdlib.h>

main()
{
 struct t_nombre{
              char nom[10];
              char apem[10];
              char apep[10];

              };

 struct t_datos{
              struct t_nombre nombrec;
              int edad;

              };

 struct t_datos *nn;
 struct t_datos n;
 clrscr();
 nn=&n;

 printf("\n\n Leer datos \n");
 printf(" \n nombre ");
 scanf(" %s",nn->nombrec.nom);
 printf(" \n apell mat ");
 scanf(" %s",nn->nombrec.apem);
 printf(" \n ape pat ");
 scanf(" %s",nn->nombrec.apep);
 printf(" \n edad ");
 scanf(" %i",&nn->edad);
 printf("\n\n Mostrar datos \n");
 printf(" nombre: %s",nn->nombrec.nom);
 printf(" ap paterno: %s",nn->nombrec.apep);
 printf(" ap materno: %s",nn->nombrec.apem);
 printf(" edad: %d",nn->edad);
 getch();
 return 0;
}
```

En este ejemplo se declaran las estructuras y dos variables: una para la estructura y una para el apuntador *nn* del tipo de la estructura. Se accede a cada miembro de la estructura mediante el apuntador *nn*.

Resumen

La estructura consiste en un conjunto de variables agrupadas con un nombre; las variables pueden ser del mismo o de diferente tipo de datos. Generalmente, los datos están relacionados.

Para definir una estructura se indica la palabra reservada `struct` y, entre llaves, los campos con tipo de dato e identificador.

Para tener acceso a la estructura se escribe el nombre de la variable y el operador punto (.) seguido del identificador del campo a acceder para la lectura o escritura.

Para definir un nuevo tipo de dato se puede utilizar `typedef`; con esto se asigna un nombre diferente a un tipo de dato. Esta práctica es común en las estructuras.

Es posible definir una estructura que contenga otras estructuras; a esto se le llama anidamiento de estructuras. Otra aplicación es la definición de un arreglo que contenga en cada posición una estructura; esto se conoce como arreglo de estructuras.

Un apuntador es una variable de tipo básico que almacena la dirección de memoria de una variable; se usa para tener acceso a variables por su dirección, y se aplica en la manipulación de datos simples, arreglos, estructuras y funciones. Tiene dos operadores: el &(ampersand) y el *(asterisco); el primero devuelve la dirección de memoria de una variable y el segundo devuelve el contenido de la variable apuntada por el apuntador.

Evaluación

I. Conteste las siguientes preguntas.

1. Una estructura es un tipo de dato _____.
2. Una estructura consiste en un conjunto de variables con el mismo tipo de datos.
   Sí ( ) No ( )

3. Una estructura consiste en un conjunto de variables que pueden ser de diferente tipo de dato.
   Sí ( ) No ( )

4. ¿Puede una estructura contener otra?
   Sí ( ) No ( )

5. El operador punto (.) sirve para tener acceso a _____ de la estructura.

6. La palabra `typedef` permite definir un nuevo tipo de dato.
   Sí ( ) No ( )

7. ¿Tiene el apuntador un tipo de dato?
   Sí ( ) No ( )

8. ¿Es posible hacer operaciones aritméticas usando apuntadores?
   Sí ( ) No ( )

9. Cuando el operador & antecede a una variable, entonces devuelve la _____ de la variable.

10. Cuando el operador * antecede una variable, entonces se tiene acceso al contenido de _____ apuntada.

II. Muestre el resultado del siguiente ejercicio.

Definir una estructura con datos de un paciente: nombre, fecha ingreso, fecha salida, y otra estructura que contenga la estructura del paciente, además de su estado de salud. Permitir almacenar datos, buscar un paciente y ordenar los pacientes por fecha de ingreso.

Ejercicios propuestos

I. Realice un programa que efectúe lo siguiente.

1. Defina el mundo con cinco continentes> compuestos por países que, a su vez, contienen ciudades; las ciudades, finalmente, albergan pueblos.

2. Haga las definiciones necesarias y la declaración de la variable del mundo. Suponga que de cada continente, país, ciudad y pueblo se conoce el nombre y el número de habitantes.
3. Muestre las poblaciones de un continente.
4. Obtenga el total de habitantes de uno de los continentes.
5. Muestre el promedio de la edad de los habitantes por población, país y continente.
6. Muestre toda la información por continente, país y población.

II. Describa qué realiza el siguiente programa.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>

struct Pizza{
       int tam;
       float precio;
       char ing[3][15];
       };

main()
{
  struct Pizza pedidos[10];
  int indice=0,opc,cont,i;
  do{
     do{
     clrscr();
     printf("***************** PIZZAS  *************\n\n\n\n");
     printf("1. Hacer pedido\n2. mostrar\n3. Salir\n\nQue desea
        hacer?:  ");
     scanf("%d",&opc);
        }while(opc!=1 && opc !=2&&opc!=3);

  //************  ESCOGER TAMAÑO
     if(opc==1)
```

```
{
  clrscr();
  printf("\n tamanio de su pizza?\n\n\n\n");
  printf("1. Chica\n2. Mediana\n3. Grande\n\n\nOpcion:\t  ");
  scanf("%d",&pedidos[indice].tam);
//*********** APLICAR PRECIO
  if(pedidos[indice].tam==1)
    pedidos[indice].precio=45;
  if(pedidos[indice].tam==2)
    pedidos[indice].precio=70;
  else pedidos[indice].precio=100;
//************ ESCOGER INGREDIENTES
  cont=0;
  do{
  clrscr();
  printf("\n ingredientes de su pizza? (maximo 3)\t");
  printf("1. Jamon\n2. Tocino\n3. Elote\n4. Jalapenio\n5.
     Salchicha
          \n6. Extra queso\n7. Listo\n\n\nAgregar
             ingrediente:  ");
   scanf("%d",&i);
   if(i==1) strcpy(pedidos[indice].ing[cont],"Jamon");
   if(i==2) strcpy(pedidos[indice].ing[cont],"Tocino");
   if(i==3) strcpy(pedidos[indice].ing[cont],"Elote");
   if(i==4) strcpy(pedidos[indice].ing[cont],"Jalapeno");
   if(i==5) strcpy(pedidos[indice].ing[cont],"Salchicha");
   if(i==6) strcpy(pedidos[indice].ing[cont],"Queso");
   printf("\nIngrediente agregado");
   cont++;
   getch();
       }while(cont<3 && opc!=7);
   clrscr();
   printf("Esta listo su pedido");
   indice++;
   }
if(opc==2)
{
   for(i=0;i<indice;i++)
   {
```

```
    printf(" %d   precio  %.2f?n",pedidos[i].tam,pedidos[i].precio);
    getch();
    }
   }
   }while(opc!=3);
}
```

III. Codifique el siguiente programa y cada uno de los incisos.

1. Defina un arreglo de registros que contenga: nombre del profesor, materias impartidas, código de profesor y salario.
2. Liste los profesores con todos sus datos.
3. Busque un profesor y muestre sus datos.
4. Muestre todos los profesores que imparten una materia en particular.
5. Muestre el profesor con el salario más alto.

CAPÍTULO 8

Funciones (programación modular)

La programación modular es una técnica que consiste en separar un problema en las diferentes tareas que se quieren resolver, dando origen a la creación de módulos (pequeños programas a los que llamaremos funciones), donde cada módulo o función se diseña, se codifica y se procesa de manera independiente.

A manera de ejemplo, supongamos que se diseña un programa que calcula el sueldo del trabajador de una empresa. Pensar en el problema de inicio a fin es demasiado complejo, ya que implica muchos cálculos. Por lo tanto, es mejor dividirlo de tal forma que en la etapa de análisis se identifiquen las distintas tareas a resolver para calcular un sueldo: por ejemplo, cuánto se debe pagar de IMSS, cuánto de ISPT, cuánto de horas extra, si es que las tuvo, etcétera. Se podría diseñar una función (o módulo) independiente para cada una de estas tareas y al final "armar" todas estas funciones en un solo programa.

Esto implica pensar en lo general de un sueldo y desglosar el problema hasta sus operaciones más básicas o particulares, llevando a cabo un diseño descendente, también conocido como top down.

El uso de las funciones hace la programación más fácil y eficiente pues permite:

- Reducir la complejidad del programa ("divide y vencerás").
- Eliminar código duplicado.
- Controlar fácilmente los efectos de los cambios.
- Ocultar detalles de implementación.
- Reutilizar código.
- Facilitar la legibilidad del código.

Una función realiza una tarea específica agrupando un conjunto de instrucciones con un nombre. Para que se ejecuten las instrucciones contenidas en la función ésta se debe invocar o llamar mediante su nombre en otra función, la cual puede ser `main`.

Cada función puede ser diseñada, verificada sintácticamente y depurada de manera independiente; en otras palabras, cada función se puede probar sin tener que esperar a que estén programadas todas las demás funciones que también se usarán en el programa. Sin embargo, las funciones no pueden actuar de manera autónoma, es decir, no pueden ejecutarse por sí mismas: su ejecución siempre dependerá de que sea invocada en alguna otra función y esto a su vez debe estar relacionado siempre con la ejecución de una función `main()`.

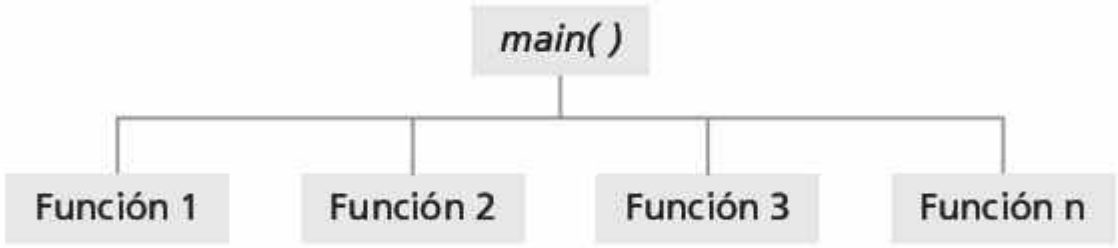
En el lenguaje C se pueden distinguir dos tipos de funciones: las predefinidas y las definidas por el programador. Las funciones predefinidas se encuentran en las bibliotecas estándar de C, y es necesario hacer uso de la directiva `#include` para invocarlas. Con respecto a las definidas por el programador, éste las diseña según sus necesidades.

Durante el presente texto se ha estado invocando varias de las funciones predefinidas en distintos ejemplos: por ejemplo `printf`, `scanf`, `clrscr`, entre otras. El objetivo de este capítulo está orientado a explicar al lector la forma de crear sus propias funciones.

Función `main ( )`

Es la función principal y puede contener de pocas a muchas líneas; su papel es coordinar a las otras funciones mediante llamadas o invocaciones. El siguiente diagrama muestra la jerarquía que existe en un programa modular en lenguaje C,

en el cual se puede ver que siempre debe existir una función `main` y ésta puede hacer uso de cualquier cantidad de funciones, ya sean creadas por el usuario o predefinidas en el lenguaje. Las funciones invocadas por `main` pueden llamar a su vez otras funciones.



Las funciones definidas por el usuario tienen todas las propiedades y algunas características similares a las de los programas que hemos manejado anteriormente, pero para poder utilizar funciones es necesario primero definirlas o crearlas. Definir una función significa codificarla y la forma de definir funciones puede variar, debido a que trabajan de diferentes maneras.

8.1 Funciones

Como ya se mencionó, cada función se diseña de manera independiente. La acción de diseñar o crear una función en C es conocida también como definirla y, según el propósito específico, las funciones pueden ser diseñadas de las siguientes maneras:

1. **Funciones sin paso de parámetros.** Son subprogramas que no requieren información adicional de su entorno, pues simplemente ejecutan una acción cada vez que son invocadas.
2. **Funciones con paso de parámetros.** Para la ejecución de estos subprogramas se requiere además de su invocación, que se le pase información adicional de su entorno.
3. **Funciones que no regresan valor.** Subprogramas que luego de su ejecución no devuelven al entorno algún valor como resultado de su ejecución.
4. **Funciones que regresan valor.** Funciones que luego de su ejecución generan un valor como resultado y "entregan" ese valor a su entorno.

Todas estas funciones se pueden combinar; es decir, se puede diseñar una función con parámetros que regrese valor o que no lo haga, si así se requiere; o bien diseñar una función sin parámetros que regrese valor en un mismo programa. Esto dependerá del programador, y de cómo decida que es más conveniente el diseño de la función.

El entorno de cualquier función es la función por la que es invocada. Por ejemplo, si *main* invoca a una función diseñada con paso de parámetros y que regresa un valor, entonces *main* es el entorno de ésta, y será *main* la que le proporcione la información (parámetros) que dicha función requiera y la que reciba el valor que dicha función devuelva. La estructura general para definir una función en C es:

```
*Tipo_dato   identificador (*lista de parametros)
{
   * variables locales;
   cuerpo de la funcion;
   *return dato;
}
```

El * significa que pueden o no aparecer, dependiendo del diseño de la función.

En la tabla 8.1 se describen los aspectos más sobresalientes en el diseño de una función, según la definición descrita.

TABLA 8.1 Descripción de la definición de función

Concepto	*Explicación*
Tipo dato	*Es el tipo de dato del valor que devuelve la función, si es que la función lo hace. Si no devuelve valor, debe iniciar con la palabra reservada* void. *Si se omite el tipo de dato, de manera predeterminada devuelve un entero.*
Lista de parámetros	*Es una lista de variables con sus respectivos tipos de datos que utiliza el siguiente formato: tipo1 parametro1, tipo2 parametro2,..., Cuando existen, éstos son los datos que debe recibir la función cuando se le invoque.*
Cuerpo de la función	*Son las sentencias o instrucciones que ejecutará la función cada vez que sea invocada.*
Variables locales	*Las constantes y variables declaradas dentro de la función son locales a la misma y no existen fuera de ella.*
Valor devuelto por la función	*Mediante la palabra reservada return se puede devolver el resultado de la función, si es que la función requiere regresarlo.*
Identificador	*Es el nombre asignado a la función.*

Iniciemos explicando la forma más sencilla de emplear funciones: sin paso de parámetros ni devolución de valor.

```
void identificador ( )
{
   variables locales;
   cuerpo de la funcion;
}
```

donde:

`void`	Palabra reservada que en una función indica que ésta sólo ejecuta sus instrucciones sin devolver nada a su entorno.
`identificador`	Nombre que el programador elige para la función y con el cual la invocará posteriormente.
`( )`	Los paréntesis son parte de la sintaxis de una función: si están vacíos, significa que la función no recibe parámetros.

Observe que aquí se omite la instrucción `return`, puesto que se inicia la función con la palabra `void`.

EJEMPLO 8.1 Resolver el simple problema de la suma de dos *s*, mediante una función sin paso de parámetros ni devolución de valor

```
void suma1( )
{
float a,b,c;
printf("teclea el primer ");
scanf("%f",&a);
printf("teclea el segundo ");
scanf("%f",&b);
c=a+b;
printf("El resultado es %f", c);
}

void main( )
{
suma1();
}
```

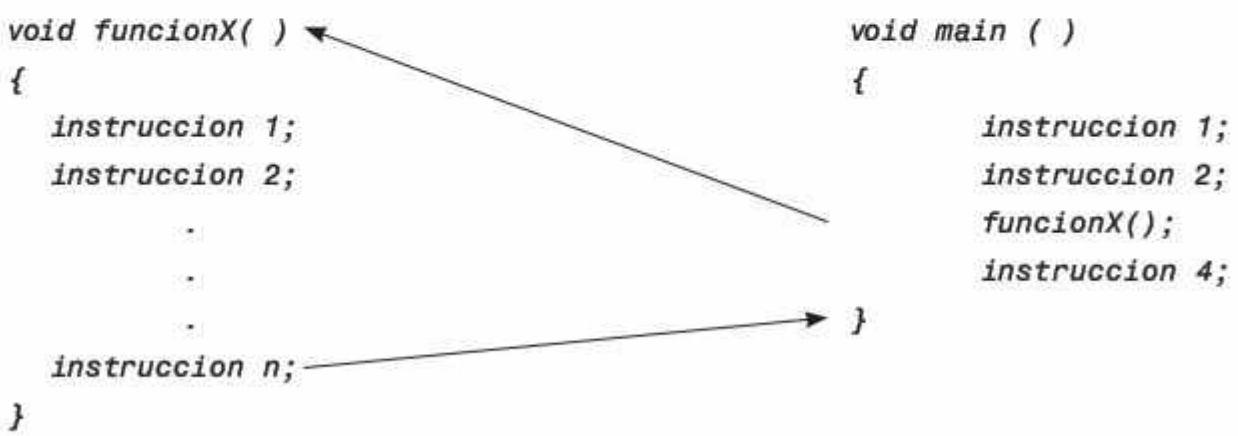
En este ejemplo se programó la función `suma1` y dentro de la misma se declaran tres variables locales: `a`, `b` y `c`. Las instrucciones asignadas a la función `suma1` son solicitar valores al usuario, almacenarlos en `a` y `b` respectivamente; calcular la suma de éstos y almacenarla en `c`; finalmente mostrar el valor de `c`. La función `suma1` hace todo lo necesario para sumar dos números. Prácticamente, la única tarea de `main` es invocar a la función para que ella se ocupe del resto. Como se puede ver en el ejemplo, la llamada a `suma1` es una sentencia más de `main`.

Es importante tener presente que la ejecución de cualquier programa creado mediante funciones, se iniciará siempre en la función `main`.

Quizás el ejemplo anterior es poco práctico, ya que no hay mucha diferencia entre solucionar el problema usando funciones o sin utilizarlas, y el problema realmente es pequeño, pero si empleamos varias tareas en un programa, vale la pena el uso de funciones.

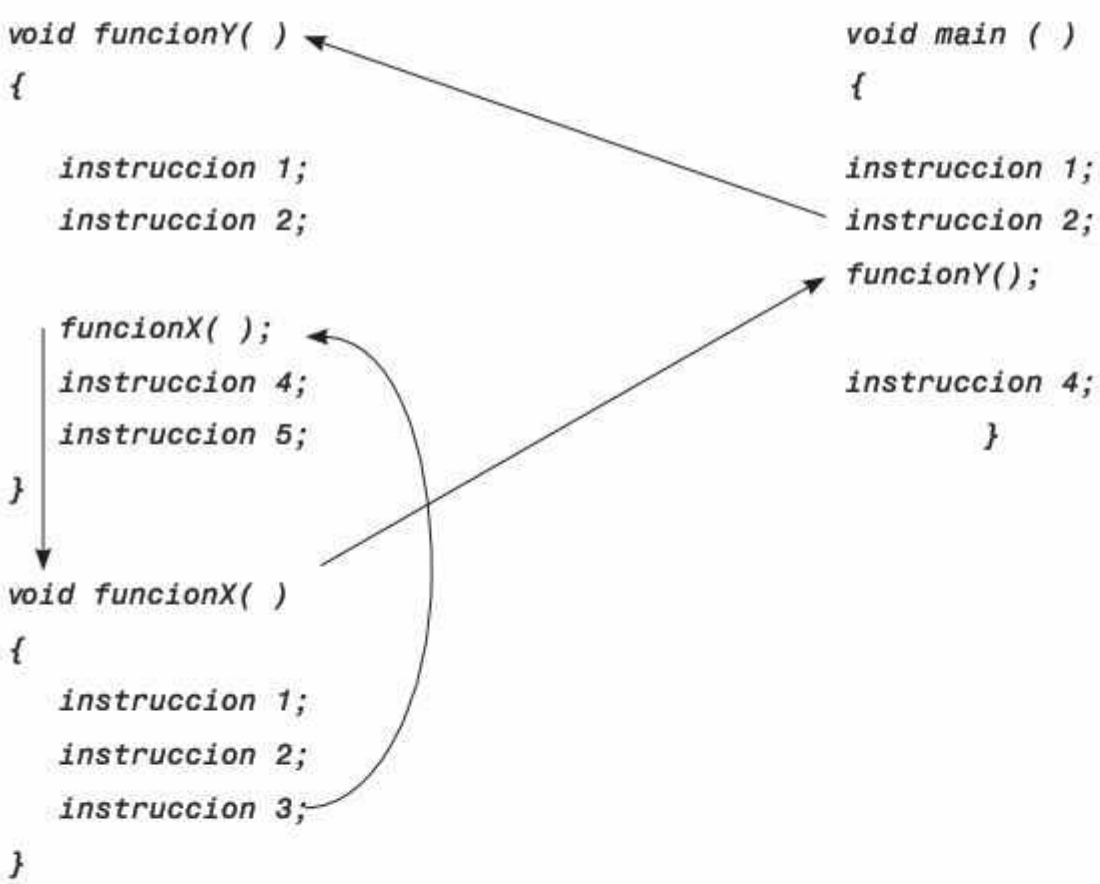
Por otro lado, cabe señalar que `suma1` hace uso de las funciones `printf` y `scanf`, que también son funciones, aunque éstas están predefinidas en el lenguaje C. Esto nos indica que no sólo `main` puede invocar a las funciones, sino que entre ellas también pueden invocarse. La ventaja de las funciones es que una vez diseñadas pueden utilizarse las veces que se requiera, sólo basta con invocarlas y ejecutarán las instrucciones que tengan asignadas. Las funciones `printf` o `scanf` se pueden emplear infinidad de veces. Cada vez que se requiere leer o mostrar algún dato en un programa se invocan y listo; lo mismo sucede con las funciones que el programador diseña.

Como se mencionó anteriormente, un programa modular o con funciones debe contar invariablemente con una función `main`, la cual controla la aparición de cualquier otra función. La ejecución de un programa inicia con lo que contiene `main`, pero en cuanto ésta llama a alguna otra función, se ejecutan las instrucciones de esta última. Una vez concluida la función que se invocó, se regresa el control a la función `main` y se continúa con la siguiente línea hasta terminar su contenido. En el siguiente diagrama las flechas indican el orden en que se lleva a cabo la ejecución.



En este ejemplo se diseña la *funcionX* con *n* instruciones y la función *main* con cuatro instrucciones de las que la tercera es precisamente la invocación a *funcionX*. La ejecución se inicia en *main*, por lo que se ejecutan las instrucciones 1 y 2, pero en cuanto aparece la llamada a *funcionX* se inicia la ejecución de sus *n* instrucciones; al terminarlas, se regresa el control al punto en que se invocó la función recién finalizada, es decir, a *main* para proseguir entonces con la ejecución de la instruccion 4 y terminar el programa.

Sin embargo, las funciones pueden ser invocadas no sólo desde *main*, sino que también se pueden invocar desde cualquier otra función y su ejecución se lleva a cabo de manera similar. Esto se puede observar en el diagrama siguiente:



En este ejemplo se diseñaron *funcionX*, *funcionY* y *main*. Como ya se explicó, la ejecución inicia en *main*, por lo que se ejecutan las instrucciones 1 y 2, cuando viene la instrucción 3 que es la invocación a *funcionY* se ejecutan sus instrucciones 1 y 2 y luego se encuentra con la llamada a *funcionX* por lo que se pasa el control a ésta ejecutando sus tres instrucciones para continuar con *funcionY* que fue la que la llamó; ahora se ejecutan las instrucciones 4 y 5 de *funcionY* con lo que finaliza y se continúa con *main* hasta que finaliza.

EJEMPLO 8.2 Calcular el promedio individual de un conjunto de alumnos

```
void prom( )
{
int n, i;
float c, p=0;
printf("Cuantas materias se le van a promediar al alumno?");
scanf("%i",&n);
for(i=1;i<=n;i++)
   {
      printf("Teclea la calificacion");
      scanf("%f",&c);
      p=p+c;
   }
p=p/n;
printf("El promedio del alumno es: %f", p);
}
```

```
void main( )
{
int resp;
do{
      prom( );
      printf("Si deseas promediar otro alumno teclea 1");
      scanf("%i",&resp);
     }
while(resp==1);
}
```

En este programa se puede apreciar mejor la ventaja en el uso de funciones, ya que *prom* se diseñó una sola vez y en el *main* se invocará las veces que el usuario necesite o decida obtener el promedio individual de un conjunto de alumnos sin salir del programa.

La ejecución una vez más inicia en *main*, donde se declaró la variable local *resp*, y luego se invoca a *prom*; a continuación se ejecutan las instrucciones de esa función. Después se pregunta si se desea calcular otro promedio y, dependiendo de la respuesta, se repite o no el contenido de *do-while*.

No obstante, el diseño específico de estas funciones tiene ciertas desventajas, ya que tanto los valores de entrada como los de salida se almacenan en variables locales y eso puede no ser recomendable en algunos casos.

Variables locales

Las variables locales tienen las siguientes características:

1. Sólo pueden ser reconocidas y utilizadas por la función en la que están declaradas; ninguna otra, ni siquiera `main`, que es el programa principal, tiene acceso a las variables locales declaradas en otra función.
2. Los espacios reservados en memoria para variables locales están disponibles sólo en el momento en que se está ejecutando la función donde fueron declaradas, una vez que se termina la ejecución de la función desaparecen de la memoria, lo que ocasiona que, después de almacenar los datos de entrada y mostrar los resultados en las funciones `suma1` y `prom` que se diseñaron, estos datos se pierdan y no exista forma de recuperarlos.

Analizando los dos ejemplos que hemos revisado, ¿qué debería hacerse si el resultado de la suma de dos números se requiere para algún otro cálculo fuera de la función suma1? O bien, ¿qué se deberá agregar al programa si después de promediar a cada alumno se necesitara calcular el promedio general de todos los alumnos? Como están diseñados estos programas sería imposible, debido a que los resultados se encuentran en variables locales, de tal manera que en cuanto se terminan ambas funciones cada resultado que generan desaparece.

Para poder disponer de los resultados, incluso cuando se ha terminado la función, debemos utilizar otro tipo de funciones.

8.2 Funciones que devuelven valores

Este segundo tipo de funciones trabaja de manera diferente a las anteriores ofreciendo una ventaja adicional: poder tener el resultado que proporciona la función disponible a las demás funciones para operaciones posteriores. Las funciones que a continuación describiremos no requieren paso de parámetros, pero devuelven un valor a su entorno (recuerde que su entorno es la función que la llama y que puede ser `main` o cualquier otra). La sintaxis es la siguiente:

```
tipo_dato identificador ( )
{
   declaración de variables locales;
   cuerpo de la función;
   return (valor);
}
```

Este tipo de funciones inicia con *tipo_dato* y se agrega la palabra reservada *return*:

Tipo_dato	Si una función va a devolver un valor, entonces debe iniciar con el tipo de dato que devolverá, que puede ser cualquier tipo primitivo int, float, char, etcétera.
return	Es la palabra reservada que "envía" el valor que la función devuelve a la función que la manda llamar; el valor es un dato, así que puede ser una variable, una expresión aritmética, etcétera, incluso lo que otra función devuelva.

EJEMPLO 8.3 Diseñar la solución para el caso de la suma de dos *s* mediante una función que regresa un valor: el resultado del cálculo de la suma

```
float suma1( )
{
float a,b,c;
printf("teclea el primer "):
scanf("%f",&a);
printf("teclea el segundo "):
scanf("%f",&b);
c=a+b;
return c;
}


void main( )
{
float res;
res=suma1();
printf("El resultado es %f", res);
}
```

En este ejemplo la función *suma1* inicia con el tipo de dato *float*, lo cual significa que regresará un valor real a la función que la invoca (en este caso *main*). Las acciones en la ejecución de *suma1* son solicitar los valores al usuario, almacenarlos en *a* y *b* respectivamente, almacenar el resultado en *c*, y devolver el valor de *c* a *main*.

Es necesario aclarar que la función regresa sólo el valor de la variable *c*, pero ésta junto con las variables *a* y *b* desaparecen una vez concluida la función *suma1*. Por lo tanto *main* se encarga de recibir el resultado de la operación; en este caso solamente para imprimirlo.

Por otro lado, también es importante observar que el *main* de esta nueva versión tiene una ligera diferencia con el de la anterior en la llamada a *suma1*, ya que si una función devuelve un valor es obligación del programa que la invoca "cachar" o recibir de alguna manera el valor que retorna la función invocada, por este motivo se declara una variable llamada *res*, en la que mediante una asignación *main* recibe el valor que le devuelve *suma1* luego de su ejecución.

El mismo programa con algunas variaciones:

```
float suma1( )
{
float a,b;
printf("teclea el primer ");
scanf("%f",&a);
printf("teclea el segundo ");
scanf("%f",&b);
return a+b;
}


void main( )
{
printf("El resultado es %f", suma1());
}
```

En la función *suma1* se prescindió de la variable *c*, ya que se regresa directamente el valor de la expresión aritmética *a+b*. En el caso de *main* también se omitió la variable *res*, puesto que en este ejemplo la función *main* recibe el valor que le regresa *suma1*, utilizándola como argumento de *printf* y es el valor de *a+b* lo que aparecerá en la pantalla.

La sentencia `printf("%f", suma1());` significa "imprime lo que regresa la función `suma1`".

Veamos otro ejemplo para mostrar la ventaja de haber modificado la función `suma1` para que ahora devuelva un valor. Se diseñará un programa que sume cuatro valores proporcionados por el usuario. Pero si nuestra función suma sólo dos números, ¿qué se puede hacer en este caso? ¡Fácil! Se llama dos veces a la función.

```
float suma1( )
{
float a,b;
printf("teclea el primer ");
scanf("%f",&a);
printf("teclea el segundo ");
scanf("%f",&b);
return a+b;
}

void main( )
{
printf("El resultado es %f", suma1()+suma1());
}
```

En este ejemplo, la función `suma1` es invocada dos veces. Primero se llama como argumento de la única instrucción de `main`, que es `printf`; la primera llamada a `suma1` pide dos valores al usuario, hace la suma y la regresa a donde fue invocada. Después hay una segunda llamada en la que se piden los otros dos valores y se devuelve también el resultado a donde se invocó. Finalmente, `main` hace la suma de los dos valores y los imprime.

EJEMPLO 8.4 Diseñar, retomando el ejemplo de la función que calcula el promedio, un programa donde la función devuelva un valor que se acumule en una variable en `main` para luego obtener el promedio grupal de *n* alumnos

```
float prom( )
{
int n, i;
float c, p=0;
```

```
printf("Cuantas materias se le van a promediar al alumno?");
scanf("%i",&n);
for(i=1;i<=n;i++)
   {
      printf("Teclea la calificacion");
      scanf("%f",&c);
      p=p+c;
   }
p=p/n;
return p;
}

void main( )
{
int resp, c= 0;
float promg=0;
do{
 promg = promg + prom( );
 printf("Si deseas promediar otro alumno teclea 1");
 scanf("%i",&resp);
 if(resp)
    c++;
     }
while(resp==1);
printf("El promedio general de los %d alumnos es %f", c, promg/c );
}
```

Veamos un nuevo ejemplo para observar cómo una función puede ser llamada por otra función, no necesaria ni exclusivamente por el *main*.

EJEMPLO 8.5 Calcular el área de un trapecio

```
float sumabases( )
{
float bm,BM;
printf("Dame la base menor ");
scanf("%f",&bm);
printf("Dame la base mayor");
```

(continúa)

(*continuación*)

```
scanf("%f",&BM);
return (bm+BM);
}


float area ( )
{
  float h;
  printf("Dame la altura ");
  scanf("%f",&h);
  return sumabases( ) * h/2;
}

void main()
{
  printf("%f",area( ) );
}
```

La función `main` invoca a la función `area`, en ésta se pide la altura y luego invoca a la función `sumabases`, la cual solicita el valor de las bases y regresa el resultado de la suma a `area`, que es la que la llama en su definición; cuando `area` recibe el valor de `sumabases` lo utiliza para concluir el cálculo, y a su vez regresa el resultado a `main` y ahí se imprime.

8.3 Funciones con paso de parámetros

Hasta ahora, el diseño de nuestras funciones ha hecho que `main` simplemente las invoque, ya que con sólo llamarlas piden directamente al usuario los valores de entrada requeridos para trabajar. Pero existen otro tipo de funciones donde hay una comunicación más estrecha en el contexto de un programa modular: las funciones con paso de parámetros.

Una función con paso de parámetros es aquella que además de ser invocada requiere información por parte del subprograma que la llama. Esta información se refiere a los datos de entrada que se necesitan para que la función trabaje.

Hay dos formas de pasar datos a las funciones: paso por valor y por referencia. Estas funciones pueden tener uno o varios parámetros que pueden entregarse sólo por valor, sólo por referencia o de ambas formas. Los parámetros pueden ser de un mismo tipo de datos o de tipos de datos diferentes, según se requiera en el diseño de la función: por ejemplo, una función puede recibir únicamente datos enteros o, si así se requiere, recibir datos de tipo entero, real o carácter.

8.3.1 Funciones con parámetros por valor

La forma general de este tipo de función es la siguiente:

```
tipo_dato identificador (tipo_dato parámetro )
{
  declaración de variables locales;
  cuerpo de la función;
  return (valor);
}
```

Lo nuevo en el diseño de la misma es el contenido de los paréntesis:

`(tipo_dato parámetro)`	Variables (parámetros) antecedidas por su tipo de dato donde la función recibe los valores que requiere para trabajar por parte del módulo que la invoca. Puede ser uno o varios.

EJEMPLO 8.6 Resolver nuevamente el problema de la suma de dos números, pero usando una función con paso de parámetros por valor

```
float suma3(float a, float b)
{
  return a+b;
}


void main( )
{
  float n1, n2, c;
  printf("Dame el primer valor");
```

(continúa)

(*continuación*)

```
    scanf(("%f",&n1);
    printf("Dame el segundo valor");
    scanf(("%f",&n2);
    c=suma3(n1,n2);
    printf("La suma es %f", c);
}
```

Observe cómo la función *suma3* se declara con dos parámetros de tipo real (*a* y *b*), los cuales son una especie de "molde" donde la función recibe los valores que necesita para trabajar.

Las acciones de *main*, al ejecutarse, son las siguientes: solicitar los datos al usuario y almacenarlos en *n1* y *n2* respectivamente, e invocar a *suma3* con los valores almacenados en *n1* y *n2* (los contenidos de *n1* y *n2*, entran a la función a través de los parámetros *a* y *b* en el orden en que se enviaron, es decir, el valor de *n1* entra a través de *a* y *n2* entra a través de *b*). Se almacena la suma en *c*; *suma3* devuelve el valor de la variable *c* a *main*. Finalmente *main* imprime el resultado.

Parámetros formales y argumentos

Los parámetros formales (también denominados parámetros ficticios) son las variables que se declaran dentro del paréntesis junto con la función; en este caso *a* y *b*. Por otro lado, los valores con que se invoca a la función se denominan argumentos o parámetros reales. En el ejemplo tales argumentos son representados por *n1* y *n2*.

Frecuentemente, el término parámetro se utiliza indistintamente tanto para referirse a los ficticios como a los reales; sin embargo, aquí se denominarán parámetros a los ficticios y argumentos a los reales. Dicho de otra manera: son parámetros los que se declaran en la definición de la función y argumentos los valores con que se invocan a la misma.

Cabe señalar también que los parámetros son variables siempre, mientras que los argumentos pueden ser variables, constantes, expresiones aritméticas e incluso lo que devuelva la llamada a otra función, siempre y cuando los argumentos coincidan en cantidad, tipo de dato y orden con los parámetros tal como se hayan declarado en la función.

En el ejemplo, *suma3* fue definida para recibir dos parámetros de tipo real (*a* y *b*), por lo tanto, cuando en *main* se invoca *suma3*, a esta variable se le entregan dos argumentos (*n1* y *n2*) también de tipo real.

Los parámetros por valor reciben una copia de lo que valen los argumentos; su manipulación es independiente, es decir, una vez finalizada la función, los argumentos continúan con el valor que tenían antes.

Este tipo de manejo de parámetros se denomina función con paso de parámetros por valor.

Para comprender mejor lo anterior, simulemos una corrida de escritorio sobre el programa. Si al solicitar los datos el usuario diera como entradas 4 y 5, sabríamos que se almacenarían en *n1* y *n2*, respectivamente, y que éstos serían los valores que *suma3* recibiría en *a* y *b*. Por tanto, *suma3* devolverá el valor de 9 a *main*, que lo recibiría en *c*, y después lo imprimiría.

Si en el *main* se imprimieran los valores de *n1* y *n2*, se visualizaría 4 y 5, ya que *suma3* recibió sólo la copia de los valores de *n1* y *n2*, pero es incapaz de modificarlos pues no tiene acceso a la dirección de memoria asignada a estas variables.

EJEMPLO 8.7 Crear la función para sumar dos números, pero con ligeros cambios a *main* para que el programa calcule la suma de cuatro *s*

```
float suma3(float a, float b)
{
   return a+b;
}

void main( )
{
float n1, n2, n3, n4, c;
printf("Dame el primer valor");
scanf(("%f",&n1);
printf("Dame el segundo valor");
scanf(("%f",&n2);
printf("Dame el tercer valor");
scanf(("%f",&n3);
```

(continúa)

(*continuación*)

```
printf("Dame el cuarto valor");
scanf(("%f",&n4);
c=suma3(suma3(n1,n2),suma3(n3,n4));
   printf("La suma es %f", c);
}
```

`main` solicita los cuatro valores al usuario y una vez que los tiene almacenados hace una llamada a `suma3`, pero en esta ocasión los argumentos de la primera invocación a `suma3` son llamadas la misma función: en una instrucción se invoca en tres ocasiones a `suma3`.

La primera en ejecutarse es la llamada a `suma3(n1,n2)`, la cual devuelve un valor; la segunda función en ejecutarse es `suma3(n3,n4)` y regresa un segundo valor. Estos dos valores son tomados como argumentos de la función `suma3(suma3(n1,n2),suma3(n3,n4))`, y todo se almacena en `c` para después imprimirse. Es importante mencionar que las llamadas a las funciones se van ejecutando de adentro hacia afuera.

EJEMPLO 8.8 Crear un programa con dos funciones: una que calcule el cuadrado de un número y otra que calcule el cubo. Ambas deben devolver el resultado

```
int cuadrado(int x)
{
return(x * x);
}

int cubo(int y)
{
return cuadrado(y)*y;
}

void main( )
{
int opc, n;
printf("Elige una opcion 1) CUADRADO\n 2) CUBO\n");
```

```
scanf(("%i",&opc);
printf("Dame el numero a elevar");
scanf(("%i",&n);
switch(opc)
   {
       case 1: printf("%d", cuadrado(n));
       break;
       case 2: printf("%d", cubo(n));
       break;
       default: printf("ERROR");
   }
}
```

Si en el *main* se selecciona la opción 1, se ejecutará *cuadrado*, la cual recibe un valor, lo multiplica por sí mismo y devuelve el resultado a *main*. En caso de que sea seleccionada la opción 2, se invocará a *cubo*, ésta a su vez invoca a *cuadrado*, el cual recibe el resultado, lo multiplica por el parámetro y devuelve el *cubo* a *main*. Observe que tanto *main* como *cubo* hacen una llamada a *cuadrado*: cualquier función puede invocar a otra.

8.3.2 Parámetros por valor y por referencia

La forma de pasar parámetros de una función a otra que se ha estado empleando en los ejemplos explicados hasta ahora se denomina paso por valor. En este tipo de paso de parámetros, las funciones llamadas reciben a través de los parámetros únicamente copia del contenido de los argumentos y no tienen la capacidad de modificar su valor. Veamos un ejemplo.

EJEMPLO 8.9 Realizar una función que intercambie dos valores entre sí

```
#include<stdio.h>
void intercambio(int x, int y)
{
int aux;
aux=x;
x=y;
```

(*continúa*)

(*continuación*)

```
y=aux;
}

void main()
{
int n1,n2;
scanf("%d %d", &n1,&n2);
intercambio(n1,n2);
printf("Los valores son n1 = %d y n2 = %d", n1,n2);
}
```

Cabe recordar a qué nos referimos con parámetros y argumentos: parámetros son las variables que se utilizan en la función de `intercambio` —en este ejemplo `x` y `y`—, mientras que los argumentos son los valores con los que se hace la llamada o invocación de la función (`n1` y `n2`) declarados en el `main` de este mismo ejemplo.

La función `main` solicita dos números al usuario y los recibe en `n1` y `n2`. Suponga que el usuario decide teclear 9 y 10. `main` pasa como argumentos a `n1` y `n2` a la función de intercambio, la cual al ejecutarse recibe el 9 a través de `x` y el 10 a través de `y`, y los intercambia. Al terminar de ejecutarse la función `intercambio`, `main` imprime los valores de `n1` y `n2`. En pantalla aparecería:

```
Los valores son n1 = 9 y n2 = 10
```

En este ejemplo es posible observar lo que se explicó antes: la función `intercambio` no tiene el poder de modificar los valores de los argumentos, sólo los utiliza. Esto se debe a que cuando un parámetro entra por valor a una función, ésta tiene acceso sólo a una copia de lo que vale el argumento, pero sin acceder a la localidad de memoria donde está almacenado el dato. En el ejemplo, la función `intercambio` recibió sólo copia de los valores de `n1` y `n2`, y aunque intercambió los parámetros, los argumentos permanecieron con su mismo valor una vez finalizada la función y de regreso en el `main`.

Parámetros por referencia

El paso de parámetros por referencia implica utilizar los operadores `&` y `*`. El operador `&` se antepone a una variable, dando con ello acceso a su dirección

de memoria asignada. Se utiliza en los argumentos para pasarle por referencia dicha variable. El ejemplo clásico de una función de este tipo es *scanf*, donde los valores introducidos por medio del teclado son almacenados por referencia en la variable o variables indicadas; cuando se utiliza la función, se le llama con el operador & antes del identificador de cada variable.

```
scanf("%i%i%i", &a, &b, &c);
```

Luego de recibir las variables argumento, lo que se almacena en ellas permanece incluso después de finalizada la función, dado que pasaron por referencia.

El operador * es un apuntador que "apunta" a la dirección de la variable pasada como argumento. Se utiliza tanto en la declaración de los parámetros formales de la función como en el cuerpo de la misma. Debe aparecer antes del nombre de un parámetro formal en la cabecera para indicar que dicho parámetro será pasado por referencia, y debe aparecer en el cuerpo de la función antepuesto al nombre de un parámetro formal para acceder al valor de la variable externa a la función y referenciada por el parámetro formal.

Observe el siguiente ejemplo, que es una versión del programa anterior. Aquí se utiliza el paso por referencia, lo cual luego de la ejecución presenta resultados diferentes a los de la versión con paso de parámetros por valor:

EJEMPLO 8.10 Realizar una función que intercambie dos valores entre *s*, utilizando parámetros por referencia

```
#include<stdio.h>

void intercambio(int *x, int *y)
{
 int aux;
 aux=*x;
 *x=*y;
 *y=aux;
}

void main()
{
 int n1,n2;
```

(continúa)

(*continuación*)

```
    scanf("%d %d", &n1,&n2);
    intercambio(&n1,&n2);
    printf("Los nuevos valores son n1 = %d y n2 = %d", n1,n2);
}
```

La ejecución del programa es igual al programa anterior: inicia en el `main` solicitando dos valores que se almacenan en `n1` y `n2`. Suponga que el usuario teclea 5 y 8. Estas variables son "entregadas" a la función `intercambio`, y como pasan por referencia, de ese punto en adelante lo que suceda con `x` y `y` dentro de la función afectará directamente a los argumentos —es decir, a `n1` y `n2`—, por lo que al finalizar la función el mensaje que se mostrará en pantalla será:

```
Los nuevos valores son n1 = 8 y n2 = 5
```

Observe que los valores se invirtieron: ahora `n1` aparece con 8 y `n2` con 5. Otra consideración que se debe hacer es que una función puede recibir ambos tipos de parámetros. En seguida se presenta un ejemplo de función que utiliza el primer parámetro por valor y el segundo por referencia, lo que implica que cualquier valor que pase a la función a través de `a` no podrá ser modificado, mientras que lo que suceda con `b` dentro de la función afectará directamente al argumento que se coloque en su lugar.

EJEMPLO 8.11 Realizar una función que intercambie dos valores entre números, utilizando un parámetro por valor y otro por referencia

```
void func(int a, in *b)
{
     a=a+5;
    *b= *b+2;
}

void main()
{
int x,y;
scanf("%d %d", &x,&y);
func(x,&y);
printf("El valor de x es de %d y el de y es de %d", x,y);
}
```

La función *main* solicita dos números al usuario, los cuales recibe en *x* y *y* respectivamente. Suponga que los valores son 3 y 7; *main* los pasa como argumentos a la función *func*, que los recibe a través de sus parámetros *a* y *b*, de tal modo que *a* recibe sólo el valor de 3, pero *b* recibe la dirección del argumento y *func* le suma 5 a *a* y también suma 2 a *b*. Se termina la función y se regresa al *main* a continuar con la siguiente línea, que consiste en imprimir el mensaje; quedaría de la siguiente manera:

```
El valor de x es de 3 y el de y es de 9
```

Como *x* entró por valor no se afectó su dato original; en cambio *y* entró por referencia y cualquier modificación al parámetro *b* le afecta. Por ello, al concluir la función, e incluso fuera de ella, *y* terminó con 9.

Ejercicios resueltos

EJERCICIO 8.1 Realizar un programa que permita al usuario escoger una operación: el factorial de un número, la potencia indicada de un número o imprimir una tabla de multiplicar

Función *main*

Descripción

Invocar la función *menu*.

Invocar la función según la opción elegida.

Invocar una función de acuerdo con la elección.

Continuar mientras el usuario así lo decida.

Variables		
Nombre	*Tipo*	*Uso*
opc	*Entero*	*Para almacenar el valor de la función deseada.*

Realiza las invocaciones a las funciones, se inicia llamando la función *menu* y recibe el número de la función que se desea calcular.

(*continúa*)

(*continuación*)

Función *menu*

Descripción

Presentar las cuatro opciones que el usuario puede elegir.

Leer el valor de la opción.

Repetir las dos acciones anteriores mientras no se elija un valor entre 1 y 4, inclusive.

Regresar a la función `main` el valor elegido.

Variables		
Nombre	*Tipo*	*Uso*
`Op`	`Entero`	`Almacena la operación deseada.`

Función *factorial*

Descripción

Solicitar un número.

Leer el número.

Calcular el factorial multiplicando el número por todos sus antecesores hasta 1.

Imprimir el factorial.

Variables		
Nombre	*Tipo*	*Uso*
`num`	`Entero`	`Almacena el cálculo del factorial.`

Función *potencia*

Descripción

Solicitar la base y el exponente.

Leer base y exponente.

Calcular la potencia multiplicando la base, tantas veces como lo indique el exponente.

Imprimir la potencia.

Variables		
Nombre	*Tipo*	*Uso*
`b`	`Entero`	`Almacena el factorial.`
`P`	`Entero`	`Almacena la base de la potencia.`
`tmp`	`Entero largo`	`Acumulador de multiplicaciones de la base por sí misma.`
`i`	`Entero`	`Variable de control que cuenta la repetición.`

Función *tabla*

Descripción

Solicitar el número de la tabla y el límite.

Leer el número y el límite.

Imprimir la tabla desde 1 hasta el límite.

Preguntar si calcula otra tabla de multiplicar o regresa al menú principal.

Leer la opción.

Repetir la tabla de multiplicar mientras el usuario así lo determine.

Variables		
Nombre	*Tipo*	*Uso*
`n`	`Entero`	`Tabla de multiplicar.`
`limit`	`Entero`	`Indica hasta qué número se mostrará la tabla de multiplicar.`
`i`	`Entero`	`Variable de control que toma los valores desde 1 hasta limit.`

Codificación

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

void factorial()
{
```

(continúa)

(*continuación*)

```
    int num;
    double fact;
    clrscr();
    printf("\n\n\tFactorial\n");
    printf("\tIntroduce un numero: ");
    scanf("%d", &num);
    fact=1;
    for(;num>0;num--)
      fact *= num;
    printf("\n\tel resultado es: %.0lf\n\n",fact);
}

void potencia()
{
    int b,p,i;
    double tmp;
    clrscr();
    printf("\n\n\tPotencia\n");
    printf("\tIntroduce la base: ");
    scanf("%d", &b);
    printf("\n\tIntroduce la potencia: ");
    scanf("%d", &p);
    tmp=b;
    for(i=1; i<p; i++)
      tmp*=b;
    printf("\n\tLa potencia de %d a la %d es: %.0lf.\n\n", b,p, tmp);
}

void tabla()
{
    int n,limit,i;
    clrscr();
    printf("\n\n\tTabla de multiplicar\n");
    printf("\tDe que numero es la tabla: ");
    scanf("%d", &n);
    printf("\tHasta que numero mostrara: ");
    scanf("%d", &limit);
    for(i=1; i<=limit; i++)
```

```
     printf("\n\n\t%d x %d = %d\n",i, n, i*n);
}

int menu()
{
    int op=0;
    clrscr();
    printf("\n\n\t M e n u.\n");
    printf("\n\t1) Factorial.\n");
    printf("\n\t2) Potencia.\n");
    printf("\n\t3) Tabla de multiplicar.\n");
    printf("\n\t4) Salir.\n");
    while(op<1 || op >4)
    {
       printf(" \n\t  > ");
       scanf("%d", &op);
    }
    return op;
}

int main()
{
    int opc = 0;

    while(opc!=4)
        {
        opc =menu();
 if(opc == 1)
    {
    factorial();
    }
 else if(opc == 2)
 {
 potencia();
 }
       else if (opc == 3)
              {
              tabla();
```

(continúa)

(*continuación*)

```
                }
        getch();

        }
    printf ("\n\n\n\t adios ");
    getch();
    return 0;
}
```

Explicación

La ejecución del programa se inicia en la función principal con la invocación de la función `menu`. El control va a esta función y se muestran cuatro opciones a escoger por el usuario; después de elegir una, el valor correspondiente se devuelve al programa principal. En caso de que el usuario teclee un valor que no esté entre 1 y 4, el menú se repetirá.

Una vez elegida la opción, se invocará a la función correspondiente o se terminará la ejecución del programa si se eligió 4. En caso de llamarse a una función, ésta se ejecuta; al finalizar, se lleva el control a la función principal, se invoca la función `menu` y se repiten los pasos.

En caso de elegir el número 4, se imprime la palabra "adios" y termina la ejecución del programa.

Ejecución

```
      M  e n u.

Factorial.

Potencia.


Tabla de multiplicar
Salir.

>1
```

```
- - - - - - - - - - - - - - - - - -
Factorial
Introduce un número: 4
El resultado es: 24

- - - - - - - - - - - - - - - - - -
        M  e n u.

Factorial.

Potencia.

Tabla de multiplicar

Salir.

4

Adios.
```

EJERCICIO 8.2 Escribir un programa con una función que realice las siguientes operaciones con un número: raíz cuadrada, cuadrado y cubo. Mostrar el resultado en forma de tabla, con los números del 1 al 10

Descripción

Generar los números del 1 al 10.

Invocar la función con cada uno de los números del 1 al 10.

Imprimir el resultado.

Tabla de parámetros

Nombre	*Tipo*	*Uso*
`n`	`Entero`	`Parámetro por valor para los valores del uno al diez.`
`raiz`	`Real`	`Parámetro por referencia para cálculo de potencia ½.`

(*continúa*)

(*continuación*)

Tabla de parámetros		
Nombre	*Tipo*	*Uso*
cuad	Entero	Parámetro por referencia para cálculo de potencia 2.
cubo	Entero	Parámetro por referencia para cálculo de potencia 3.

Codificación

```
/*Funciones con parámetros por valor*/
Mostrar una tabla con los números del 1 al diez con sus respectivas
potencias ½, 2  y  3  (raiz cuadrada, cuadrado y cubo). */
/*****************************************

#include <stdio.h>
#include <conio.h>
#include <math.h>

void potencias(int n, float *raiz, int *cuad, int *cubo)
{
  *raiz=sqrt(n);
  *cuad=n*n;
  *cubo=pow(n,3);
}

main()
{
  int i, cuad, cubo;
  float raiz;
  clrscr();
  printf("\n\t numero\t raiz\tcuadrado  cubo\n");
  for (i=1;i<=10;i++)
   {
     potencias(i,&raiz,&cuad,&cubo);
     printf("\t%6d %6.2f %6d %6d\n",i,raiz,cuad,cubo);
   }
  getch();
  return 0;
}
```

Explicación

Se muestra el encabezado de la tabla y, con el ciclo `for`, se generan los números del 1 al 10 (que es el número de ocasiones a llamar la función `potencias`). Este ciclo contiene dos instrucciones. La primera llama la función, con el símbolo `&` antepuesto a los parámetros de referencia (que en ese momento pueden tener cualquier valor). Al ejecutarse potencias se realizan las operaciones respectivas y se almacenan en la dirección de memoria correspondiente a cada parámetro. La siguiente instrucción muestra el valor de las variables, que son la raíz cuadrada, el cuadrado y el cubo de cada número.

Se definió la función potencias con cuatro parámetros: `i`, que es un parámetro por valor; `raiz`, `cuad` y `cubo` que son parámetros por referencia. Al ser invocada la `i` tiene un número y el resto llevan la dirección de memoria en que se encuentran.

La primera ocasión que se invoca la función `potencias`, `i` lleva el valor de 1 y los demás parámetros tienen almacenada basura hasta que se ejecuta la operación y almacenan el resultado, que al ser parámetro por referencia se almacena en la dirección que tiene cada variable.

Ejecución

```
numero          raiz          cuadrado          cubo
 1              1.00             1               1
 2              1.41             4               8
 3              1.73             9              27
 4              2.00            16              64
 5              2.24            25             125
 6              2.45            36             216
 7              2.65            49             343
 8              2.83            64             512
 9              3.00            81             729
10              3.16           100            1000
```

Resumen

En este capítulo se analizó la programación modular, una técnica que consiste en resolver por separado un problema, dando origen a la creación de módulos (pequeños programas llamados funciones).

Cada módulo o función se diseña, se codifica y se procesa de manera independiente.

El uso de funciones permite:

Reducir la complejidad del programa ("divide y vencerás").
Eliminar código duplicado.
Controlar fácilmente los efectos de los cambios.
Ocultar detalles de implementación.
Reutilizar el código.
Facilitar la legibilidad del código.

Hay funciones de dos tipos: las internas o predefinidas, y las externas o diseñadas por el programador.

Independientemente de todas las funciones, `main` debe existir siempre en un programa en C, dado que es la función principal y sólo a partir de ésta se podrá hacer uso de las demás.

Las funciones pueden ser diseñadas de diferentes maneras:

Sin paso de parámetros.
Con paso de parámetros.
Que regresen valor.
Que no regresen valor.
Combinando las anteriores.

Un programa puede tener variables locales o globales. Variable local es aquella que se declara dentro de cualquier función y puede ser utilizada sólo por dicha función. Las variables globales se declaran fuera de cualquier función y, según donde se declaren, varias funciones pueden tener acceso a ellas.

Una función puede tener paso de parámetros por valor o por referencia. Los parámetros por valor son aquellos en los cuales la función recibe sólo una copia del contenido de los argumentos. Los parámetros por referencia son aquellos a través de los cuales la función recibe la dirección en memoria de los argumentos, haciendo posible la modificación de estos datos dentro de la función y manteniendo los cambios al terminar la ejecución de la función mencionada.

Evaluación

I. Conteste las siguientes preguntas:

1. Técnica que consiste en dividir un problema en problemas más pequeños para resolverlos por separado _____.

2. Resolver un problema de lo general a lo particular significa realizar un diseño _____.

3. Conjunto de instrucciones que se agrupan bajo un mismo nombre _____.

4. Mencione tres ventajas que el uso de funciones ofrece _____, _____ y _____.

5. ¿El uso de la función `main` es opcional? _____.

6. Funciones que no requieren información de su entorno y para su ejecución basta con su llamada: _____.

7. Funciones que como resultado de su ejecución generan un valor entregándolo a su entorno: _____.

8. A las variables que se declaran dentro de una función (y que por tanto pueden ser utilizadas sólo por esa función) se les conoce como _____.

9. ¿Qué es una variable global? _____.

10. ¿Cuál de los siguientes sería el prototipo de una función con paso de parámetros por valor que no regresa valor?

a) `int f1 (int x);` b) `void f1 ( );` c) `void f1 (int x);`

11. ¿Cuál de los siguientes sería el prototipo de una función sin paso de parámetros que no regresa valor?

a) `int f1 (int x);` b) `void f1 ( );` c) `void f1 (int x);`

12. ¿Cuál de los siguientes sería el prototipo de una función con paso de parámetros y que regresa valor?

a) `int f1 (int x);` b) `void f1 ( );` c) `void f1 (int x);`

13. Las funciones pueden ser diseñadas por _____
o pueden ser _____.

14. Explique la diferencia entre parámetro y argumento.

_____

_____

15. ¿Una constante o una expresión aritmética podrían ser parámetros formales?
Sí ( ) No ( )

16. Los parámetros por _____ reciben una copia de lo que valen los argumentos.

17. Los parámetros por _____ reciben la dirección en memoria de los argumentos.

II. Realice el siguiente ejercicio.

Realice una función que permita almacenar un símbolo. En otra función defina las coordenadas de un recuadro y codifique otra que permita hacer un recuadro con dicho símbolo. Codifique el programa principal que invoca la primera función.

Ejercicios propuestos

1. Realizar una función que muestre los datos del usuario: nombre, dirección, edad, número de hermanos, aficiones.
2. Realizar cuatro funciones para calcular el cuadrado de un número: una de ellas sin parámetros, otra que devuelva el resultado a la función principal, otra con parámetros por valor y que devuelva el resultado a la función principal, y finalmente otra con parámetros: uno de valor para el número y otro por referencia para el resultado.
3. Realizar una función que lea hasta 10 valores para un arreglo, enviándolo como parámetro, y calcule en otra función el promedio, el valor mayor, el valor menor y los muestre en la función principal.
4. Realizar una función que dibuje en la pantalla un marco con asteriscos.

CAPÍTULO 9

Ejercicios resueltos

A continuación se muestran varios ejercicios codificados en lenguaje C, organizados por estructura de control y llevan una breve descripción. Algunos ejercicios son aplicaciones de casos de carreras de ingeniería, para promover el interés de los estudiantes.

9.1 Secuenciación

Ejercicios que muestran el uso de la estructura de control secuenciación.

9.1.1 Calcular el área de una balastra

```
/*Cálculo del volumen de una balastra*/
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
 textbackground(YELLOW);
 textcolor(BLUE);
 clrscr();
 float b,h,l,v;
 printf("Calcular el volumen de la balastra de 240 volts\n");
 printf("Dame la base\n");
 scanf("%f", & b);
 printf("Dame la altura\n");
 scanf("%f", & h);
 printf("Dame la longitud\n");
 scanf("%f", & l);
 v=b*h*l;
 printf("el volumen de la balastra es %.2f",v);
 getch();
}
```

9.1.2 Calcular el pago a realizar por número de apagadores y contactos

```
/*Precio por salida eléctrica*/
#include<stdio.h>
#include<conio.h>
#define salida 100
void main()
{
 textbackground(YELLOW);
 textcolor(RED);
 clrscr();
 float precio,c;
 printf("Precio por salida electrica\n");
 printf("Dame la cantidad de salidas tomando en cuenta que cada apagador y
    contacto es una salida\n");
 scanf("%f", & c);
 precio=salida*c;
 printf("el precio por las salidas es % f", precio);
 getch();
}
```

9.1.3 Calcular las coordenadas del vértice de una parábola

```
/*Programa para obtener las coordenadas del vértice de una parábola*/
 #include<conio.h>
 #include<stdio.h>
 void main()
 {
  clrscr();
  float a,b,c,x,y;
  printf("Programa para calcular el vertice de una parabola\n");
  printf("Dame el coeficiente del termino cuadratico ");
  scanf("%f",&a);
  printf("Dame el coeficiente del termino lineal ");
  scanf("%f",&b);
  printf("Dame el termino independiente ");
  scanf("%f",&c);
  x=-b/(2*a);
  y=((4*a*c)-(b*b))/(4*a);
  printf("El vertice esta en la coordenada (%5.2f,%5.2f)",x,y);
  getch();
 }
```

9.1.4 Calcular la medida de los ángulos complementario y suplementario, dado el valor de un ángulo

```
/* Ángulos complementarios y suplementarios */
 #include<stdio.h>
 #include<conio.h>
 void main()
 {
  clrscr();
  float ang,ang_sup,ang_com;
  printf("Angulos complementarios y suplementarios\n");
  printf("Dame la medida de los angulos en grados decimales ");
  scanf("%f",&ang);
  ang_com=90-ang;
  ang_sup=180-ang;
  printf("Su angulo complementario es %.2f\n",ang_com);
```

```
 printf("Su angulo suplementario es %.2f\n",ang_sup);
 getch();
}
```

9.1.5 Calcular la magnitud de un vector dados sus componentes

```
/* Magnitud de vectores en el espacio */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
 clrscr();
 float c1,c2,c3,mag;

 printf(" Magnitud de vectores en el espacio\n");
 printf(" Dame la 1era. componente ");
 scanf("%f",&c1);
 printf(" Dame la 2da.  componente ");
 scanf("%f",&c2);
 printf(" Dame la 3era. componente ");
 scanf("%f",&c3);
 mag=sqrt(c1*c1+c2*c2+c3*c3);
 printf(" Su magnitud es igual a %.2f",mag);
 getch();
}
```

9.1.6 Calcular las ppm (partes por millón) en una solución

```
/*Programa para calcular las ppm de una solución*/
#include<stdio.h>
#include<conio.h>
void main()
{
 textcolor(RED);
```

```
textbackground(BLACK);
clrscr();
float ppm,soluto,solvente;
gotoxy(18,2);
printf("Programa para calcular las partes por millon en una solucion\n");
printf("\n La masa del soluto y del solvente deben estar en las mismas
   unidades\n");
printf("\n Introduce la masa del soluto\n");
scanf("%f",&soluto);
printf("\n Ahora la masa del solvente\n");
scanf("%f",&solvente);
ppm=soluto/solvente*1000000;
printf("\n %f ppm",ppm);
getch();
}
```

9.1.7 Calcular el porcentaje de masa

```
/*Calcular el porcentaje en masa*/
#include<stdio.h>
#include<conio.h>
void main()
{
 textcolor(RED);
 textbackground(BLACK);
 clrscr();
 float pm,comp,solucion;
 gotoxy(32,2);
 printf("Calula el % en masa\n");
 printf("\n Introduce la masa del componente a calcular\n");
 scanf("%f",&comp);
 printf("\n Ahora la masa total de la solucion -Recuerda que las masas
    tienen que estar en las mismas unidades-\n");
 scanf("%f",&solucion);
 pm=comp/solucion*100;
 printf("\n %f porciento en masa",pm);
 getch();
}
```

9.1.8 Calcular el número de ladrillos que se necesitan

```
/*Número_de_ladrillos*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int mc,nl;
 clrscr();
 printf("Te indicare el numero de ladrillos que necesitas\ndime
    cuantos metros cuadrados construiras");
 scanf("%d",&mc);
 nl=mc*72;
 printf("El numero de ladrillos que necesitas es%d",nl);
 getch();
}
```

9.1.9 Calcular el número de escalones de una distancia

```
/*Escalones*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int esc,d;
 clrscr();
 printf("Dame la distancia horizontal y te dire el numero de
    escalones\ndame la  distancia en cm");
 scanf("%d",&d);
 esc=d/30;
 printf("El numero de escalones es%d",esc);
 getch();
}
```

9.1.10 Separa un número de cuatro dígitos en millares, centenas, decenas y unidades

```
/* Separa*/
#include<stdio.h>
```

```
#include<conio.h>
void main()
{
 int num;
 textbackground(54);
 textcolor(15);
 clrscr();
 gotoxy(15,5);printf("Introduce una cantidad entera de 4 digitos
    como maximo ");
 scanf("%d",&num);
 printf("\n\n\nEste numero tiene:");
 printf("\n\n%d unidades de millar",num/1000);
 num = num%1000;
 printf("\n%d centenas",num/100);
 num=num%100;
 printf("\n%d decenas",num/10);
 num=num%10;
 printf("\n%d unidades",num/1);
 num=num%1;
 getch();
}
```

9.1.11 Calcular el campo eléctrico

```
/*Campo eléctrico*/
#include <conio.h>
#include <stdio.h>
void main ()
{
clrscr ();
float e, f, q;
printf ("El programa calcula el campo electrico\n");
printf ("Dame la fuerza\n");
scanf ("%f", & f);
printf ("Dame la carga");
scanf ("%f", & q);
e=f/q;
printf ("El campo electrico es %6.2f", e);
getch ();
}
```

9.1.12 Calcular la resistencia

```
/*Resistencia*/
#include <conio.h>
#include <stdio.h>
void main ()
{
 float v, a, r;
 clrscr ();
 printf ("El programa calcula la resistencia\n");
 printf ("Dame el voltaje\n");
 scanf ("%f", & v);
 printf ("Dame el amperaje\n");
 scanf ("%f", & a);
 r=v/a;
 printf ("la resistencia es %6.2fomhs", r);
 getch ();
}
```

9.2 Selectiva simple

9.2.1 Calcular el coeficiente de variación

```
/* Programa para calcular el coeficiente de variación */
 #include<stdio.h>
 #include<conio.h>
 void main()
 {
  clrscr();
  float S,X,CV;
  int op;
  printf("Programa para calcular el coeficiente de variacion\n");
  printf(" 1.Muestra\n 2.Poblacion\n");
  printf("Elige una opcion ");
  scanf("%d",&op);
  printf("Dame la media ");
```

```
 scanf("%f",&X);
 printf("Dame la desviacion estandar ");
 scanf("%f",&S);
 CV=S/X*100;
  if (op==1) printf("El coeficiente de variacion muestral es
   %.2f%\n",CV);
  if (op==2) printf("El coeficiente de variacion poblacional es
   %.2f%\n",CV);
  if ((op!=1) && (op!=2)) printf("Tu opcion no esta en el menu");
 getch();
}
```

9.2.2 Calcular el producto punto de un vector

```
/* Producto punto de vectores en R2 y R3 */

#include<conio.h>
#include<stdio.h>
#include<math.h>
void main()
{
 clrscr();
 float a1,a2,a3,b1,b2,b3,prod;
 int op;
 printf("Producto punto de vectores en R2 y R3\n");
 printf(" 1.En R2\n 2.En R3\n");
 printf("Elige una opcion ");
 scanf("%d",&op);
 if (op==1)
 {
   printf("Nota: Usa un espacio entre cada numero\n");
   printf("Dame el primer vector ");
   scanf("%f%f",&a1,&a2);
   printf("Dame el segundo vector ");
   scanf("%f%f",&b1,&b2);
   prod=sqrt(a1*b1+a2*b2);
  printf("El producto punto es %.2f",prod);
```

```
    }
    if (op==2)
    {
       printf("Nota: Usa un espacio entre cada numero\n");
       printf("Dame el primer vector ");
       scanf("%f%f%f",&a1,&a2,&a3);
       printf("Dame el segundo vector ");
       scanf("%f%f%f",&b1,&b2,&b3);
       prod=sqrt(a1*b1+a2*b2+a3*b3);
       printf("El producto punto es %.2f",prod);
     }
    getch();
}
```

9.2.3 Calcular la pendiente de una recta

```
/* Pendiente de una recta dados dos puntos */

#include<conio.h>
#include<stdio.h>
void main()
{
 clrscr();
 float x1,x2,y1,y2,m,X;
 printf("Pendiente de una recta dados dos puntos\n");
 printf("Primera coordenada\n");
 printf("Dame la X de la 1era. coordenada ");
 scanf("%f",&x1);
 printf("Dame la Y de la 1era. coordenada ");
 scanf("%f",&y1);
 printf("Segunda coordenada\n");
 printf("Dame la X de la 2da. coordenada ");
 scanf("%f",&x2);
 printf("Dame la Y de la 2da. coordenada ");
 scanf("%f",&y2);
 X=x1-x2;
 if (X!=0)
 {
```

```
    m=(y1-y2)/X;
    printf("La pendiente de la recta es %.2f",m);
   }
  if ((X==0) && (y1!=y2)) printf("La recta es vertical \"pendiente
    indefinida\"");
  if((X==0) && (y1==y2)) printf("Es un punto \"no tiene pendiente\"");
  getch();
 }
```

9.2.4 Calcular la fórmula VENA

```
/*Fórmula "vena"*/
#include<stdio.h>
#include<conio.h>
void main()
{
 textcolor(RED);
 textbackground(BLACK);
 clrscr();
 float V,E,N,A;
 char op;
 gotoxy(32,2);
 printf("Uso de la formula VENA\n Indique que desea calcular:\n");
 printf("A)Volumen   B)Equivalentes quimicos\n");
 printf("C)Normalidad   D)Gramos de sustancia\n");
 scanf("%c",&op);
 if(op=='A')
 {
   printf("\n Dame los gramos del soluto\n");
   scanf("%f",&A);
   printf("\n Ahora los equivalentes quimicos\n");
   scanf("%f",&E);
   printf("\n Por ultimo la normalidad\n");
   scanf("%f",&N);
   V=A/(E*N);
   printf("\n%f L",V);
  }
 if (op=='B')
```

```
{
  printf("\n Dame los gramos del soluto\n");
  scanf("%f",&A);
  printf("\n Ahora el volumen\n");
  scanf("%f",&V);
  printf("\n Por ultimo la normalidad\n");
  scanf("%f",&N);
  E=A/(V*N);
  printf("\n%f Equivalentes quimicos",E);
 }
if (op=='C')
{
  printf("\n Dame los gramos del soluto\n");
  scanf("%f",&A);
  printf("\n Ahora los equivalentes quimicos\n");
  scanf("%f",&E);
  printf("\n Por ultimo el volumen\n");
  scanf("%f",&V);
   N=A/(E*V);
   printf("\n%f EQ/L",N);
  }
 if (op=='D')
 {
   printf("\n Dame el volumen\n");
   scanf("%f",&V);
   printf("\n Ahora los equivalentes quimicos\n");
   scanf("%f",&E);
   printf("\n Por ultimo la normalidad\n");
  scanf("%f",&N);
  A=V*E*N;
  printf("\n%f L",A);
 }
getch();
}
```

9.2.5 Calcular las ecuaciones básicas del gas

```
/*Fórmula PV=nRT*/
#include<stdio.h>
```

```
#include<conio.h>
#define R 0.0821
void main()
{
 textcolor(RED);
 textbackground(BLACK);
 clrscr();
 float P,V,n,T;
 char op;
 gotoxy(33,2);
 printf("Ecuacion del gas ideal\nQue desea calcular?\n");
 printf("\nA)Presion   B)Volumen   C)Moles   D)Temperatura\n");
 scanf("%c",&op);
 if (op=='A')
 {
   printf("\n Dame los moles\n");
   scanf("%f",&n);
   printf("\n Ahora la temperatura (en K)\n");
   scanf("%f",&T);
   printf("\n Finalmente el volumen (en L)\n");
   scanf("%f",&V);
   P=n*R*T/V;
   printf("\n%f atm",P);
  }
if (op=='B')
{
   printf("\n Dame los moles\n");
   scanf("%f",&n);
   printf("\n Ahora la temperatura (en K)\n");
   scanf("%f",&T);
   printf("\n Finalmente la presion (en atm)\n");
   scanf("%f",&P);
   V=(n*R*T)/P;
    printf("\n%f L",V);
  }
 if (op=='C')
 {
    printf("\n Dame la presion (en atm)\n");
    scanf("%f",&P);
```

```
    printf("\n Ahora la temperatura (en K)\n");
    scanf("%f",&T);
    printf("\n Finalmente el volumen (en L)\n");
    scanf("%f",&V);
    n=(P*V)/(R*T);
    printf("\n%f moles",n);
  }
if (op=='D')
{
  printf("\n Dame los moles\n");
  scanf("%f",&n);
  printf("\n Ahora la presion (en atm)\n");
  scanf("%f",&P);
  printf("\n Finalmente el volumen (en L)\n");
  scanf("%f",&V);
  T=(P*V)/(n*R);
  printf("\n%f K",T);
 }
 getch();
}
```

9.2.6 Calcular el costo indirecto de cada departamento de la compañía Good Mark

```
/*Good Mark Company*/
#include<stdio.h>
#include<conio.h>
#define mon 120000
void main()
{
   int mam,man,ins,dp,total;
   clrscr();
   printf("Calcula el costo indirecto total de cada departamento
       de la Good Mark Company\n");
   printf("Departamento del que necesitas saber el total de su costo
       indirecto\n");
   printf("1)departamento de corte\n2)departamento de impresion\n");
   scanf("%d",&dp);
```

```
    printf("Cuanto fue del costo de manejo de materiales\n");
    scanf("%d",&mam);
    printf("Cuanto se invirtio de manufactura\n");
    scanf("%d",&man);
    printf("Cuanto fue del costo de inspeccion\n");
    scanf("%d",&ins);
    if (dp==1)
    {
      total=mam+man+ins+mon;
      printf("El costo indirecto del departamento de corte es $%d",total);
     }
    if (dp==2)
    {
      total=(mam+man+ins+mon)-(.20*(mam+man+ins+mon));
      printf("El costo indirecto del departamento de impresion
          es $%d",total);
     }
getch();
}
```

9.2.7 Calcular la cantidad de piedra que se necesita para un cimiento

```
/*Cimientos*/
#include<stdio.h>
#include<conio.h>
void main()
{
  float m,c;
  int m3;
  clrscr();
  printf("Te indicare la cantidad de piedra que necesitas para tus
      cimientos, ya sean colindantes o centrales\nCuantos
      metros de cimientos?\n");
  scanf("%d",&m);
  printf(" El cimiento, es central o colindante?\ncolindante=1\
      ncentral=2\n");
  scanf("%d",&c);
```

```
 if (c==1)  m3=m*1.5;
 if (c==2)  m3=m*2;
 printf("Necesitas%f",m3,"m3 de piedra");
 getch();
}
```

9.2.8 Calcular el índice de masa muscular

```
/*Programa índice de masa corporal*/
#include<stdio.h>
#include<conio.h>
void main()
{
  float peso, estatura, indice;
  textbackground(5);
  textcolor(2);
  clrscr();
  gotoxy(5,5);
  printf("*-* Programa para calular el indice de masa muscular *-*");
  gotoxy(1,9);
  printf("Introduce tu peso (en kg): ");
  scanf("%f",&peso);
  gotoxy(1,11);
  printf("Introduce tu estatura (en metros): ");
  scanf("%f",&estatura);
  indice=peso/(estatura*estatura);
  gotoxy(5,15);
  printf("Tu indice de masa corporal es: %.2f", indice);
 if (indice>=30)  printf("\n\n\n\n\n Cuidado! tu indice de masa corporal es
     ELEVADO,  \n  necesitas hacer ejercicio");
 if (indice<=18.5) printf("\n\n\n\n\n Cuidado!! tu indice de masa corporal
     es demasiado BAJO  \n  necesitas alimentarte mas");
 if (18.5<indice>30) printf("\n\n\n\n\n Felicidades!! tu indice de masa
     corporal es NORMAL");
 getch();
}
```

9.2.9 Determinar el tipo de compuesto


```
/*alcano, alqueno o alquino*/
#include <stdio.h>
#include <conio.h>
void main ()
{
  int h, c, h1, h2, h3;
  clrscr ();
  printf ("El programa indica si el compuesto es alcano, alqueno o alquino
    segun el numero de hidrogenos\n");
  printf ("Dame el numero de carbonos\n");
  scanf ("%d", & c);
  printf ("Dame el numero de hidrogenos\n");
  scanf ("%d", & h);
  h1=2*c+2;
  h2=2*c;
  h3=2*c-2;
  if (h==h1) printf ("El compuesto es un alcano");
  if (h==h2) printf ("El compuesto es un alqueno");
  if (h==h3) printf ("el compuesto es un alquino");
  if (h!=h1 && h!=h2 && h!=h3) printf ("El compuesto no existe");
  getch ();
 }
```

9.2.10 Sensor óptico para encender una luz

```
/*Sensor*/
#include <stdio.h>
#include <conio.h>
void main ()
{
  int d;
  clrscr ();
  printf ("sensor optico\n");
  printf ("Dame la distancia a la que te encuentras, en metros\n");
  scanf ("%d", & d);
  if (d>0 && d<=10) printf ("El sensor entra en funcionamiento y se activa
       la luz");
```

```
 if (d>10) printf ("La luz no se encendera");
 getch ();
}
```

9.2.11 Temporizador de una represa

```
/*Temporizador*/
#include <stdio.h>
#include <conio.h>
void main ()
{
 clrscr ();
 float t, l;
 printf ("El programa analiza el temporizador de una represa de agua\n");
 printf ("El temporizador marca el tiempo que la compuerta permanecera
      abierta\n");
 printf ("Cada segundo entran 500 L de agua\n");
 printf ("La represa tiene capacidad de solo 500,000 l.\n");
 printf ("Dame el temporizador\n");
 scanf ("%f", & t);
 l=t*60*500;
 printf ("Entra una cantidad de %6.2f litros\n", l);
 if (l>0 && l<=200000) printf ("La represa funciona bien");
 if (l>200000 && l<500000) printf ("La represa funciona pero no al 100 por
      ciento\n");
 if (l>500000) printf ("Emergencia sobrecarga! Debe cerrar inmediatamente
      las compuertas");
 getch ();
}
```

9.2.12 Descripción de un compuesto químico según sus componentes

```
/*Compuesto químico*/
#include <conio.h>
#include <stdio.h>
void main ()
```

```
{
 clrscr ();
 int H, O, c1, c2, C;
 printf ("Compuesto quimico\n");
 printf ("El programa indica el compuesto si unes carbono con oxigeno
      e hidrogeno o solo carbono e hidrogeno\n");
 printf ("Que componente deseas unir, 1)C 2)O, 3)H\n");
 scanf ("%d", & c1);
 printf ("Cual otro\n");
 scanf ("%d", & c2);
if (c1==1 && c2==3) printf ("El compuesto con componentes %d y
     %d es un hidrocarburo\n", c1, c2);
if (c1==2 && c2==3) printf ("El compuesto con componentes %d y
     %d es agua\n", c1, c2);
if (c1==3 && c2==2) printf ("El compuesto con componentes %d
     y %d es agua\n", c1, c2);
if (c1==2 && c2==1) printf ("El compuesto con componentes %d
     y %d es un oxido de carbono\n", c1, c2);
 if (c1==1 && c2==2) printf ("El compuesto con componentes %d
     y %d es un oxido de carbono\n", c1, c2);
 if (c1==3 && c2==1) printf ("El compuesto con componentes %d
     y %d es un hidrocarburo\n", c1, c2);
 getch ();
}
```

9.2.13 Calcular corriente, potencia y resistencia de un aparato eléctrico

```
/*Aparatos eléctricos*/
#include<conio.h>
#include<stdio.h>
#include<MATH.H>
void main()
{
  clrscr ();
  int op;
  float i, p, r;
  printf ("Aparatos electricos y algunos de sus calculos\n");
```

```
printf ("Se usa la corriente, la potencia y la resistencia\nseleccione
     el valor a calcular\n");
printf (" Seleccione su opcion\n1)potencia\n2)intensidad\n3)
  resistencia\n");
scanf  ("%d", &op);
if (op==1)
{
  printf ("Potencia \ndame la intensidad\n");
  scanf ("%f",&i);
  printf("Dame la resistencia del circuito\n");
  scanf ("%f",&r);
  p=i*i*r;
  printf ("La potencia del aparato es %7.2f \n", p);
 }

if (op==2)
{
   printf ("Intensidad\ndame la potencia\n");
   scanf ("%f",&p);
   printf ("Dame la resistencia\n");
   scanf ("%f", &r);
   i= sqrt(p/r);
   printf ("La intesidad del circuito es %7.3f \n", i);
  }

  if (op==3)
  {
    printf ("Resistencia\ndame la intensidad\n");
    scanf ("%f", &i);
    printf ("Dame la potencia\n");
    scanf ("%f", &p);
    r=p/(i*i);
    printf ("La resistencia es %7.2f \n",r);
  }

  if (op!=1&&op!=2&&op!=3) printf (" Error opcion invalida\n");
getch();
}
```

9.3 Selectiva doble

9.3.1 Calcular la cantidad de concreto requerido según la humedad

```
/*humedad_construccion*/
#include<stdio.h>
#include<conio.h>
void main()
{
  int m3,h,p;
  clrscr();
  printf("Te indicare el costo de tu pedido segun el concreto que necesitas
       y los m3 que desees\ncuantos m3 de concreto necesitas?\n");
  scanf("%d",&m3);
  printf("La humedad en el lugar de la construccion sobrepasa el 80%?\nsi
       es asi marca 1\nde lo contrario marca 2\n");
 scanf("%d",&h);
 if (h==1)
 {
  p=m3*700;
   printf("Si la humedad pasa el 80%, necesitas un concreto de 200kg/cm2
       y el precio de tu pedido es%d",p);
  }
else
{
  p=m3*500;
  printf("Si la humedad no pasa el 80%, necesitas un concreto de 150kg/cm2
       y el precio de tu pedido es%d",p);
  }
getch();
}
```

9.3.2 Sensor para encender una luz

```
/*Sensor*/
#include <stdio.h>
```

```
#include <conio.h>
void main ()
{
  int d;
  clrscr ();
  printf ("Sensor optico\n");
  printf ("Dame la distancia a la que te encuentras, en metros\n");
  scanf ("%d", & d);
  if (d>0 && d<=10) printf ("El sensor entra en funcionamiento y se activa
       la luz");
  else
        printf ("La luz no se encendera");
getch ();
}
```

9.3.3 Calcular resistencias en paralelo o en serie

```
/*Resistencias serie o paralelo*/
#include<conio.h>
#include<stdio.h>
void main()
{
  float r, rp, r1, r2, r3;
  int op;
  printf ("El programa permite el calculo de tres resistencias en paralelo
       o en serie\n");
  printf ("Seleccione el tipo de circuito\n");
  printf ("1)En serie\n2)En paralelo\n");
  scanf ("%d",&op);
 if (op==1)
 {
  printf ("En serie\ningrese el valor de la primera resistencia\n");
  scanf ("%f",&r1);
  printf ("Ingrese la segunda resistencia\n");
  scanf ("%f",&r2);
  printf ("Ingrese el valor de la tercer resistencia\n");
  scanf ("%f",&r3);
  r=r1+r2+r3;
```

```
  printf ("La resistencia total es %7.2fohms\n",r);
 }
else
{
  printf ("En paralelo\n");
  printf ("Ingrese la primera resitencia\n");
  scanf ("%f",&r1);
  printf ("Ingrese la segunda resitencia\n");
  scanf ("%f",&r2);
  printf ("Ingrese la tercera resistencia\n");
  scanf (" %f",&r3);
  rp=(1/r1)+(1/r2)+(1/r3);
  r=1/rp;
  printf ("La resistencia del total es %7.2fohms\n",r);
 }
getch();
}
```

9.3.4 Calcular el número de huevos que una viejecita lleva en su cesta

```
/*Compra*/
#include<stdio.h>
#include<conio.h>
void main()
{
  int a;
  textbackground(27);
  textcolor(0);
  clrscr();
  gotoxy(29,5);
  printf("La viejecita en el mercado");
  gotoxy(1,10);
  printf("Una viejecita llevaba huevos al mercado cuando\nse le cayo
       la cesta.");
  printf("\n\n-Cuantos huevos llevabas? -Le preguntaron,\n\n-No lo se,
    recuerdo que al contarlos en grupos\n");
  printf("de 2,3,4 y 5, sobraban 1,2,3 y 4\nrespectivamente.\n\n
```

```
 Cuantos huevos tenia la viejecita?");
printf("\n\n1) 20 huevos\n\n2)59 huevos\n\n3)1000 huevos\n\n(Seleccione
     la opcion correcta) ");
scanf("%d",&a);
if (a==2) printf("                                    Acertaste!");
else
      printf("                                   Fallaste!");
getch();
}
```

9.3.5 Juego de multiplicaciones

```
/*Juego*/
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void main()
{
 int a,b,c,d;
 textbackground(0);
 textcolor(3);
 clrscr();
 gotoxy(30,5);
 printf("***** Aprende jugando *****");
 gotoxy(1,8);
 printf("Hola amiguito(a)!!\n\n                    ....VaMoS A
     JuGaR!....");
 gotoxy(1,14);
 printf("Instrucciones:\n\nYo te ire mostrando multiplicaciones y tu
     tendras que responder\nde manera correcta para poder seguir
     jugando\n\n\n .... Estas listo? ....");
 gotoxy(35,45);
 printf("...presiona Enter para continuar...");
 getch();
 clrscr();
 gotoxy(30,5);
 printf("...Cuanto es?...");
```

```
srand(time(NULL));
a=rand()%10+1;
b=rand()%10+1;
c=a*b;
printf("\n\n%i X %i = ",a,b);
scanf("%i",&d);
if (c==d) printf("            Bien hecho");
else
     printf("            Eso NO es correcto");
a=rand()%10+1;
b=rand()%10+1;
c=a*b;
printf("\n\n%i X %i = ",a,b);
scanf("%i",&d);
if (c==d) printf("            Bien hecho");
else
     printf("            Eso NO es correcto");
a=rand()%10+1;
b=rand()%10+1;
c=a*b;
printf("\n\n%i X %i = ",a,b);
scanf("%i",&d);
if (c==d)  printf("            Bien hecho");
else
     printf("            Eso NO es correcto");
a=rand()%10+1;
b=rand()%10+1;
c=a*b;
printf("\n\n%i X %i = ",a,b);
scanf("%i",&d);
if (c==d) printf("            Bien hecho");
else
     printf("            Eso NO es correcto");
a=rand()%10+1;
b=rand()%10+1;
c=a*b;
printf("\n\n%i X %i = ",a,b);
scanf("%i",&d);
if (c==d) printf("            Bien hecho");
```

```
else
     printf("            Eso NO es correcto");
a=rand()%10+1;
b=rand()%10+1;
c=a*b;
printf("\n\n%i X %i = ",a,b);
scanf("%i",&d);
if (c==d) printf("            Bien hecho");
else
      printf("            Eso NO es correcto");
a=rand()%10+1;
b=rand()%10+1;
c=a*b;
printf("\n\n%i X %i = ",a,b);
scanf("%i",&d);
if (c==d) printf("            Bien hecho");
else
     printf("            Eso NO es correcto");
 getch();
}
```

9.3.6 Calcular el coeficiente de correlación

```
/* Calcula el coeficiente de correlación*/
 #include<conio.h>
 #include<stdio.h>
 void main()
 {
  clrscr();
  float Sxy,Sx,Sy,r;
  printf("Calcula el coeficiente de correlacion\n");
  printf("Dame la covarianza de los datos ");
  scanf("%f",&Sxy);
  printf("Dame la desviacion estandar de X ");
  scanf("%f",&Sx);
  printf("Dame la desviacion estandar de Y ");
  scanf("%f",&Sy);
  if ((Sx==0) || (Sy==0)) printf("Datos invalidos");
```

```
 else
 {
    r=Sxy/(Sx*Sy);
    printf("El coeficiente de correlacion es %.4f",r);
 }
getch();
 }
```

9.3.7 Indicar si un compuesto es soluble o no

```
/*Indica la solubilidad*/
#include<stdio.h>
#include<conio.h>
void main()
{
  textcolor(RED);
  textbackground(BLACK);
  clrscr();
  char cation;
  gotoxy(13,2);
  printf("Indica si un compuesto es o no soluble\n");
  printf("\nIMPORTANTE: Este programa usa como base las reglas
       de solubilidad de compuestos inorganicos en agua\n");
  printf("\n Elija el cation:\nN=Na\tK=K\tL=Li\tR=Rb\tC=Cs");
  scanf("%c\n",&cation);
  if ((cation=='N') || (cation=='K') || (cation=='L') || (cation=='R') ||
       (cation=='C')) printf("\n El compuesto es soluble");
  else
  printf("\n El compuesto es insoluble; sin embargo, si el anion es
       un nitrato, carbonato acido, clorato o perclorato, el compuesto
       es soluble");
  getch();
 }
```

9.3.8 Identificar semirreacciones, oxidación o reducción

```
/*Identifica rxn's redox*/
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
  textcolor(RED);
  textbackground(BLACK);
  clrscr();
  int v1,v2;
  gotoxy(16,2);
  printf("Identifica semirreacciones redox: Oxidacion o reduccion\n");
  printf("\n Recuerde que cualquier semirreaccion redox, ya sea de
       oxidacion o de reduccion, va acompania de su contraria\n");
  printf("\n Introduzca la valencia del elemento en el compuesto
       reactivo\n");
  scanf("%i",&v1);
  printf("\n Ahora la valencia del mismo elemento, pero ahora en el
       compuesto producto:\n");
  scanf("%i",&v2);
  if (v1<v2) printf("\n La semirreaccion es de oxidacion");
 else
  printf("\n La semirreaccion es de reduccion");
  getch();
}
```

9.3.9 Indicar el rendimiento teórico de una reacción

```
/*Rendimiento de rxn*/
#include<stdio.h>
#include<conio.h>
void main()
{
  textcolor(RED);
  textbackground(BLACK);
  clrscr();
  float porcentaje, rendt, rendp;
  gotoxy(23,2);
  printf("Rendimiento de una reaccion\n");
  printf("\n Indique el rendimiento teorico (se calcula
    estequiometricamente) de la reaccion\n");
  scanf("%f",&rendt);
```

```
 printf("\n Ahora el rendimiento practico (se observa experimentalmente)
      de la reaccion\n");
 scanf("%f",&rendp);
 if (rendt!=rendp)
 {
   porcentaje=(rendt/rendp)*100;
   printf("\n El rendimiento de la reaccion (en porcentaje) es:
      %f",porcentaje);
  }
 else
      printf("\n La reaccion se lleva a cabo al 100%");
 getch();
}
```

9.4 Selectiva doble anidada

9.4.1 Juego de adivinanza de números

```
/*Adivina*/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
void main()
{
  textbackground(BLACK);
  textcolor(9);
  clrscr();
  int usuario, yo;
  srand(time(NULL));
  yo = rand()%10+1;
  gotoxy(25,5);printf("Adivina adivinador");
  gotoxy(1,12);printf("Que numero entero entre 1 y 10 crees que estoy
       pensando? ");
  scanf("%d",&usuario);
  gotoxy(1,15);
  if (usuario==yo) printf("Felicidades, adivinaste mi numero");
  else
```

```
{
   if (usuario<yo) printf("\n\nLo siento, tu numero es MENOR que el
        mio");
   if (usuario>yo) printf("\n\nLo siento, tu numero es MAYOR que el
        mio");
   gotoxy(42,25);
   printf("... presiona enter para continuar ...");
   getch();
   clrscr();
   printf("\f\n\nEsta es tu segunda oportunidad ingresa otro
        numero: ");
   scanf("%d",&usuario);
   if(usuario==yo) printf("\n\nFelicidades, adivinaste mi numero");
   else
   {
     if(usuario<yo) printf("\n\nLo siento, tu numero es MENOR que el
          mio");
     if(usuario>yo) printf("\n\nLo siento, tu numero es MAYOR que el
          mio");
      gotoxy(42,25);
      printf("... presiona enter para continuar ...");
     getch();
     clrscr();
     printf("\n\nEsta es tu ultima oportunidad ingresa otro
          numero: ");
     scanf("%d",&usuario);
     if (usuario==yo) printf("\n\nFelicidades, adivinaste mi
          numero");
     else
     {
      if (usuario<yo) printf("\n\nLo siento, tu numero es MENOR que
          el mio");
      gotoxy(32,35);
      printf("El numero era: %d",yo);
      }
     }
    }
  getch();
}
```

9.4.2 Indicar el tipo de dato que el usuario introduzca


```
/*Tipo dato*/
#include<conio.h>
#include<stdio.h>
void main()
{
  char usuario;
  textcolor(0);
  textbackground(98);
  clrscr();
  gotoxy(15,5);
  printf("Programa que indica el tipo de dato introducido");
  printf("\n\n\nIntroduce tu dato (un digito) ");
  scanf("%s",&usuario);
  if ((57>=usuario) && (usuario>=48)) printf("\n\nEste es un numero");
   else
       if ((122>=usuario) && (usuario>=97) || (90>=usuario) &&
            (usuario>=65))   printf("\n\nEsta es una letra");
      else
    printf("\n\nEste es un caracter especial");
   getch();
 }
```

9.4.3 El número menor de cinco números

```
/* Menor*/
#include<conio.h>
#include<stdio.h>
void main()
{
  int a,b,c,d,e;
  textcolor(25);
  textbackground(0);
  clrscr();
  printf("Introduce cinco numeros(presiona enter despues de cada
       numero):\n");
  scanf("%i%i%i%i%i",&a,&b,&c,&d,&e);
```

```
if ((a<b) && (a<c) && (a<d) && (a<e)) printf("El numero menor es: %i",a);
else
{
   if((b<a) && (b<c) && (b<d) && (b<e)) printf("El numero menor es:
        %i",b);
   else
   {
      if ((c<a) && (c<b) && (c<d) && (c<e)) printf("El numero menor es:
         %i",c);
      else
      {
         if ((d<a) && (d<b) && (d<c) && (d<e))        printf("El numero
             menor es: %i",d);
         else
       printf("El numero menor es: %i",e);
      }
    }
  }
getch();
}
```

9.4.4 Indicar el tipo de ángulo introducido, según su medida

```
/* Tipos de ángulos */
 #include<conio.h>
 #include<stdio.h>
 void main()
 {
  clrscr();
  float med;
  printf("Tipos de angulos\n");
  printf("Dame la medida del angulo ");
  scanf("%f",&med);
  if (med<90) printf("Es un angulo agudo");
  else
       if (med==90) printf("Es un angulo recto");
       else
```

```
        if (med<180) printf("Es un angulo obtuso");
        else
          if (med==180) printf("Es un angulo llano");
      else
                if (med<360) printf("Es un angulo concavo");
                else
                     if (med==360) printf("Es un angulo perigono");
         else
                          printf("No tiene ningun nombre especifico");
 getch();
}
```

9.4.5 Calcular el coeficiente de variación

```
/* Calcula el coeficiente de variación */
 #include<stdio.h>
 #include<conio.h>
 void main()
 {
   clrscr();
   float S,X,CV;
   int op;
   printf("Calcula el coeficiente de variacion\n");
   printf(" 1.Muestra\n 2.Poblacion\n");
   printf("Elige una opcion ");
   scanf("%d",&op);
   if (op==1)
   {
     printf("Dame la media ");
     scanf("%f",&X);
     printf("Dame la desviacion estandar ");
     scanf("%f",&S);
     CV=S/X*100;
     printf("El coeficiente de variacion muestral es %.2f%\n",CV);
    }
    else
       if (op==2)
       {
```

```
        printf("Dame la media ");
        scanf("%f",&X);
        printf("Dame la desviacion estandar ");
        scanf("%f",&S);
        CV=S/X*100;
       printf("El coeficiente de variacion poblacional es %.2f%\n",CV);
      }
     else
         printf("Tu opcion no esta en el menu");
  getch();
}
```

9.4.6 Calcular la desviación de costos y materiales de la manufacturera Choice

```
/*Desviación de empresa choice*/
#include<stdio.h>
#include<conio.h>
#define pr 5.20
#define pe 5.00
#define tr 10.20
#define te 10.00
void main()
{
   int op;
   float cr,ce,hr,he,civa,civ,total,dpm,dum,dtm,demo;
   clrscr();
   printf("Desviacion de la manufacturera choice\n");
   printf("Que desviacion desea calcular?\n");
   printf("1)desviacion de materiales\n2)desviacion de mano de obra\n3)
         desviacion de costos indirectos\n");
   scanf("%d",&op);
   if (op==1)
   {
     printf("Dame el valor de la cantidad real\n");
     scanf("%f",&cr);
     dpm=cr*(pr-pe);
     printf("La desviacion en precio es %d",dpm);
```

```
  printf("\nDame el valor de la cantidad estandar\n");
  scanf("%f",&ce);
  dum=(cr-ce)*pe;
  printf("La desviacion en cantidad (uso) es %.2f",dum);
  total=dpm+dum;
  printf("\nEntonces la desviacion de materiales es $%.2f",total);
 }
else
    if (op==2)
    {
      printf("Dame la cantidad de horas reales\n");
      scanf("%f",&hr);
      dtm=hr*(tr-te);
      printf("La desviacion de tasa es $%.2f",dtm);
      printf("\nDame la cantidad de horas estandares\n");
      scanf("%f",&he);
      demo=te*(hr-he);
      printf("La desviacion de eficiencia es $%.2f",demo);
      total=(demo-dtm);
      printf("\nLa desviacion de mano de obra es $%.2f",total);
     }
    else
        if (op==3)
        {
          printf("Dame el total de costos indirectos variables\n");
          scanf("%f",&civ);
          printf("Dame el total de costos indirectos variables
              aplicados\n");
          scanf("%f",&civa);
          total=civ-civa;
          printf("La desviacion de costos indirectos es $%.2f",total);
        }
   getch();
}
```

9.4.7 Calcular las propiedades coligativas de soluciones acuosas

```
/*Propiedades coligativas*/
#include<stdio.h>
```

```
#include<conio.h>
#define R 0.0821
#define kb 0.58
#define kf 1.86
void main()
{
  textcolor(RED);
  textbackground(BLACK);
  clrscr();
  float pi,tb,tf,dtb,dtf,m,M,T,Pvapsln,Pvapsolv,Xsolv;
  char op;
  gotoxy(13,2);
  printf("*Propiedades coligativas\n De soluciones acuosas, no electolito
       no volatil (en el caso de la presion vapor)");
  printf("\n Que propiedad desea calcular? \nA)Temperatura de ebullicion\
       tB)Temperatura de congelacion\nC)Presion osmotica\tD)Presion
    vapor\n");
  scanf("%c",&op);
if (op=='A')
{
  printf("\n Dame la molaridad\n");
  scanf("%f",&m);
  dtb=kb*m;
  tb=100+dtb;
  printf("%f øC",tb);
 }
 else
 {
   if (op=='B')
   {
     printf("\n Dame la molaridad\n");
     scanf("%f",&m);
     dtf=kf*m;
     tf=0-dtf;
     printf("%f øC",tf);
   }
  else
  {
    if (op=='C')
     {
```

```
    printf("\n Dame la molaridad\n");
    scanf("%f",&M);
  printf("\n Ahora la temperatura, en K\n");
  scanf("%f",&T);
  pi=R*T*M;
  printf("%f atm",pi);
}
else
{
   if (op=='D')
   {
     printf("\n Dame la presion vapor del solvente\n");
    scanf("%f",&Pvapsolv);
    printf("\n Ahora la fraccion mol del solvente\n");
    scanf("%f",&Xsolv);
    Pvapsln=Pvapsolv*Xsolv;
    printf("%f atm",Pvapsln);
   }
        else
            printf("\n Opcion incorrecta");
       }
     }
   }
  getch();
}
```

9.5 Selectiva múltiple

9.5.1 Calcular la resistencia de un cable

```
/*Cálculo de la resistencia*/
#include<conio.h>
#include<stdio.h>
#define AU 1.06e-8
#define CO 1.72e-8
#define AL 3.21e-8
#define PL 11.05e-8
void main()
```

```
{
  clrscr();
  float l, a, res, r;
  int op;
  printf ("El programa permite calcular la resistencia de un cable\n");
  printf ("Selecione el material del cable\n");
  printf (" 1)plata\n2)cobre\n3)aluminio\n4)platino \n");
  scanf ("%d", &op);
  switch (op)
  {
    case 1: res=AU;
                printf ("El material es AU\n");
                break;
    case 2: res=CO;
                printf ("El material es CO\n");
                break;
    case 3: res=AL;
                printf ("El material es AL\n");
                break;
     case 4: res=PL;
                 printf ("El material es PL\n");
                 break;
     default:
                 printf ("Error conductor desconocido\n");
    }
  printf ("Dame la longitud del metal en metros\n");
  scanf ("%f",&l);
  printf ("Dame el area grosor del cable en m2.\n");
  scanf ("%f",&a);
  r=res*l/a;
  printf (" La resistencia es %g ohms\n", r);
  getch();
}
```

9.5.2 Calcular la magnitud de vectores

```
/* Magnitud de vectores en R2,R3,R4 */
 #include<conio.h>
 #include<stdio.h>
```

```
#include<math.h>
void main()
{
  clrscr();
  float a1,a2,a3,a4,mag;
  int op;
  printf("Calcula magnitud de vectores\n");
  printf(" 1)EN R2\n 2)EN R3\n 3)EN R4\n");
  printf("Elige una opcion ");
  scanf("%d",&op);
  printf("NOTA: Usa un espacio entre cada numero del vector\n");
  switch(op)
  {
    case 1: printf("Dame el vector en R2 ");
         scanf("%f%f",&a1,&a2);
         mag=sqrt(a1*a1+a2*a2);
          printf("La magnitud del vector en R2 es %.2f",mag);
          break;
    case 2: printf("Dame el vector en R3 ");
         scanf("%f%f%f",&a1,&a2,&a3);
         mag=sqrt(a1*a1+a2*a2*a3*a3);
         printf("la magnitud del vector en R; es %.2f",mag);
         break;
    case 3: printf("Dame el vector en R4 ");
         scanf("%f%f%f%f",&a1,&a2,&a3,&a4);
         mag=sqrt(a1*a1+a2*a2+a3*a3+a4*a4);
         printf("La magnitud del vector en R4 es %.2f",mag);
         break;
   default:
              printf("Opcion no valida");
  }
 getch();
}
```

9.5.3 Calcular las funciones básicas de un polígono regular

```
/* Elementos de un polígono regular*/
 #include<conio.h>
 #include<stdio.h>
```

```
void main()
{
  clrscr();
  int n,op;
  float resul;
  printf("Elementos de un poligono regular\n");
  printf(" 1.Angulo central\n 2.Angulo interno\n 3.Angulo externo\n 4.Num.
        de Diagonales en un V,rtice\n 5.Total de diagonales\n");
  printf("Elige lo que quieres calcular ");
  scanf("%d",&op);
  printf("Dame el numero de lados del poligono ");
  scanf("%d",&n);
  switch(op)
  {
     case 1: resul=360/n;
             printf("Cada angulo central mide %.2fø",resul);
             break;
     case 2: resul=90*(n-2)/n;
             printf("Cada angulo interno mide %.2fø",resul);
             break;
     case 3: resul=360/n;
             printf("Cada angulo externo mide %.2fø",resul);
             break;
     case 4: resul=n-3;
            printf("El numero de diagonales de cada v,rtice es %.0f",resul);
             break;
     case 5: resul=n*(n-3)/2;
             printf("El numero total de diagonales es %.0f",resul);
             break;
    default:
               printf("Opcion invalida");
   }
 getch();
}
```

9.5.4 Juego de piedra, papel o tijera

```
/*Programa juego piedra, papel o tijera*/
#include<stdio.h>
```

```
#include<conio.h>
#include<stdlib.h>
#include<time.h>
void main()
{
 textbackground(27);
 textcolor(32);
 clrscr();
 int usuario, yo;
 srand(time(NULL));
 gotoxy(25,5);
 printf("**** Juego piedra, papel o tijera ****");
 gotoxy(1,8);
 printf("Introduce tu jugada: \n\n1)Piedra\n2)Papel\n3)Tijera\n\n");
 scanf("%d",&usuario);
 yo = (rand()%3)+1;
 /*Piedra */
 gotoxy(25,20);
 if (usuario==1)
 {
   switch(yo)
   {
     case 1: printf("EMPATAMOS: piedra VS piedra");
                  break;
     case 2: printf("PERDISTE: piedra VS papel");
                  break;
     case 3: printf("GANASTE: piedra VS tijera");
                  break;
    }
  }
/* Papel */
 if (usuario==2)
 {
   switch(yo)
   {
     case 1: printf("GANASTE: papel VS piedra");
                  break;
     case 2: printf("EMPATAMOS: papel VS papel");
                  break;
```

```
        case 3: printf("PERDISTE: papel VS tijera");
                    break;
      }
    }
  /* Tijera */
  if (usuario==3)
  {
    switch(yo)
    {
      case 1: printf("PERDISTE: tijera VS piedra");
                    break;
      case 2: printf("GANASTE: tijera VS papel");
                    break;
      case 3: printf("EMPATAMOS: tijera VS tijera");
                     break;
      }
  }
  getch();
  }
```

9.5.5 Calcular las propiedades coligativas de una solución

```
/*Propiedades coligativas*/
#include<stdio.h>
#include<conio.h>
#define R 0.0821
#define kb 0.58
#define kf 1.86
void main()
{
  textcolor(RED);
  textbackground(BLACK);
  clrscr();
  float pi,tb,tf,dtb,dtf,m,M,T,Pvapsln,Pvapsolv,Xsolv;
  char op;
  gotoxy(13,2);
  printf("*Propiedades coligativas\n De soluciones acuosas, no electrolito
       no volatil (en el caso de la presion vapor)");
  printf("\n Que propiedad desea calcular? \nA)Temperatura de ebullicion\
```

```
    tB)Temperatura de congelacion\nC)Presion osmotica\tD)Presion vapor\n");
scanf("%c",&op);
switch(op)
{
 case 'A': printf("\n Dame la molaridad\n");
           scanf("%f",&m);
           dtb=kb*m;
           tb=100+dtb;
           printf("%f øC",tb);
           break;
 case 'B': printf("\n Dame la molaridad\n");
           scanf("%f",&m);
           dtf=kf*m;
           tf=0-dtf;
           printf("%f øC",tf);
            break;
 case 'C': printf("\n Dame la molaridad\n");
           scanf("%f",&M);
           printf("\n Ahora la temperatura, en K\n");
           scanf("%f",&T);
           pi=R*T*M;
           printf("%f atm",pi);
           break;
  case 'D': printf("\n Dame la presion vapor del solvente\n");
           scanf("%f",&Pvapsolv);
           printf("\n Ahora la fraccion mol del solvente\n");
           scanf("%f",&Xsolv);
           Pvapsln=Pvapsolv*Xsolv;
            printf("%f atm",Pvapsln);
           break;
  default:
           printf("\n Opcion incorrecta");
   }
 getch();
}
```

9.5.6 Calcular la solubilidad y sus variables

```
/*Solubilidad y variables*/
#include<stdio.h>
```

```
#include<conio.h>
void main()
{
  textcolor(RED);
  textbackground(BLACK);
  clrscr();
  float Sg,kh,Pg;
  char op;
  gotoxy(13,2);
  printf("Calculo de la solubilidad y sus variables\n");
  printf("\n Indique que se desea calcular:\nA)Solubilidad\tB)Constante de
    Henry\tC)Presion del gas\n");
  scanf("%c",&op);
  switch(op)
  {
   case 'A': printf("\n Dame la constante de Henry para el gas\n");
             scanf("%f",&kh);
             printf("\n Ahora la presion vapor del gas\n");
             scanf("%f",&Pg);
             Sg=kh*Pg;
             printf("%f n/L",Sg);
             break;
  case 'B': printf("\n Dame la solubilidad del gas\n");
             scanf("%f",&Sg);
             printf("\n Ahora la presion vapor del gas\n");
             scanf("%f",&Pg);
             kh=Sg/Pg;
             printf("%f n/L atm",kh);
             break;
  case 'C': printf("\n Dame la solubilidad del gas\n");
             scanf("%f",&Sg);
             printf("\n Ahora la constante de Henry para el gas\n");
             scanf("%f",&kh);
             Pg=Sg/kh;
             printf("%f atm",Pg);
             break;
   default:
        printf("\n Opcion incorrecta. Ejecute el programa de nuevo y elija
           una opcion correcta");
```

```
   }
 getch();
}
```

9.5.7 Calcular los costos unitarios de la empresa Gelstrap

```
/*Empresa Gelstrap*/
#include<stdio.h>
#include<conio.h>
void main()
{
  int op,md,mod,cin;
  float total;
  clrscr();
  printf("\nCalcule los costos unitarios de la empresa Gelstrap\n");
  printf("\nQue costo unitario deseas calcular\n");
  printf("1)Materiales directos\n2)Mano de obra\n3)Costos INDIRECTOS\n");
scanf("%d",&op);
switch(op)
{
  case 1: printf("Dame el costo de materiales directos\n");
          scanf("%d",&md);
          otal=md/9000;
          printf("El costo unitario de materiales directos es
             $%.3f",total);
          if (total<=2) printf("\nExcelente, el costo unitario de
             materiales ha disminuido lo que provoca mas produccion\n");
          else
             printf("\nHay que disminuir el costo de materiales, provoca
                 menos produccion\n");
          break;
   case 2: printf("Dame el costo de mano de obra directa\n");
           scanf("%d",&mod);
           total=mod/9000;
           printf("El costo unitario de mano de obra directa es
               $%.3f",total);
           if (total<=2) printf("\nExcelente, el costo unitario de mano
              de obra directa ha disminuido lo que provoca mas
              produccion\n");
```

```
                else
                printf("\nHay que disminuir el costo de mano de obra
                    provoca menos produccion\n");
                break;
        case 3: printf("Dame el total de costos indirectos\n");
                scanf("%d",&cin);
                total=cin/9000;
                printf("El costo unitario de costos indirectos es
                    $%.3f",total);
                if (total<=2) printf("\nExcelente, el costo unitario de
                    costos indirectos ha disminuido lo que provoca mas
                    produccion\n");
                else
                   printf("\nHay que disminuir el costo de indirecto,
provoca menos produccion\n");
                break;
         default:
                  printf("Opcion invalida");
                  }
       getch();
}
```

9.5.8 Calcular el color de una onda de longitud

```
/*Colores onda longitud*/
#include<conio.h>
#include<stdio.h>
void main()
{
  clrscr();
  int l;
  printf ("El programa te permite saber el color que tiene una onda de
    cierta longitud\n");
  printf ("Los promedios a los que se puede observar la luz son:\n");
  printf                ("1)400nm\n2)450nm\n3)470nm\n4)500nm\n5)570nm\
       n6)590nm\n7)610nm\n8)780nm\n");
  printf ("Cual es el caso de la longitud de la onda\n");
  scanf ("%d",&l);
  switch (l)
  {
```

```
case 1: printf ("ondas fuera del rango de la luz visible\n");
        break;
case 2: printf ("el color es violeta\n");
        break;
case 3: printf ("el color es indigo\n");
         break;
case 4: printf ("el color es azul\n");
        break;
case 5: printf ("el color es verde\n");
        break;
case 6: printf ("el color es amarillo");
        break;
case 7: printf ("el color es naranja\n");
        break;
case 8: printf ("el color es rojo\n");
        default:
        printf ("La onda escapa a la longitud de onda de la luz
            visible\n");
    }
 getch();
}
```

9.6 Estructura repetitiva while

9.6.1 Calcular la producción de cualquier empresa en un día, una semana o un mes

```
/*Productos en una hora*/
#include<conio.h>
#include<stdio.h>
void main()
{
   int t=0,p=0,m;
   clrscr();
   printf("\nCalcular la produccion de cualquier empresa en un dia,
       semana y mes \n");
   printf("\nCuantos minutos se tarda en ser elaborado el producto
       (dame minutos enteros)?\n");
```

```
    scanf("%d",&m);
    while (t<1440)
    {
       p=p+1;
       t=t+m;
     }
    printf("\nLos productos elaborados en un dia son: %d\n",p);
    printf("\nLos productos elaborados en una semana son(lunes a domingo):
       %d\n",p*7);
    printf("\nLos productos elaborados en un mes son(mes de 30 dias):
       %d\n",p*30);
    getch();
}
```

9.6.2 Sumar los elementos de una progresión aritmética

```
/* Suma de progresión aritmética*/
 #include<conio.h>
#include<stdio.h>
void main()
{
 clrscr();
 int i,n;
 float sum,d,x,x1,resul;
 printf("Suma los elementos de una progresion aritmetica\n");
 printf("Dame el numero de terminos de la progresion ");
 scanf("%d",&n);
 printf("Dame el valor del primer termino ");
 scanf("%f",&x);
 printf("Dame el valor de la diferencia comun ");
 scanf("%f",&d);
 i=2;
 x1=x;
 sum=0;
 printf("%.2f + ",x);
  while(i<=n)
    {
       x+=d;
       printf("%.2f + ",x);
```

```
        sum+=x;
        resul=sum+x1;
        i++;
       }
 printf("\b\b");
 printf("= %.2f",resul);
 getch();
}
```

9.6.3 Sumar los elementos de una progresión geométrica

```
/* Suma de progresión geométrica */
 #include<conio.h>
 #include<stdio.h>
 void main()
 {
  clrscr();
  int i,n;
 float sum,r,x,x1,resul;
 printf("Suma los elementos de una progresion geometrica\n");
 printf("Dame el numero de terminos de la progresion ");
 scanf("%d",&n);
 printf("Dame el valor del primer termino ");
 scanf("%f",&x);
 printf("Dame el valor de la razon ");
 scanf("%f",&r);
 i=2;
 x1=x;
 sum=0;
 printf("%.2f + ",x);
 while(i<=n)
  {
      x*=r;
      printf("%.2f + ",x);
      sum+=x;
      resul=sum+x1;
      i++;
    }
 printf("\b\b");
```

```
 printf("= %.2f",resul);
 getch();
}
```

9.6.4 Obtener la nómina de hombres y mujeres, y su promedio

```
/*Nómina*/
#include<stdio.h>
#include<conio.h>
void main()
{
    int x,pht,nht,sexo,ptt,sumasm,sumasf;
    float sm,sf;
    clrscr();
     printf("\n\tCuanto ganan los hombres y mujeres en total\n\t y el
         promedio de acuerdo al numero de ellos?\n");
 sumasm=0;
 sumasf=0;
 sm=0;
 sf=0;
while ((sumasm<=10000) && (sumasf<=10000))
  {
    printf("elije tu sexo\n1)hombre\n2)mujer\n");
    scanf("%d",&sexo);
    if (sexo==1)
    {
      sm=sm+1;
      printf("Ingrese num. de horas trabajadas \n");
      scanf("%d",&nht);
      printf("Ingrese pago por hora trabajada\n");
      scanf("%d",&pht);
      ptt=nht*pht;
      printf(" gana %d pesos \n\n",ptt);
      sumasm=sumasm+ptt;
     }
    else
    {
      sf=sf+1;
```

```
        printf("Ingrese num. de horas trabajadas \n");
        scanf("%d",&nht);
        printf("Ingrese pago por hora trabajada\n");
        scanf("%d",&pht);
        ptt=nht*pht;
        printf("GANA %d pesos \n\n",ptt);
        sumasf=sumasf+ptt;
       }
     }
    printf("los hombres ganan total: %d pesos, promedio: %.2f
        \n",sumasm,sumasm/sm);
    printf("las mujeres ganan total: %d pesos, promedio:
        %.2f\n",sumasf,sumasf/sf);
   getch();
}
```

9.6.5 Calcular el precio del concreto según su resistencia

```
/*Cemex*/
#include<conio.h>
#include<stdio.h>
void main()
{
  int x,p;
  clrscr();
  printf("Te dare el precio de las ollas de concreto segun resistencia
       del concreto\n");
  x=50;
  p=1000;
  while (x<=500)
  {
    printf("El precio de la olla de cemex %d es %d pesos\n",x,p);
    x=x+50;
    p=p+1000;
   }
 getch();
}
```

9.6.6 Calcular el coeficiente de correlación entre dos variables

```
/*Coeficiente de correlación*/
#include<conio.h>
#include<stdio.h>
#include<math.h>
void main()
{
  clrscr();
  double cov1,r, x, y, sx=0, sy=0, sxy=0, sx2=0, sy2=0, cov, dx, dy;
  double rcov, rdx, rdy, dx1, dy1;
  int cont=0;
  char op='s';
  char op1='s';
  while (op=='s' || op=='S')
  {
    printf ("El programa permite calcular el coeficiente de coorelacion
        entre dos variables\n(dos pares de numeros)\n");
while (op1=='s' || op1=='S')
{
  printf ("Dame el valor de x\n");
  scanf ("%lf",&x);
  printf ("Dame el valor de y\n");
  scanf ("%lf",&y);
  cont=cont+1;
  sx=sx+x;
  sy=sy+y;
  sxy=(x*y)+sxy;
  sx2=sx2+(x*x);
  sy2=sy2+(y*y);
  printf ("Desea ingresar otro conjunto de valores?\nsi/no\n");
  scanf ("%s",&op1);
 }
cov=sxy-(sx*sy/cont);
cov1=cov/(cont-1);
dx=sx2-(sx*sx/cont);
dx1=dx/(cont-1);
dy=sy2-(sy*sy/cont);
```

```
    dy1=dy/(cont-1);
    rdx=sqrt (dx1);
    rdy=sqrt (dy1);
    r=cov1/(rdx*rdy);
    printf ("El coeficiente de correlacion es %f\n", r);
    printf ("Desea calcular otro coeficiente de correlacion?\nsi/no\n");
    scanf ("%s",&op);
   }
 getch();
}
```

9.6.7 Calcular el balance de masa

```
/*Balance de masa*/
/*Reacción de elementos*/
#include<stdio.h>
#include<conio.h>
void main()
{
   float m,mf;
   int e,n=1;
   textcolor(RED);
   clrscr();
   gotoxy(23,3);
   printf("Balance de masa\n\n Cuantos elementos reaccionan? ");
   scanf("%i",&e);
   mf=0;
   while (n<=e)
   {
     printf("\n Introduce la masa del elemento %i: ",n);
     scanf("%f",&m);
     mf=mf+m;
     n=n+1;
   }
   printf("\n\n La masa total inicial es %f. Verifique que la masa total
         de los productos sea la misma que la de los reactivos",m);
  getch();
}
```

9.6.8 Imprimir la fracción mol de los elementos de un compuesto

```
/*Fracción mol*/
#include<stdio.h>
#include<conio.h>
void main()
{
   int v,e;
   float m,mc,porc;
   textcolor(RED);
   clrscr();
   gotoxy(12,3);
   printf("Imprime la fraccion mol de los elementos de un compuesto\n\n
         Cuantos elementos tiene el compuesto?\n");
   scanf("%i",&e);
   printf("\n Introduce los moles del compuesto\n");
   scanf("%f",&mc);
   v=1;
   while (v<=e)
   {
     clrscr();
     printf("\n Dame los moles del elemento %i: ",v);
     scanf("%f",&m);
     porc=m/mc;
     printf("\n La fraccion mol del compuesto %i = %f",v,porc);
     v=v+1;
    }
 getch();
}
```

9.6.9 Calcular porcentaje en presión

```
/*Porciento presión*/
#include<stdio.h>
#include<conio.h>
void main()
{
  float PT,pp,porcp;
```

```
int n=1,ne;
textcolor(RED);
clrscr();
gotoxy(19,3);
printf("Calculo del porcentaje en presion\n\n Introduzca la presion
     total del sistema: ");
scanf("%f",&PT);
printf("\n Ahora la cantidad de elementos/compuestos tiene la mezcla
     gaseosa: ");
scanf("%i",&ne);
while (n<=ne)
{
  printf("\n Introduce la presion parcial del componente %i\n",n);
  scanf("%f",&pp);
  porcp=pp/PT*100;
  printf("\n El porcentaje en presion es=%f para el compuesto
       %i",porcp,n);
  n=n+1;
   }
 getch();
}
```

9.6.10 Calcular los átomos de un elemento

```
/* Tomos de algún elemento*/
#include<stdio.h>
#include<conio.h>
void main()
{
   float pa,g,a;
  char op='S';
  textcolor(RED);
  while (op=='S')
  {
    clrscr();
    gotoxy(26,3);
    printf("Atomos de un elemento\n\n Introduce el peso atomico del
         elemento\n");
    scanf("%f",&pa);
```

```
    printf("\n Ahora los gramos del elemento\n");
    scanf("%f",&g);
    a=(g*6.023e23)/pa;
    printf("\n%f  tomos\n\n Desea calcularlo de nuevo? S/N: ",a);
    scanf("%s",&op);
   }
 getch();
}
```

9.7 Estructura repetitiva do-while

9.7.1 Calcular el determinante, dados el cofactor y el vector

```
/* Cálculo del determinante dados el vector y el cofactor */
 #include<stdio.h>
 #include<conio.h>
 void main()
{
  clrscr();
  int i,n;
  float cof,a,sum=0,p;
  printf("Calcula el determinante dados el cofactor y el vector\n");
  printf("Dame el orden de la matriz ");
  scanf("%d",&n);
  i=1;
  do
   {
    printf("Dame el componente %d del vector ",i);
    scanf("%f",&a);
    printf("Dame el valor del cofactor ",i);
    scanf("%f",&cof);
    p=a*cof;
    sum+=p;
    i++;
   }
  while(i<=n);
  printf("El determinante es %.2f",sum);
  getch();
}
```

9.7.2 Calcular la frecuencia relativa de datos


```
/* Calcula frecuencia relativa de datos */
 #include<stdio.h>
 #include<conio.h>
 void main()
 {
   clrscr();
   int i=1;
   long int c;
   float x,frec,n;
   printf("Calcula la frecuencia relativa de n datos\n");
   printf("Dame la cantidad total de observaciones a evaluar ");
   scanf("%f",&n);
   printf("Dame el numero de categorias ");
   scanf("%ld",&c);

    do
     {
      printf("Dame la cantidad en la categoria %d: ",i);
      scanf("%f",&x);
      frec=x/n;
      printf("La frecuencia relativa es:%.4f\n",frec);
      i++;
     }
    while(i<=c);
  getch();
 }
```

9.7.3 Juego del ahorcado

```
/*Ahorcado*/
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<stdlib.h>
main()
```

```
{
  int op, p, c, e;
  char l;
  e=0;
  c=0;
  clrscr();
  textcolor(15);
  gotoxy(20,5);
  cprintf("Bienvenido a * Ahorcado *");
  delay(1000);
  gotoxy(20,7);
  cprintf("1) Jugar");
  gotoxy(20,9);
  cprintf("0) Salir");
  gotoxy(20,12);
  cprintf("( )");
  gotoxy(21,12);
delay(1000);
scanf("%d",&op);
switch(op)
 {
    case 1: srand(time(NULL));
            p=rand()%5;
            switch(p)
                 {
         default:
                      clrscr();
                cprintf("Adivina las letras");
                gotoxy(20,10);
                cprintf("_ _ _");
                do
                                 {
                   gotoxy(15,5);
                   cprintf("( )");
                   gotoxy(16,5);
                    if (e==5)
                   exit(1);
                   scanf("%s",&l);
```

```
switch(l)
                    {
      case 'L': gotoxy(20,10);
                cprintf("L");
                gotoxy(20,11);
                printf("Correcto!");
                c=c+1;
                break;
      case 'I': gotoxy(22,10);
                printf("I");
                gotoxy(20,11);
                cprintf("Correcto!");
                c=c+1;
                break;
      case 'Z': gotoxy(24,10);
                cprintf("Z");
                gotoxy(20,11);
                cprintf("Correcto!");
                c=c+1;
                break;
                default:
                                delay(1000);
                                e=e+1;
                                gotoxy(15,5);
                                cprintf("Numero
                                   de errores:
                                   %d",e);
                                gotoxy(20,12);
                               cprintf("Error");
            }

            }
                  while (c!=3);
                 }
               delay(1000);
               break;
               default:
                         exit(1);
            }
   getch();
}
```

9.8 Estructura repetitiva for

9.8.1 Calcular la aceleración de un cuerpo cada segundo, los primeros ocho segundos

```
/* Aceleración final*/
#include <conio.h>
#include <stdio.h>
void main ()
{
  clrscr ();
  float vf, vo, a, t;
  char op; op='s';
  while (op == 's')
  {
    printf ("El programa calcula la aceleracion cada segundo durante los
         ocho primeros segundos\n");
    printf ("Dame la velocidad inicial\n");
    scanf ("%f", & vo);
    printf ("Dame la aceleracion\n");
    scanf ("%f", & a);
    for (t=1; t<=8; t++)
    {
      vf=vo+(a*t);
      printf ("La aceleracion a los %fs = %f\n", t, vf);
     }
     printf ("Quieres calcular alguna otra velocidad final?\n");
     scanf ("%s", & op);
    }
    getch ();
 }
```

9.8.2 Determinar la cantidad de productos defectuosos y perfectos

```
/*Calidad de los productos*/
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
  int pro,cont,p,de=0,per=0;
  clrscr();
  printf("\n\t\tChecar la calidad de los productos\n");
  printf("\nCuantos productos se elaboraron?\n");
  scanf("%d",&pro);
  for (cont=1;cont<=pro;cont++)
  {
    printf("\nEl producto, tiene algun defecto? si=1 no=2\n");
    scanf("%d",&p);
    if (p==1)
    {
      printf("Desecha el producto\n");
      de=de+1;
     }
    else
    {
      printf("El producto esta perfecto; empacalo\n");
      per=per+1;
     }
    }
    printf("\nEl total de productos defectuosos es =%d",de);
    printf("\nEl total de productos perfectos es =%d",per);
    getch();
  }
```

9.8.3 Calcular el salario de un trabajador, dependiendo de las piezas que elaboró

```
/*Pago por cada pieza elaborada*/
#include<stdio.h>
#include<conio.h>
void main()
{
    int pie,tra,cont;
    float sue=0,pa,s;
    clrscr();
    printf("\n\t\tSueldo de un trabajador por cada pieza elaborada\n");
```

```
printf("\nCuantos empleados trabajan por piezas elaboradas?\n");
scanf("%d",&tra);
for (cont=1;cont<=tra;cont++)
 {
     printf("\nEmpleado numero %d\n",cont);
     printf("Cuantas piezas elaboraste?\n");
     scanf("%d",&pie);
     printf("Cuanto se te paga por cada pieza?\n");
     scanf("%f",&pa);
     if (pie<=30)
      {
         s=pa*pie;
         sue=sue+s;
         printf("Tu salario es $%.2f\n",sue);
      }
     else
      {
         s=(pa*pie)*.20;
         sue=sue+s;
         printf("Tu salario es $%.2f",sue);
      }
  }
 getch();
}
```

9.8.4 Calcular diluciones por pasos

```
/*Diluciones por pasos*/
#include<stdio.h>
#include<conio.h>
void main()
{
  float c1,c2,v1,v2,nc;
  int i,n;
  textcolor(RED);
  clrscr();
  gotoxy(27,3);
  printf("Realiza una dilucion por pasos\n\n Indica de cuantos pasos deseas
       realizar la dilucion: ");
```

```
scanf("%i",&n);
printf("\n Introduce el primer volumen: ");
scanf("%f",&v1);
printf("\n Y la concentracion inicial: ");
scanf("%f",&c1);
printf("\n Cuanto volumen vas a agregar? ");
scanf("%f",&v2);
c2=v1*c1/(v2+v1);
printf("\n La concentracion en la primer dilucion es: ",c2);
for (i=1;i<n;i++)
{
  clrscr();
  printf("\n Que volumen de la solucion con la concentracion anterior
       quiere?\n");
  scanf("%f",&v1);
  printf("\n Que volumen va a agregar?\n");
  scanf("%f",&v2);
  nc=(v1*c2)/(v2+v1);
   printf("\n Ahora la concentracion es %f",nc);
   c2=nc;
  }
 getch();
}
```

9.8.5 Indicar los moles de un elemento

```
/*Moles*/
#include<stdio.h>
#include<conio.h>
void main()
{
  float n,o,mc,pe,pae,mol;
  int i;
  textcolor(RED);
  clrscr();
  gotoxy(9,3);
  printf("Indica los moles de cada elemento de formula molecular
       desconocida\n\n  Dame el num. de elementos que tiene: ");
  scanf("%f",&n);
  printf("\n Ahora la masa del compuesto: ");
```

```
 scanf("%f",&mc);
 for(i=1;i<=n;i++)
 {
   printf("\n Introduce el porcentaje en peso del elemento %i\n",i);
   scanf("%f",&pe);
   printf("\n Ahora el peso atomico del elemento\n");
   scanf("%f",&pae);
   mol=(mc/(pe/100))/pae;
   printf("\n%f moles",mol);
 }
getch();
}
```

9.8.6 Indicar la cantidad de agua que se necesita para diluir una solución

```
/*Disoluciones*/
#include<stdio.h>
#include<conio.h>
void main()
{
  float v1,v2,va,c1,c2;
  int i;
  textcolor(RED);
  clrscr();
  gotoxy(32,3);
  printf("Disoluciones\n\n Introduzca el volumen a diluir: ");
  scanf("%f",&v1);
  printf("\n La concentracion de dicha solucion: ");
  scanf("%f",&c1);
  printf("\n Finalmente la concentracion a la que quiere llegar: ");
  scanf("%f",&c2);
  v2=v1*c1/c2;
  va=v2-v1;
  printf("\n El volumen de agua a agregar es %f. Otras concentraciones
       (submultiplos del volumen inicial)",va);
  for (i=1;i<=3;i++)
```

```
  {
    v1=v1/10;
    v2=v1*c1/c2;
    va=v2-v1;
    printf("\n Para %f se necesitan %f de agua",v1,va);
   }
 getch();
}
```

9.8.7 Calcular la presión parcial de un componente

```
/*Presión parcial*/
#include<stdio.h>
#include<conio.h>
void main()
{
   float m,p,v,t,pt=0;
   int n,i;
   textcolor(RED);
   clrscr();
   gotoxy(26,3);
   printf("Presiones parciales\n\n Cuantos componentes tiene el gas?\n");
   scanf("%i",&n);
   for (i=1;i<=n;i++)
   {
     clrscr();
     printf("\n Introduce los moles del compuesto %i: ",i);
     scanf("%f",&m);
     printf("\n Ahora la temperatura: ");
     scanf("%f",&t);
     printf("\n Finalmente el volumen: ");
     scanf("%f",&v);
     p=m*0.0821*t/v;
     pt=pt+p;
     printf("\n La presion parcial del componente es: %f",p);
    }
  printf("\n La presion total es %f",pt);
  getch();
}
```

9.8.8 Calcular la varianza de *X*

```
/* Calcula la varianza de X */
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
  int n,i;
  float ex,fx,x,prod,sum;
  printf("Calcula la varianza de X\n");
  printf("Dame el numero de valores que se tomaron para X ");
  scanf("%d",&n);
  sum=0;
  printf("Dame el valor de la esperanza de X ");
  scanf("%f",&ex);
  for (i=1;i<=n;i++)
  {
    printf("Dame el valor %d de X ",i);
    scanf("%f",&x);
    printf("Dame el valor %d de la funcion de probabilidades ",i);
    scanf("%f",&fx);
    prod=(x-ex)*(x-ex)*fx;
    sum+=prod;
   }
 printf("La varianza de X es de: %.4f",sum);
 getch();
}
```

9.8.9 Calcular la derivada de *X* a la *n*

```
/* Derivadas de X a la n */
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
  int x,n,i,acum,o;
  printf("Derivadas de X a la n de orden superior\n");
```

```
 printf("Dame la constante ");
 scanf("%d",&x);
 printf("Dame el exponente ");
 scanf("%d",&n);
 printf("De que orden quieres calcular la derivada? ");
 scanf("%d",&o);
 for(i=1;i<=o;i++)
 {
     x*=n;
     n--;
  }
printf("La derivada es %dX^%d\n",x,n);
getch();
}
```

9.9 Arreglos unidimensionales

9.9.1 Calcular la magnitud al cuadrado de un vector

```
/*Cálculo de la magnitud de un vector*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
   int e,i,a[20],aa[20],AA;
   printf("Magnitud al cuadrado de un vector");
   printf("De cuantos elementos es el vector (maximo 20)?");
   scanf("%i",&e);
   for (i=0;i<e;i=i+1)
   {
      printf("Dame el valor del elemento %i del primer vector",i+1);
      scanf("%i",&a[i]);
   }
   for (i=0;i<e;i=i+1) aa[i]=a[i]*a[i];
   AA=aa[0];
   for (i=1;i<e;i=i+1) AA=AA+aa[i];
    printf("%i",AA);
    getch();
  }
```

9.9.2 Calcular el producto cruz de dos vectores

```
/*Producto cruz*/
#include<stdio.h>
#include<conio.h>
void main()
{

   int i,a[3],b[3],ab[3],AB;
   printf("Producto cruz de dos vectores");
   printf("Recuerda que son de 3 elementos");
   for (i=0;i<3;i=i+1)
   {
    printf("Dame el valor del elemento %i del primer vector",i+1);
    scanf("%i",&a[i]);
   }
   for (i=0;i<3;i=i+1)
   {
    printf("Dame el valor del elemento %i del segundo vector",i+1);
    scanf("%i",&b[i]);
   }
   ab[0]=(a[2]*b[3])-(b[2]*a[3]);
   ab[1]=(a[1]*b[3])-(b[1]*a[3]);
   ab[2]=(a[1]*b[2])-(b[1]*a[2]);
   AB=ab[0];
   for (i=1;i<3;i=i+1) AB=AB+ab[i];
   printf("%i",AB);
   getch();
}
```

9.9.3 Calcular el reactivo limitante de una reacción

```
/*Reactivo limitante*/
#include<stdio.h>
#include<conio.h>
void main()
{
  float reac,pm,moles[15],comp;
  int c,i,elemento;
```

```
printf("Reactivo limitante");
printf("Indique cuantos compuestos tiene la reaccion (no mas de 15)");
scanf("%i",&c);
for (i=0;i<c;i=i+1)
{
   printf("Dame la cantidad del reactivo %i", i+1);
   scanf("%f",&reac);
   printf("Dame el peso molecular del reactivo");
   scanf("%f",&pm);
   moles[i]=reac/pm;
 }
 comp=moles[0];
 for (i=1;i<c;i=i+1)
 {
   if (comp<moles[i])
   {
     comp=moles[i];
     elemento=i;
    }
  }
 printf("El reactivo limitante es %i", elemento);
 getch();
}
```

9.9.4 Calcular la proyección entre dos vectores de *n* elementos

```
/* Calcula la proyección entre dos vectores de n elementos */
 #include<stdio.h>
 #include<conio.h>
 void main()
 {
   int n,i;
   float v1[100],v2[100],pro,propun,mag2,den,esc,com;
   char op;
   textcolor(BLUE);
   textbackground(WHITE);
   do
    {
```

```
  clrscr();
  printf("Calcula la proyeccion entre dos vectores\n\n");
  printf("Dame el numero de elementos de los vectores: ");
  scanf("%d",&n);
  propun=0;
  den=0;
  printf("\n");
   for(i=0;i<n;i++)
   {
      printf("Dame la componente %d del vector sobre el cual es la
proyeccion: ",i+1);
      scanf("%f",&v1[i]);
    }
    printf("\n");
    for(i=0;i<n;i++)
    {
       printf("Dame la componente %d del otro vector: ",i+1);
       scanf("%f",&v2[i]);
     }
     for (i=0;i<n;i++)
     {
       pro=v1[i]*v2[i];
       propun+=pro;
       mag2=v1[i]*v1[i];
       den+=mag2;
      }
     esc=propun/den;
     printf("\nLa proyeccion resultante es el vector:   ");
     printf("{ ");
     for (i=0;i<n;i++)
     {
        com=v1[i]*esc;
printf( "%.2f ",com);
      }

printf("}\n");
printf("\nDesea calcular otra proyeccion? S/N ");
scanf("%s",&op);
}
```

```
    while((op=='s')||(op=='S'));
     getch();
  }
```

9.9.5 Calcular el ángulo en grados entre dos vectores

```
/* Ángulo en grados entre dos vectores de n componentes */
 #include<stdio.h>
 #include<conio.h>
 #include<math.h>
 void main()
  {
    int n,i;
    float v1[100],v2[100],m1,m2,mag1,mag2,pro,propun,x,anrad,angrad;
    char op;
    textcolor(WHITE);
    textbackground(BLUE);
    do
{
  clrscr();
  printf("Calcula el angulo entre dos vectores\n\n");
  printf("Dame el numero de elementos de los vectores: ");
  scanf("%d",&n);
  mag1=0;
  propun=0;
  mag2=0;
  printf("\n");
   for (i=0;i<n;i++)
   {
     printf("Dame la componente %d del primer vector: ",i+1);
     scanf("%f",&v1[i]);
    }
   printf("\n");
   for (i=0;i<n;i++)
    {
      printf("Dame la componente %d del segundo vector: ",i+1);
      scanf("%f",&v2[i]);
     }
   for (i=0;i<n;i++)
```

```
    {
       pro=v1[i]*v2[i];
       propun+=pro;
       m1=v1[i]*v1[i];
       mag1+=m1;
      m2=v2[i]*v2[i];
      mag2+=m2;
     }

   x=propun/(sqrt(mag1*mag2));
   anrad=acos(x);
   angrad=180*anrad/3.14159;
   printf("\nEl angulo entre los vectores es de %.2fø\n",angrad);
   printf("\nDeseas hacer otro calculo? S/N ");
   scanf("%s",&op);
  }
while ((op=='s')||(op=='S'));
getch();
}
```

9.9.6 Calcular la desviación estándar y varianza muestral de *n* datos

```
/* Calcula la desviación estándar y varianza muestral de n datos */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
 {
   int i,n;
   char op;
   float dato[100],sum,prom,var,desv,sumat;
   textbackground(BLUE);
   textcolor(WHITE);
   do
    {
     clrscr();
     sum=0;
     sumat=0;
```

```
printf("Calcula la desviacion estandar y la varianza de n datos\
    n\n");
printf("Dame el numero de datos de tu muestra: ");
scanf("%d",&n);
printf("\n");
for (i=0;i<n;i++)
 {
    printf("Dame el valor %d de tu muestra: ",i+1);
    scanf("%f",&dato[i]);
    sum+=dato[i];
  }
  prom=sum/n;
  for (i=0;i<n;i++)
   {
sumat+=pow(dato[i]-prom,2);
    }
   var=sumat/(n-1);
    desv=sqrt(var);
    printf("\nLa varianza es igual a %.3f\n",var);
    printf("\nLa desviacion estandar es igual a: %.3f\n",desv);
    printf("\nDeseas repetir el programa? S/N ");
   scanf("%s",&op);
   }
  while ((op=='s')||(op=='S'));
  getch();
}
```

9.9.7 Calcular el producto punto entre tres vectores

```
/*Producto punto entre vectores*/
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main ()
{
 clrscr ();
```

```
float v1[10], v2[10], v3[10];
int i, n;
char op='s';
printf ("producto punto entre 3 vectores\n");
while (op=='s'|| op=='S')
{
  printf ("Dame la medida de los vectores max 10\n");
  scanf ("%d", & n);
  for (i=0; i<n; i++)
  {
    printf ("Dame el dato %d del primer vector\n", i+1);
    scanf ("%f", & v1[i]);
   printf ("Dame el dato %d del segundo vector\n", i+1);
   scanf ("%f", & v2[i]);
   printf ("Dame el dato %d del tercer vector\n", i+1);
   scanf ("%f", & v3[i]);
  }
 for (i=0; i<n; i++)
  printf ("el componente %d del producto punto entre los vectores es
    %7.2f\n", i+1,     v1[i]*v2[i]*v3[i]);
  printf ("Deseas calcular otro producto punto\n?");
  scanf ("%s", & op);
  }
 getch ();
}
```

9.9.8 Calcular los estimadores de la recta de regresión

```
/*Estimadores*/
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main ()
{
  clrscr();
  int i, n, X;
  float x[30], y[30], a=0, b=0, c=0, cov, d=0, B0, B1, Y, my=0;
  char op='s';
  printf ("El programa calcula los estimadores de la recta de
       regresion\n");
```

```
while (op=='s'||op=='S')
{
  printf ("Dame la cantidad de datos, max 30\n");
 scanf ("%d", & n);
 for (i=0; i<n; i++)
 {
    printf ("Dame un valor del eje x\n");
    scanf ("%f", & x[i]);
    printf ("Dame un valor del eje y\n");
    scanf ("%f", & y[i]);
    a=a+(x[i]*y[i]);
    b=b+x[i];
    c=c+y[i];
    d=d+pow(x[i],2);
    my=my+y[i];
  }
 Y=my/n;
   printf ("Dame el valor de X\n");
   scanf ("%d", & X);
   B1=(a-((b*c)/n))/(d-(pow(b,2)/n));
   B0=Y-(B1*X);
   printf ("B1=%f\n", B1);
   printf ("B0=%f\n", B0);
   printf ("La recta de regresion esta dada por %7.4f + %7.4f\n", B0, B1);
   printf ("Deseas calcular otra recta de regresion? s/n\n");
   scanf ("%s", & op);
   }
 getch ();
}
```

9.9.9 Calcular el coeficiente de correlación

```
/*Coeficiente de correlación*/
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main ()
{
```

```
clrscr();
int i, n;
float x[30], y[30], a=0, b=0, c=0, cov, d=0, e=0, cdc;
char op='s';
printf ("El programa calcula el coeficiente de correlacion\n");
while (op=='s'||op=='S')
{
   printf ("Dame la cantidad de datos, max. 30\n");
   scanf ("%d", & n);

   for (i=0; i<n; i++)
   {
     printf ("Dame un valor del eje x\n");
     scanf ("%f", & x[i]);
     printf ("Dame un valor del eje y\n");
     scanf ("%f", & y[i]);
     a=a+(x[i]*y[i]);
     b=b+x[i];
     c=c+y[i];
     d=d+pow(x[i],2);
     e=e+pow(y[i],2);
    }
   cov=((a-((b*c)/n))/(n-1));
   cdc=pow((a-((b*c)/n))/(d-(pow(b,2)/n)*(e-(pow(c,2)/n))), 1/2);
   printf ("La covarianza es %7.4f\n", cov);
   printf ("El coeficiente de correlacion es %f\n", cdc);
   printf ("Deseas calcular otro coeficiente de correlacion? s/n\n");
   scanf ("%s", & op);
  }
 getch ();
}
```

9.9.10 Calcular la covarianza

```
/*Covarianza*/
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main ()
```

```
{
  clrscr();
  int i, n;
  float x[30], y[30], a=0, b=0, c=0, cov;
  char op='s';
  printf ("El programa calcula la covarianza\n");
  while (op=='s'||op=='S')
  {
    printf ("Dame la cantidad de datos, max. 30\n");
    scanf ("%d", & n);
    for (i=0; i<n; i++)
    {
      printf ("Dame un valor del eje x\n");
      scanf ("%f", & x[i]);
      printf ("Dame un valor del eje y\n");
      scanf ("%f", & y[i]);
      a=a+(x[i]*y[i]);
      b=b+x[i];
      c=c+y[i];
    }


    cov=((a-((b*c)/n))/(n-1));
    printf ("La sumatoria del eje x es %7.2f\n", b);
    printf ("La sumatoria del eje y es %7.2f\n", c);
    printf ("La sumatoria de la multiplicacion de los ejes es %7.2f\n",
        a);
    printf ("La covarianza es %7.4f\n", cov);
    printf ("Deseas calcular otra covarianza? s/n\n");
    scanf ("%s", & op);
  }
  getch ();
}
```

9.9.11 Calcular ganancias y ventas de una pastelería

```
/*Inventario de una pastelería*/
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
      int d,p[30],g[30],pa[30],ac=0,acu=0,acum=0,x,i;
      float  dp,dpa,dg;
      clrscr();
      printf("\t\tInventario de una pasteleria\n");
      printf("\nCon este inventario te dare las ganacias y ventas de los
           dias que quieras\n");
      printf("\nDame el precio de un pastel\n");
      scanf("%f",&dp);
      printf("Dame el precio de una gelatina\n");
      scanf("%f",&dg);
      printf("Dame el precio de un pay\n");
      scanf("%f",&dpa);
      printf("De cuantos dias quieres tener tu inventario(maximo un mes)?
           \n");
      scanf("%d",&d);
      for (i=0;i<d;i++)
   {
        printf("Dame el total de ventas en el dia %d\n",i+1);
        printf("Cuantos pasteles vendiste?\n");
        scanf("%d",&p[i]);
        printf("Cuantas gelatinas vendiste?\n");
        scanf("%d",&g[i]);
        printf("Cuantos pays vendiste?\n");
        scanf("%d",&pa[i]);
        ac=ac+p[i];
        acu=acu+g[i],
        acum=acum+pa[i];
   }
printf("\nEl total de ventas en %d dias fueron:\n",d);
printf("\n\t\tpasteles\t\tgelatinas\t\tpays\n\n");
printf("\t\t%d\t\t\t%d\t\t\t%d",ac,acu,acum);
printf("\n\nDe que dia quieres saber las ganancias que obtuviste?\n");
scanf("%d",&x);
printf("\n\tEl dia %d obtuviste la siguientes ganancias y ventas\
     n",x);
printf("\n\tPasteles: %d ventas con una ganancia de %.2f\n",p[x-
     1],p[x-1]*dp);
```

```
        printf("\n\tGelatinas: %d ventas con una ganancia de
            %.2f\n",g[x-1],g[x-1]*dg);
        printf("\n\tPays: %d ventas con una ganancia de
            %.2f\n",pa[x-1],pa[x-1]*dpa);
        getch();
}
```

9.10 Arreglos bidimensionales

9.10.1 Calcular la matriz traspuesta

```
/*Matriz traspuesta*/
#include <conio.h>
#include <stdio.h>
void main ()
{
   clrscr ();
int i, j, m[10][10], n, x;
printf ("Matriz traspuesta\n");
printf ("Dame el numero de filas\n");
scanf ("%d", & n);
printf ("Dame el numero de columnas\n");
scanf ("%d", & x);
for (i=0; i<n; i++)
 for (j=0; j<x; j++)
 {
   printf ("Dame el elemento con coordenada %d %d\n", i+1, j+1);
   scanf ("%d", & m[i][j]);
  }
  for (j=0; j<x; j++)
  {
   for (i=0; i<n; i++)
   {
     printf ("%d ", m[i][j]);
    }
    printf ("\n");
   }
 getch ();
}
```

9.10.2 Crear una tabla con los tipos de concreto disponibles, junto con sus resistencias

```
/*Concretos*/
#include<conio.h>
#include<stdio.h>
void main()
{
  int con[99][99],c,s,x,y;
  clrscr();
  printf("Tabulare la resistencia de concretos segun la semana de prueba\
       nCuantos tipos de concretos tienes? \n");
 scanf("%d",&c);
 printf("Cuantas semanas haras pruebas?\n");
 scanf("%d",&s);
 for (x=0;x<c;x++)
 {
   for (y=0;y<s;y++)
   {
    printf("Dime la resistencia del concreto %d en la semana
         %d \n",x+1,y+1);
    scanf("%d",&con[x][y]);
    }
  }
  for (x=0;x<c;x++)
  {
    printf("\n");
    for (y=0;y<s;y++)
    {
      printf("%d ",con[x][y]);
     }
   }
 getch();
}
```

9.10.3 Calcular la determinante de un matriz triangular

```
/*Determinante de una matriz triangular*/
#include<stdio.h>
```

```
#include<conio.h>
void main()
{
   int i,j,x[10][10],det=1,t;
   clrscr();
   gotoxy(16,3);
   printf("Determinante de una matriz triangular\n\n Introduzca el tamanio
       de la matriz (m x. 10x10, solo matrices cuadradas\n");
   scanf("%i",&t);
   for (i=0;i<t;i++)
     for(j=0;j<t;j++)
     {
       if (j==i)
       {
         printf("\n Dame el valor en la posicion %i,%i: ",i+1,j+1);
         scanf("%i",&x[i][j]);
        }
      }
   for (i=0;i<t;i++)
      for(j=0;j<t;j++) if(j==i) det=det*x[i][j];
      printf("\n El determinante es: %i",det);
      getch();
}
```

9.10.4 Calcular la determinante de una matriz de 2 × 2

```
/*Determinante de una matriz de 2x2*/
#include<stdio.h>
#include<conio.h>


void main()
{
  int j,i,x[2][2],det;
  gotoxy(26,3);
  printf("Determinante de una matriz de 2x2\n\n");
  for (i=0;i<2;i++)
    for (j=0;j<2;j++)
    {
```

```
        printf("\n Dame el valor en la posicion %i,%i",i+1,j+1);
        scanf("%i",&x[i][j]);
      }
    det=(x[0][0]*x[1][1])-(x[1][0]*x[0][1]);
    printf("\n El determinante es %i",det);
    getch();
}
```

9.10.5 Devolver el inventario por semana

```
/*Inventario de una semana*/
#include<stdio.h>
#include<conio.h>
void main()
{
  int pro[30][5],i,j,s,se,total;
  clrscr();
  printf("Inventario de produccion de una semana(5dias)\n");
  printf("De cuantas semanas quieres tu inventario?\n");
  scanf("%d",&se);
  for (i=0;i<se;i++)
   {
     total=0;
     printf("SEMANA %d",i+1);
     for (j=0;j<5;j++)
     {
       printf("\nDame la produccion del dia %d\n",j+1);
       scanf("%d",&pro[i][j]);
       total=total+pro[i][j];
      }
      pro[i][j]=total;
     }
   printf("De que semana quieres saber la produccion?\n");
   scanf("%d",&s);
   printf("\n\t\tSemana %d",s);
   for (j=0;j<5;j++)  printf("\n\t\tDia %d = %d productos\n",j+1,pro[s-1]
       [j]);
   getch();
}
```

9.10.6 Calcular en qué turno de la empresa se elaboran más piezas


```
/*Turnos de una empresa*/
#include<stdio.h>
#include<conio.h>
void main()
{
   int t,em,ac,i,j,x,m,pr,p[5][80];
   float sal;
   clrscr();
   printf("\t\tEn que turno de la empresa realizan mas produccion\n");
   printf("\nCuantos turnos brinda la empresa para los trabajadores?\n");
   scanf("%d",&t);
   printf("Cuantos empleados elaboran en cada turno\n?");
   scanf("%d",&em);
   for (i=0;i<t;i++)
   {
    ac=0;
    printf("TURNO NUMERO %d\n",i+1);
    for (j=0;j<em;j++)
      {
        printf("\nCuantas piezas elaboraste, trabajador? %d\n",j+1);
        scanf("%d",&p[i][j]);
        ac=ac+p[i][j];
      }
    p[i][em]=ac;
}
for (i=0;i<t;i++) printf("\n\tTurno %d\t produjo.... %d piezas
    \n\n ",i+1,p[i][em]);;
printf("De que trabajador deseas saber su salario por pieza elaborada?
    \n");
printf("\n\t\tPor cada pieza....0. 9pesos\n");
printf("\nEn que turno labora?\n");
scanf("%d",&x);
printf("Que numero de trabajador es? \n");
scanf("%d",&m);
printf("El trabajador %d del turno %d elaboro %d piezas\n",m,x,p[x-1]
    [m-1]);
```

```
    sal=p[x-1][m-1]*.89;
    printf("\n\tPor lo tanto obtuvo un salario de  $%.2f",sal);
    getch();
}
```

9.10.7 Calcular la cantidad de grasa perdida según las horas de ejercicio realizadas

```
/*Control de peso*/
#include<stdio.h>
#include<conio.h>
void main()
{
  int peso[4][5],i,j,s,se,p,total;
  clrscr();
  printf("Tiempo de ejercicio (semana de 5 dias)\n");
  printf("De cuantas semanas quieres tu control de peso?\n");
  scanf("%d",&se);
for (i=0;i<se;i++)
 {
  total=0;
  printf("Semana %d",i+1);


   for (j=0;j<5;j++)
   {
    printf("\nCuantas horas de ejercicio realizaste en el dia?
         %d\n",j+1);
    scanf("%d",&peso[i][j]);
   total=total+peso[i][j];
   }
   p=total;
  }
printf("De que semana quieres saber la cantidad de horas que
       realizaste?\n");
scanf("%d",&s);
printf("\n\t\tSemana %d",s);
for (j=0;j<5;j++)
{
```

```
    printf("\n\t\tDia %d = %d horas\n",j+1,peso[s-1][j]);
   }
   printf("\n\t\t1 hrs.....Quemas 150 KCAL");
   printf("\n\n\tEl total de horas de la semana %d fue de %d",s,p);
   printf("\n\tPor lo tanto, quemaste %d kilocalorias",p*150);
   getch();
}
```

9.10.8 Calcular la raíz cuadrada de una matriz

```
/*Raiz cuadrada matriz*/
#include<conio.h>
#include<stdio.h>
#include<math.h>
void main()
{
   float x[20][20],y[20][20];
   int i,j,f,c;
   textcolor(3);
   textbackground(0);
   clrscr();
   gotoxy(15,5);printf("Raiz cuadrada de una matriz");
   printf("\n\n\nNumero de filas: ");
   scanf("%d",&f);
   printf("\nNumero de columnas: ");
   scanf("%d",&c);
   printf("\n\n\n");
   for (i=0;i<f;i++)
   {
      for (j=0;j<c;j++)
      {
         printf("Dame el elemento %d,%d: ",i+1,j+1);
         scanf("%f",&x[i][j]);
         y[i][j]=sqrt(x[i][j]);
       }
     }
 printf("\n\nMatriz original\n");
 for (i=0;i<f;i++)
{
```

```
 for (j=0;j<c;j++) printf(" %.1f ",x[i][j]);
     printf("\n");
   }
 printf("\n\nRaiz cuadrada de la matriz: \n");
 for (i=0;i<f;i++)
 {
    for (j=0;j<c;j++)  printf(" %.1f ",y[i][j]);
    printf("\n");
   }
 getch();
}
```

9.10.9 Calcular la multiplicación de una matriz por un escalar

```
/* Multiplicación de matriz por un escalar */
 #include<stdio.h>
 #include<conio.h>
 void main()
  {
   textcolor(WHITE);
   textbackground(BLUE);
   int i,j,m,n;
   char op;
   float mat1[10][10],mat2[10][10],esc;
  do
   {
    clrscr();
    printf("Multiplicacion de matriz por un escalar\n\n");
    printf("Dame el numero de filas de la matriz:  ");
    scanf("%d",&m);
    printf("Dame el numero de columnas de la matriz:  ");
    scanf("%d",&n);
    printf("\n");
     for (i=0;i<m;i++)
       for (j=0;j<n;j++)
       {
  printf("Dame el elemento %d - %d de tu matriz: ",i+1,j+1);
  scanf("%f",&mat1[i][j]);
```

```
    }
  printf("\n");
  printf("Dame el escalar que va a multiplicar tu matriz: ");
  scanf("%f",&esc);
  printf("\n\n");
  for (i=0;i<m;i++)
  {
    for (j=0;j<n;j++)
    {
mat2[i][j]=mat1[i][j]*esc;
printf(" %f ",mat2[i][j]);
     }
     printf("\n");
    }
   printf("\n\nDeseas repetir el programa? S/N ");
   scanf("%s",&op);
  }
 while ((op=='s')||(op=='S'));
 getch();
}
```

9.10.10 Determinar si una matriz es de identidad o no

```
/* Matriz de identidad */
 #include<stdio.h>
 #include<conio.h>
 void main()
 {
   textcolor(BLACK);
   textbackground(WHITE);
   int i,j,mat[10][10],n,cont=0;
   char op;
  do
   {
     clrscr();
     printf("Imprime si la matriz es de identidad o si no lo es\n\n");
     printf("Dame el tamanio de la matriz cuadrada: ");
     scanf("%d",&n);
      for (i=0;i<n;i++)
```

```
      for (j=0;j<n;j++)
      {
       printf(" Dame el valor %d-%d: ",i+1,j+1);
       scanf("%d",&mat[i][j]);
       }
    printf("\n");
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
        if (((i==j)&&(mat[i][j]==1))||((i!=j)&&(mat[i][j]==0)))  cont++;
        if(cont==n*n)  printf("Es matriz de identidad");
        else
              printf("No es matriz de identidad");
    printf("\n\nDeseas evaluar otra matriz? S/N  ");
    scanf("%s",&op);
   }
 while ((op=='s')||(op=='S'));
getch();
}
```

9.10.11 Calcular el producto punto de dos matrices

```
/*Producto punto de matrices*/
#include <conio.h>
#include <stdio.h>
void main ()
{
  char op='s';
  clrscr ();
  int i, j, m1[10][10], m2[10][10], m3[10][10], m, n;
  printf ("producto punto de dos matrices\n");
  while (op=='s'||op=='S')
  {
    printf ("Dame el total de filas\n");
    scanf ("%d", & m);
   printf ("Dame el total de columnas\n");
   scanf ("%d", & n);
   for (i=0; i<m; i++)
        for (j=0; j<n; j++)
      {
```

```
        printf ("Dame el elemento con coordenada %d %d de la matriz 1\n",
             i+1, j+1);
        scanf ("%d", & m1[i][j]);
        printf ("Dame el elemento con coordenada %d %d de la matriz 2\n",
             i+1, j+1);
        scanf ("%d", & m2[i][j]);
      }
 for (i=0; i<m; i++)
 {
   for (j=0; j<n; j++)
   {
     m3[i][j]=m1[i][j]*m2[i][j];
     printf ("%d", m3[i][j]);
   }
   printf ("\n");
 }
printf ("Deseas calcular otra matriz\n?");
scanf ("%s", & op);
 }
 getch ();
}
```

9.10.12 Calcular la inversa de una matriz cuadrada de 3 × 3

```
/*mat1*/
/*Inversa*/
#include <conio.h>
#include <stdio.h>
void main ()
{
  int i, j, n, m[3][3], det3;
  float t[3][3], a;
  clrscr ();
  char op='s';
  printf ("El programa calcula la inversa de una matriz cuadrada de
       3x3\n");
  while (op=='s'||op=='S')
  {
    printf ("Dame la matriz 3x3\n");
```

```
  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
    {
       printf ("Dame el componente %d %d de la matriz\n", i+1, j+1);
       scanf ("%d", & m[i][j]);
     }
 det3=(m[0][0]*m[1][1]*m[2][2])+(m[0][1]*m[1][2]*m[2][0])+(m[0][2]*m[1]
      [0]*m[2][1])-(m[0][1]*m[1][0]*m[2][2])-(m[0][0]*m[1][2]*m[2][1])-
      (m[0][2]*m[1][1]*m[2][0]);
 a=(1/det3);
 printf ("El determinante es %d\n", det3);
for (i=0; i<3; i++)
{
  for (j=0; j<3; j++)
  {
     t[i][j]=a*m[i][j];
     printf ("%f", t[i][j]);
   }
    printf ("\n");
   }
   printf ("Deseas calcular otro determinante? s/n\n");
   scanf ("%s", & op);
  }
 getch ();
}
```

9.10.13 Calcular la inversa de una matriz cuadrada

```
/*Inversa*/
#include <conio.h>
#include <stdio.h>
void main ()
{
  int i, j, n, m[2][2], det2;
  float a, b, c, d, e;
  clrscr ();
  char op='s';
  printf ("_El programa calcula la inversa de una matriz cuadrada\n");
  while (op=='s'||op=='S')
```

```
{
   printf ("Dame la matriz 2x2\n");
   for (i=0; i<n; i++)
     for (j=0; j<n; j++)
     {
       printf ("Dame el componente %d %d de la matriz\n", i+1, j+1);
       scanf ("%d", & m[i][j]);
     }
    det2=(m[0][0]*m[1][1])-(m[0][1]*m[1][0]);
    a=1/det2;
    b=-a*m[1][1];
    c=a*m[0][1];
    d=a*m[1][0];
    e=-a*m[0][0];
   gotoxy (4,30);
   printf ("%f", a+b+c+d+e);
   printf ("Deseas calcular otro determinante? s/n\n");
   scanf ("%s", & op);
     }
  getch ();
}
```

9.10.14 Calcular la raíz cuadrada de una matriz

```
/*Raiz de matriz*/
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main ()
{
  clrscr ();
  float x[20][20], y[20][20];
  int i,j,f,c;
  printf ("El programa te da una matriz y la raiz de la misma\n");
  printf ("Dame el numero de filas\n");
  scanf ("%d", & f);
  printf ("Dame el numero de columnas\n");
  scanf ("%d", & c);
  for (i=0; i<f; i++)
```

```
{
  printf ("Lectura de la fila %d de la matriz a: \n", i);
  for (j=0; j<c; j++)
  {
    printf ("a (%d, %d)=", i, j);
    scanf ("%f", & x[i][j]);
    y[i][j]= sqrt (x[i][j]);
   }
  }
printf ("Matriz original\n");
for (i=0; i<f; i++)
{
  for (j=0; j<c; j++) printf ("%5.2f", x[i][j]);
  printf ("\n");
 }
printf ("Matriz raiz cuadrada\n");
for (i=0; i<f; i++)
{
   printf ("%5.2f", y[i][j]);
   printf ("\n");
  }
 getch ();
}
```

9.11 Funciones sin paso de parámetros

9.11.1 Calcular el costo de los artículos manufacturados por la empresa Kenner

```
/*Empresa Kenner*/
#include<stdio.h>
#include<conio.h>
#define mod 150000
void cost()
{
 float md,cim,ti,tf,cam;
 printf("Dame el costo de materiales directos usados en la produccion\n");
 scanf("%f",&md);
```

```
printf("Dame el costo indirecto de manufactura\n");
scanf("%f",&cim);
printf("Dame el costo de trabajo inicial en proceso\n");
scanf("%f",&ti);
printf("Dame el costo del trabajo final en proceso\n");
scanf("%f",&tf);
cam=(md+mod+cim+ti)-tf;
printf("El costo de articulos manufacturados es=$%f",cam);
}
void main()
{
      clrscr();
      printf("costo de los articulos manufacturados de la empresa
          Kenner\n");
      cost();
      getch();
}
```

9.11.2 Calcular los costos unitarios de la empresa Gelstrap

```
/*Empresa Gelstrap*/
#include<stdio.h>
#include<conio.h>
void md()
{
  float md,total;
  printf("Dame el costo de materiales directos(mas de 9000)\n");
  scanf("%f",&md);
  total=md/9000;
  printf("El costo unitario de materiales directos es $%.3f",total);
}
void cmd()
{
  float mod,total;
  printf("Dame el costo de mano de obra directa(mas de 9000)\n");
  scanf("%f",&mod);
  total=mod/9000;
  printf("El costo unitario de mano de obra directa es $%.3f",total);
```

```
}
void ci()
{
  float cin,total;
  printf("Dame el total de costos indirectos(mas de 9000)\n");
  scanf("%f",&cin);
  total=cin/9000;
  printf("El costo unitario de costos indirectos es $%.3f",total);
}
void main()
{
      int op;
      clrscr();
      printf("\nCalcule los costos unitarios de la empresa Gelstrap\n");
      printf("\nQue costo unitario deseas calcular?\n");
      printf("1)Materiales directos\n2)Mano de obra\n3)Costos
          indirectos\n");
      scanf("%d",&op);
      switch(op)
       {
            case 1: md();
                        break;
            case 2: cmd();
                         break;
            case 3: ci();
                        break;
            default:
                        printf("Opcion invalida");
       }
      getch();
}
```

9.11.3 Calcular la molaridad de una solución

```
/*Cálculo de la molaridad*/
#include<stdio.h>
#include<conio.h>
void molaridad()
{
```

```
  float M,n,L;
  printf("\n \n Introduzca los moles: ");
  scanf("%f",&n);
  printf("\n Ahora el volumen en litros: ");
  scanf("%f",&L);
      M=n/L;
  printf("La molaridad es: %f",M);
 }
void main()
{
  char op;
  do
  {
    gotoxy(26,3);
    printf("Calculo de la molaridad (M)");
    molaridad();
    printf("\n\n Quiere calcular otra molaridad? S/N  ");
    scanf("%s",&op);
   }
  while (op=='S');
  getch();
}
```

9.11.4 Calcular el porcentaje de masa de una solución

```
/*Cálculo de % masa*/
#include<stdio.h>
#include<conio.h>
void masa()
{
  float pm,comp,solucion;
  printf("\n\n Introduce la masa del componente a calcular: ");
  scanf("%f",&comp);
  printf("\n Recuerda que las unidades deben ser las mismas\n Por ultimo,
       la masa de la solucion: ");
  scanf("%f",&solucion);
  pm=(comp/solucion)*100;
  printf("\n El porcentaje es: %f",pm);
```

```
 }
void main()
{
 char op;
 clrscr();
 do
 {
   gotoxy(26,3);
   printf("Calculo del %masa");
   masa();
   printf("\n\n Quieres calcular otro porcentaje? S/N  ");
   scanf("%s",&op);
  }
 while(op=='S');
 getch();
}
```

9.11.5 Calcular la normalidad de una normalidad

```
/*Cálculo de la normalidad*/
#include<stdio.h>
#include<conio.h>
void normalidad()
{
  float n,EQ,L;
  printf("\n\n Es necesario conocer el num. de equivalentes quimicos del
       componente; introduzcalos: ");
  scanf("%f",&EQ);
  printf("\n Finalmente, el volumen de la solucion en litros: ");
  scanf("%f",&L);
  n=EQ/L;
  printf("\n N=%f",n);
 }
void main()
{
 char op;
 clrscr();
 do
 {
```

```
   gotoxy(26,3);
   printf("Calculo de la normalidad");
   normalidad();
   printf("\n\n Quiere calcular otra normalidad? S/N  ");
   scanf("%s",&op);
  }
 while(op=='S');
 getch();
}
```

9.11.6 Calcular la velocidad

```
/*Velocidad*/
#include <conio.h>
#include <stdio.h>
float v, d, t;
void vel ()
{
  printf ("Dame la distancia\n");
  scanf ("%f", & d);
  printf ("Dame el tiempo\n");
  scanf ("%f", & t);
  v=d/t;
  printf ("La velocidad es %7.2fm/s\n", v);
 }
void dist ()
{
  printf ("Dame la velocidad\n");
  scanf ("%f", & v);
  printf ("Dame el tiempo\n");
  scanf ("%f", & t);
  d=t*v;
  printf ("La distancia es %7.2fm\n", d);
 }
void tiempo ()
{
  printf ("Dame la velocidad\n");
  scanf ("%f", & v);
  printf ("Dame la distancia\n");
```

```
  scanf ("%f", & d);
  t=d/v;
  printf ("El tiempo es %7.2fs\n", t);
 }
void main ()
{
  clrscr ();
  int f;
  char op='s';
  while (op=='s'||op=='S')
  {
    clrscr ();
    printf ("El programa calcula 1)la velocidad 2)la distancia 3)el
    tiempo\n");
    printf ("Que deseas calcular?\n");
    scanf ("%d", & f);
    switch (f)
    {
      case 1: printf ("Velocidad\n");
                  vel ();
           break;
      case 2: printf ("Distancia\n");
                  dist ();
           break;
      case 3: printf ("Tiempo\n");
                  tiempo ();
           break;
      default: printf ("Error de caso \n");
    }
   printf ("Deseas repetir la funcion? s/n\n");
   scanf ("%s", & op);
 }
getch ();
}
```

9.11.7 Calcular el campo eléctrico

```
/*Campo eléctrico*/
#include <conio.h>
#include <stdio.h>
```

```
float e, f, q;
void campo_ele ()
{
  printf ("Dame la fuerza\n");
  scanf ("%f", & f);
  printf ("Dame la carga\n");
  scanf ("%f", & q);
  e=f/q;
  printf ("El campo electrico es %7.2f\n", e);
  }
void fuerza ()
{
  printf ("Dame el campo electrico\n");
  scanf ("%f", & e);
  printf ("Dame la carga\n");
  scanf ("%f", & q);
  f=e*q;
  printf ("La fuerza es %7.2f", f);
 }
void carga ()
{
  printf ("Dame el campo electrico\n");
  scanf ("%f", & e);
  printf ("Dame la fuerza \n");
  scanf ("%f", & f);
  q=f/e;
  printf ("La carga es %7.2f", q);
 }
void main ()
{
  clrscr ();
  int f;
  char op='s';
  while (op=='s'||op=='S')
  {
    clrscr ();
    printf ("El programa calcula\n 1)Campo electrico\n 2)Fuerza\n
         3)Carga\n");
    printf ("Que deseas calcular?\n");
```

```
    scanf ("%d", & f);
    switch (f)
    {
       case 1: printf ("Campo electrico\n");
               campo_ele ();
               break;
      case 2: printf ("Fuerza\n");
               fuerza ();
               break;
      case 3: printf ("Carga\n");
               carga ();
               break;
      default: printf ("Error de caso \n");
     }
   printf ("Deseas repetir la funcion? s/n\n");
   scanf ("%s", & op);
  }
getch ();
}
```

9.11.8 Calcular el número de ladrillos y la cantidad de cemento necesarios para construir una pared

```
/*Costales_de_cemento*/
#include<stdio.h>
#include<conio.h>
void costales()
{
   float kg;
   int cc;
   printf("Dime el numero de kg y te indicare cuantos costales necesitas\
    ndime el numero de kg");
   scanf("%f",&kg);
   cc=kg/50;
   printf("El numero de costales es%d",cc);
  }
void ladrillos()
{
  int mc,nl;
```

```
  printf("Te indicare el numero de ladrillos que necesitas\ndime cuantos
       metros cuadrados construiras");
  scanf("%d",&mc);
  nl=mc*72;
  printf("El numero de ladrillos que necesitas es%d",nl);
 }
void main()
{
  int op,res;
  clrscr();
  printf("Dime el numero de kg de cemento y te indicare los costales que
       necesitas, o de metros cuadrados de pared y te indicare el numero
       de ladrillos\n");
  printf("Que deseas calcular?\n1 cemento\n2 ladrillos");
  scanf("%d",&op);
  if (op==1)
  {
    costales();
  }
  else
  {
    ladrillos();
  }
 getch();
}
```

9.12 Funciones con prototipo sin paso de parámetros

9.12.1 Calcular la distancia entre dos puntos

```
/* Programa de la distancia entre dos puntos con funciones sin paso de
   parámetros */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void fun_puntos();
void main()
{
```

```
char op;
 do
  {
   clrscr();
   printf("Programa para calcular la distancia entre dos puntos\n\n");
   fun_puntos();
   printf("\n\n\nDeseas repetir el programa? S/N  ");
   scanf("%s",&op);
  }
 while((op=='s')||(op=='S'));
 getch();
}

void fun_puntos()
 {
  float x1,x2,y1,y2,dist;
  printf("Primer punto\n\n");
  printf("Dame la x de la primera coordenada ");
  scanf("%f",&x1);
  printf("Dame la y de la primera coordenada ");
  scanf("%f",&y1);
  printf("\nSegundo punto\n\n");
  printf("Dame la x de la segunda coordenada ");
  scanf("%f",&x2);
  printf("Dame la y de la segunda coordenada ");
  scanf("%f",&y2);
  dist=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
  printf("\n\nLa distancia es de %.2f unidades",dist);
 }
```

9.12.2 Calcular la excentricidad de una elipse sin paso de parámetros

```
/* Programa de la excentricidad de una elipse sin paso de parámetros */
#include<conio.h>
#include<stdio.h>
#include<math.h>
void exc_fun();
void main()
```

```
{
 textcolor(WHITE);
 textbackground(BLUE);
 char op;
 do
  {
   clrscr();
   printf("Excentricidad de una elipse\n\n");
   exc_fun();
   printf("\n\nDeseas repetir el programa? S/N  ");
   scanf("%s",&op);
  }
 while((op=='s')||(op=='S'));
}
void exc_fun()
 {
   float a2,b2,a,b,e;
   printf("Dame la medida del eje mayor ");
   scanf("%f",&a2);
   printf("Dame la medida del eje menor ");
   scanf("%f",&b2);
   a=a2/2;
   b=b2/2;
   e=(sqrt(a*a-b*b))/a;
  if((a!=0)&&(b!=0)) printf("\nLa excentricidad es de %f u.",e);
   else
      printf("\nLas medidas no son validas");
 getch();
}
```

9.12.3 Calcular la derivada de *X* a la *n*

```
/* Programa de derivadas de X a la n sin paso de parámetros */
 #include<stdio.h>
 #include<conio.h>
 void der_fun();

 void main()
 {
```

```
textcolor(BLUE);
textbackground(WHITE);
char op;
do
 {
  clrscr();
  printf("Derivadas de X a la n de orden superior\n");
  der_fun();
  printf("\nDeseas repetir el programa? S/N ");
  scanf("%s",&op);
 }
while((op=='s')||(op=='S'));
}

void der_fun()
 {
 int x,n,i,acum,o;
 printf("\nDame la constante ");
 scanf("%d",&x);
 printf("Dame el exponente ");
 scanf("%d",&n);
 printf("\nDe que orden quieres calcular la derivada? ");
 scanf("%d",&o);
 for (i=1;i<=o;i++)
 {
     x*=n;
     n--;
 }
 printf("\n\nLa derivada es %dX^%d\n",x,n);
 getch();
}
```

9.13 Funciones con paso de parámetros

9.13.1 Calcular costos en la empresa Good Mark Company

```
/*Good Mark Company*/
#include<stdio.h>
```

```
#include<conio.h>
#define mon 120000
float dimpre(int dp,int mam,int man,int ins)
{
  return((mam+man+ins+mon)-(.20*(mam+man+ins+mon)));
}
int dcorte(int dp,int mam,int man,int ins)
{
  return(mam+man+ins+mon);
}
void main()
{
  int mam,man,ins,dp;
  clrscr();
  printf("Calcula el costo indirecto total de cada departamento de la
       Good Mark Company\n");
  printf("De cual departamento necesitas saber el total de su costo
       indirecto?\n");
  printf("1)departamento de corte\n2)departamento de impresion\n");
  scanf("%d",&dp);
  printf("Cuanto fue del costo de manejo de materiales?\n");
  scanf("%d",&mam);
  printf("Cuanto se invirtio de manufactura?\n");
  scanf("%d",&man);
  printf("Cuanto fue del costo de inspeccion?\n");
  scanf("%d",&ins);
  if (dp==1) printf("El costo indirecto del departamento de corte es
       $%d",dcorte(dp,mam,man,ins));
  if (dp==2) printf("El costo indirecto del departamento de impresion es
       $%f",dimpre(dp,mam,man,ins));
  getch();
}
```

9.13.2 Calcular los costos unitarios de la empresa Gelstrap

```
/*Empresa Gelstrap*/
#include<stdio.h>
#include<conio.h>
float mdi(float md)
```

```
{
  return(md/9000);
 }
float maobra(float mod)
{
  return(mod/9000);
}
float cosind(float cin)
{
  return(cin/9000);
}
void main()
{
   int op;
   float total,md,mod,cin;
   clrscr();
   printf("\nCalcule los costos unitarios de la empresa Gelstrap\n");
   printf("\nQue costo unitario deseas calcular?\n");
 printf("1)materiales directos\n2)mano de obra\n3)costos indirectos\n");
 scanf("%d",&op);
 switch(op)
{
    case 1: printf("Dame el costo de materiales directos(mas de
 9000)\n");
                scanf("%f",&md);
               printf("El costo unitario de materiales directos es
 :%.3f",mdi(md));
               break;
    case 2: printf("Dame el costo de mano de obra directa(mas de
       9000)\n");
                   scanf("%f",&mod);
                   printf("El costo unitario de mano de obra directa
                        es $%.3f",maobra(mod));
                   break;
     case 3:  printf("Dame el total de costos indirectos (mas de
         9000)\n");
                  scanf("%f",&cin);
                  printf("El costo unitario de costos indirectos es
                       $%.3f",cosind(cin));
```

```
                    break;
          default:
                    printf("Opcion invalida");
        }
      getch();
}
```

9.13.3 Calcular la excentricidad de una elipse

```
/* Programa de la excentricidad de una elipse con paso de parámetros */
 #include<conio.h>
 #include<stdio.h>
 #include<math.h>
 float exc_fun(float x,float y);
 void main()
 {
   textcolor(WHITE);
   textbackground(BLUE);
 char op;
 float a2,b2,resul;
 do
 {
  clrscr();
  printf("Excentricidad de una elipse\n\n");
  printf("Dame la medida del eje mayor ");
  scanf("%f",&a2);
  printf("Dame la medida del eje menor ");
  scanf("%f",&b2);
  resul=exc_fun(a2,b2);
  printf("\nLa excentricidad es de %.2f u.",resul);
  printf("\n\nDeseas repetir el programa? S/N  ");
  scanf("%s",&op);
 }
 while((op=='s')||(op=='S'));
 getch();
}
float exc_fun(float x,float y)
{
  float a,b,e;
```

```
    a=x/2;
    b=y/2;
    e=sqrt(a*a-b*b)/a;
    if ((a!=0)&&(b!=0))  return e;
    else
         printf("Medidas no validas\n");
    }
```

9.13.4 Calcular derivadas de *X* a la *n*

```
/* Programa de derivadas de X a la n con paso de parámetros */

#include<stdio.h>
#include<conio.h>
int deriv_cons(int x,int n,int o);
int deriv_exp(int x,int n,int o);
void main()
{
 int x,n,o,resul1,resul2;
 char op;
 do
  {
   clrscr();
   printf("Derivadas de X a la n de orden superior\n");
   printf("\nDame la constante: ");
   scanf("%d",&x);
   printf("Dame el exponente: ");
   scanf("%d",&n);
   printf("\nDe que orden quieres calcular la derivada? ");
   scanf("%d",&o);
   resul1=deriv_cons(x,n,o);
   resul2=deriv_exp(x,n,o);
   printf("\nLa derivada es: %d X^%d\n",resul1,resul2);
   printf("\nDeseas repetir el programa? S/N  ");
   scanf("%s",&op);
  }
 while((op=='s')||(op=='S'));
 getch();
}
```

```
int deriv_cons(int x,int n,int o)
 {
  int i;
   for(i=1;i<=o;i++)
     {
      x*=n;
      n--;
     }
   return x;
 }

int deriv_exp(int x,int n,int o)
 {
  int i;
   for(i=1;i<=o;i++)
     {
      x*=n;
      n--;
     }
   return n;
  }
```

9.13.5 Calcular velocidad, tiempo y distancia

```
/*Velocidad*/
#include <conio.h>
#include <stdio.h>
float vel (float a, float b)
{
return a/b;
}
float tiempo (float a, float b)
{
return a/b;
}
float dist (float a, float b)
{
return a*b;
}
void main ()
```

```
{
  clrscr ();
  float v, d, t;
  int f;
  char op='s';
  while (op=='s'||op=='S')
  {
    printf ("Dame la funcion que deseas llevar a cabo\n1)velocidad\n2)
         tiempo\n3)distancia\n");
    scanf ("%d", & f);
    switch (f)
    {
      case 1: printf ("velocidad\n");
                   printf ("Dame la distancia\n");
              scanf ("%f", & d);
              printf ("Dame el tiempo\n");
              scanf ("%f", & t);
              v = vel (d,t);
              printf ("La velocidad es %7.2fm/s\n", v);
              break;
     case 2: printf ("tiempo\n");
                  printf ("Dame la distancia\n");
              scanf ("%f", & d);
              printf ("Dame la velocidad\n");
              scanf ("%f", & v);
              t= tiempo (d,v);
              printf ("El tiempo es %7.2fs\n", t);
              break;
      case 3: printf ("distancia\n");
              printf ("Dame la velocidad\n");
              scanf ("%f", & v);
              printf ("Dame el tiempo\n");
              scanf ("%f", & t);
              d=dist (v,t);
              printf ("La distancia es %7.2fm\n", d);
              break;
      default: printf ("Error de caso\n");
    }
  printf ("Deseas calcular otra funcion?\n");
```

```
    scanf ("%s", & op);}
   getch ();
}
```

9.13.6 Calcular fuerza, masa y aceleración

```
/*Fuerza*/
#include <conio.h>
#include <stdio.h>
float fuerza (float a, float b)
{
  return a*b;
}
float masa (float a, float b)
{
  return a/b;
}
float aceleracion (float a, float b)
{
  return a/b;
}
void main ()
{
  clrscr ();
  float F, m, a;
  int f;
  char op='s';
  while (op=='s'||op=='S')
  {
    printf ("Dame la funcion que deseas llevar a cabo\n1)Fuerza\n2)masa\n3)
        aceleracion\n");
    scanf ("%d", & f);
    switch (f)
   {
     case 1: printf ("Fuerza\n");
                  printf ("Dame la masa\n");
           scanf ("%f", & m);
           printf ("Dame la aceleracion\n");
           scanf ("%f", & a);
```

```
        F = fuerza (m,a);
        printf ("La fuerza es %7.2fN\n", F);
        break;
case 2: printf ("masa\n");
                printf ("Dame la fuerza\n");
           scanf ("%f", & F);
           printf ("Dame la aceleracion\n");
           scanf ("%f", & a);
           m= masa (F,a);
           printf ("La masa es %7.2fm\n", m);
           break;
case 3: printf ("aceleracion\n");
                printf ("Dame la fuerza\n");
           scanf ("%f", & F);
           printf ("Dame la masa\n");
           scanf ("%f", & m);
           a=aceleracion (F,m);
           printf ("La aceleracion es %7.2fm/s2\n", a);
                break;
                   default: printf ("Error de caso\n");
      }
    printf ("Deseas calcular otra funcion?\n");
    scanf ("%s", & op);
   }
   getch ();
}
```

9.13.7 Determinar si un compuesto es alcano, alqueno o alquino

```
/*Alcano alqueno o alquino*/
#include <conio.h>
#include <stdio.h>
int c, h, a1, a2, a3;
float alcano (float c)
{
 return  2*c+2;
}
float  alqueno (float c)
{
```

```
 return 2*c;
}
float alquino (float c)
{
 return 2*c-2;
}
void main ()
{
  clrscr ();
  int h, x;
  char op='s';
  while (op=='s'||op=='S')
  {
    clrscr ();
    printf ("El programa indica si el compueso es un alcano, alqueno y
         alquino\n");
    printf ("Dame el numero de carbonos\n");
    scanf ("%d", & x);
    printf ("Dame el numero de hidrogenos\n");
    scanf ("%d", & h);
   alcano (x);
   alqueno (x);
   alquino (x);
   if (h==alcano (x)) printf ("Es un alcano\n");
   else
        if (h==alqueno (x)) printf ("Es un alqueno\n");
       else
            if (h==alquino (x)) printf ("Es un alquino\n");
            else
                 printf ("Error, no pertenece a ningun grupo\n");
    printf ("Deseas repetir la funcion? s/n\n");
    scanf ("%s", & op);
   }
  getch ();
}
```

9.13.8 Calcular resistencia, amperaje o voltaje

```
/*Resistencias*/
#include <conio.h>
```

```
#include <stdio.h>
float resistencia (float a, float b)
{
 return a/b;
}
float voltaje  (float a, float b)
{
 return a*b;
}
float amperaje (float a, float b)
{
 return a/b;
}
void main ()
{
  clrscr ();
  float r, v, a;
  int f;
char op='s';
while (op=='s'||op=='S')
{
   printf ("Dame la funcion que deseas llevar a cabo\n1)resistencia\n2)
  voltaje\n3)amperaje\n");
 scanf ("%d", & f);
 switch (f)
 {
    case 1: printf ("resistencia\n");
         printf ("Dame el voltaje\n");
         scanf ("%f", & v);
         printf ("Dame el amperaje\n");
         scanf ("%f", & a);
         r = resistencia (r,a);
                 printf ("La resistencia es %7.2f Ohms\n", r);
         break;
    case 2: printf ("voltaje\n");
                 printf ("Dame la resistencia\n");
         scanf ("%f", & r);
         printf ("Dame el amperaje\n");
         scanf ("%f", & a);
```

```
            v= voltaje (r,a);
           printf ("El voltaje es %7.2 Volts\n", v);
           break;
      case 3: printf ("amperaje\n");
                    printf ("Dame la resistencia\n");
           scanf ("%f", & r);
           printf ("Dame el voltaje\n");
           scanf ("%f", & v);
           a=amperaje (r,v);
           printf ("El amperaje es %7.2f Amperes\n", a);
           break;
      default: printf ("Error de caso\n");
    }
  printf ("Deseas calcular otra funcion?\n");
  scanf ("%s", & op);}
 getch ();
}
```

9.13.9 Calcular campo eléctrico, fuerza y carga

```
/*Resistencias*/
#include <conio.h>
#include <stdio.h>
float campo_ele (float a, float b)
{
 return a/b;
}
float fuerza  (float a, float b)
{
 return a*b;
}
float carga (float a, float b)
{
 return a/b;
}
void main ()
{
  clrscr ();
  float e, f, q;
```

```
int g;
char op='s';
while (op=='s'||op=='S')
{
 printf ("Dame la funcion que deseas llevar a cabo\n1)campo electrico\
 n2)fuerza\n3)carga\n");
scanf ("%d", & g);
switch (g)
{
  case 1: printf ("campo electrico\n");
                printf ("Dame la fuerza\n");
        scanf ("%f", & f);
        printf ("Dame la carga\n");
        scanf ("%f", & q);
                e = campo_ele (f,q);
        printf ("El campo electrico es %7.2fN\n", e);
                break;
  case 2: printf ("fuerza\n");
                printf ("Dame el campo electrico\n");
        scanf ("%f", & e);
        printf ("Dame la carga\n");
        scanf ("%f", & q);
        f= fuerza (e,q);
        printf ("La fuerza es %7.2fm\n", f);
        break;
    case 3: printf ("carga\n");
                printf ("Dame el campo electrico\n");
        scanf ("%f", & e);
        printf ("Dame la fuerza\n");
        scanf ("%f", & f);
                q=carga (e,f);
        printf ("La carga es %7.2fm/s2\n", q);
        break;
     default: printf ("Error de caso\n");
    }
  printf ("Deseas calcular otra funcion?\n");
  scanf ("%s", & op);}
  getch ();
}
```

9.14 Funciones con arreglos

9.14.1 Mostrar el inventario de una librería

```
/*Inventario de los libros más vendidos*/
#include<stdio.h>
#include<conio.h>
float cd(int c[4])
{
   int t=0,i;
   for(i=0;i<4;i++)     t=t+c[i];
   c[i]=t;
   return(c[i]*350);
 }
float se(int c[4])
{
  int t=0,i;
  for(i=0;i<4;i++)  t=t+c[i];
  c[i]=t;
  return(c[i]*250);
}
float tre(int c[4])
{
  int t=0,i;
  for(i=0;i<4;i++) t=t+c[i];
  c[i]=t;
  return(c[i]*190);
}
void main()
{
  int op,c[4],i;
  clrscr();
  printf("\n\t\tInventario de libros mas vendidos\n");
  printf("\n\t\tElija su libro  (Ingrese el codigo):\n\n");
  printf("\t\tLibro\t\t\tCodigo\tPrecio \n\n");
  printf("\t\tEl codigo da vinci..... 1.....$350\n");
  printf("\t\tEl secreto..... 2.....$250\n");
  printf("\t\tLa tregua..... 3.....$190\n");
  scanf("%d",&op);
```

```
switch(op)
{
  case 1: for(i=0;i<4;i++)
                {
                  printf("Cuantos libros se vendieron en la semana?
                       %d\n",i+1);
                  scanf("%d",&c[i]);
                 }
                printf("Las ventas del mes son $%.3f pesos",cd(c));
                       break;
  case 2: for(i=0;i<4;i++)
                {
                  printf("Cuantos libros se vendieron en la semana?
                       %d\n",i+1);
                  scanf("%d",&c[i]);
                 }
                printf("Las ventas del mes son $%.3fpesos",se(c));
                break;
     case 3: for(i=0;i<4;i++)
                    {
                      printf("Cuantos libros se vendieron en la semana?
                           %d\n",i+1);
                      scanf("%d",&c[i]);
                    }
                  printf("Las ventas del mes son $%.3f pesos",tre(c));
                  break;
     }
    getch();
}
```

9.14.2 Calcular el salario de un trabajador en consideración de las piezas elaboradas

```
/*Pago por cada pieza elaborada*/
#include<stdio.h>
#include<conio.h>
float ma(int pie[100],float pa[100])
{
    float sue=0,s;
```

```
        s=pa[100]*pie[100];
        sue=sue+s;
        return(sue);
}
float me(int pie[100],float pa[100])
{
        float sue=0,s;
        s=(pa[100]*pie[100])+((pa[100]*pie[100])*.80);
        sue=sue+s;
        return(sue);
}
void main()
{
    int pie[100],tra,i;
    float pa[100],t;
    clrscr();
    printf("\n\t\tSueldo de un trabajador por cada pieza elaborada\n");
    printf("\nCuantos empleados trabajan por piezas elaboradas?\n");
    scanf("%d",&tra);
    for (i=0;i<tra;i++)
     {
        printf("\nEmpleado numero %d\n",i+1);
        printf("Cuantas piezas elaboraste(mas de 30 %%20 adicional)?\n");
        scanf("%d",&pie[100]);
        printf("Cuanto se te paga por cada pieza?\n");
        scanf("%f",&pa[100]);
        if (pie[100]<=30)
         {
            t=ma(pie,pa);
            printf("Tu salario es $%.2f\n",t);
         }
        else
         {
            t=me(pie,pa);
            printf("Tu salario es $%.2f",t);
         }
     }
    getch();
}
```

9.14.3 Calcular los gastos perdidos por piezas defectuosas

```
/*Productos defectuosos*/
#include<stdio.h>
#include<conio.h>
float costo(int p[30],float c[30],float pi)
{
    int i;
    float m,t=0,g=0,l,h;
    for (i=0;i<1;i++)
     {
       t=t+p[i];
       g=g+c[i];
       m=t*g;
       l=pi*t;
       h=m-l;
     }
    return(h);
 }
void main()
{
   int p[30],i,d;
   float c[30],pi;
   clrscr();
   printf("\n\t\tCalcula el gasto ocasionado por los defectos en un
        dia\n");
   printf("Cuanto cuesta la reparacion de la pieza?\n");
   scanf("%f",&pi);
   for (i=0;i<1;i++)
   {
     printf("\nDia %d Cuantos productos salieron defectuosos?\n",i+1);
     scanf("%d",&p[i]);
     printf("Dame el costo unitario de la pieza\n");
     scanf("%f",&c[i]);
     printf("\n\t\tEl dinero perdido por reparacion es %f",costo(p,c,pi));
   }
   getch();
}
```

9.14.4 Mostrar el inventario de refrescos más vendidos de las marcas de cola más conocidas


```
/*Inventario de los refrescos más vendidos*/
#include<stdio.h>
#include<conio.h>
int cd(int c[4])
{
  int t=0,i;
  for (i=0;i<4;i++)     t=t+c[i];
  c[i]=t;
  return(c[i]);
}
int se(int c[4])
{
  int t=0,i;
  for(i=0;i<4;i++)  t=t+c[i];
  c[i]=t;
  return(c[i]);
}
void main()
{
  int op,c[4],i;
  clrscr();
  printf("\n\t\tInventario de refrescos mas vendidos\n");
  printf("\n\tDe que refresco quiere saber el total de ventas en un mes?\n\
    n\t");
  printf("1)PEPSI\n\t2)COCA-COLA\n");
  scanf("%d",&op);
  switch(op)
  {
    case 1: for(i=0;i<4;i++)
                  {
                    printf("Cuantas PEPSIS se vendieron en la semana?
%d\n",i+1);
                    scanf("%d",&c[i]);
                  }
                printf("\n\tEn el mes se vendieron %d PEPSIS",cd(c));
                break;
```

```
    case 2: for(i=0;i<4;i++)
                  {
                    printf("Cuantas coca-colas se vendieron en la semana
%d\n",i+1);
                    scanf("%d",&c[i]);
                  }
                  printf("\n\tEn el mes se vendieron %d COCA-COLAS",se(c));
                  break;
    }
   getch();
}
```

9.14.5 Calcular la masa molecular de un compuesto

```
/*Fórmula molecular*/
#include<stdio.h>
#include<conio.h>
void formula()
{
float pe, muestra, moles[10], porc, comp;
int e, i, sub[10];
printf("Indique el num. de elementos del compuesto (maximo 10)");
scanf("%i",&e);
printf("Dame la masa de muestra del compuesto");
scanf("%i",&muestra);
for (i=0;i<e;i=i+1)
{
 printf("Dame el peso atomico del elemento %i", i+1);
 scanf("%i",&pe);
 printf("Ahora la cantidad del compuesto (en % masa)");
 scanf("%i",&porc);
 moles[i]=((porc/100)*muestra)/pe;
}
comp=moles[0];
for (i=1;i<e;i=i+1)
{
 if (comp<moles[i]) comp=moles[i];
 else
  comp=comp;
```

```
 }
 for (i=0;i<e;i=i+1) sub[i]=moles[i]/comp;
 printf("Los subindices en la formula del compuesto son:");
 for (i=0;i<e;i=i+1) printf("Elemento %i=%i",i+1,sub[i]);
}
void main()
{
 printf("Obtencion de la formula molecular de algun compuesto");
 formula();
 getch();
}
```

9.14.6 Calcular el reactivo limitante de un elemento

```
/*Reactivo limitante*/
#include<stdio.h>
#include<conio.h>
void rl()
{
 float reac,pm,moles[15],comp;
 int c,i,elemento;
 printf("Indique cuantos compuestos tiene la reaccion (no mas de 15)");
 scanf("%i",&c);
 for (i=0;i<c;i=i+1)
 {
  printf("Dame la cantidad del reactivo %i", i+1);
  scanf("%f",&reac);
  printf("Dame el peso molecular del reactivo");
  scanf("%f",&pm);
  moles[i]=reac/pm;
 }
 comp=moles[0];
 for (i=1;i<c;i=i+1)
 {
  if (comp<moles[i])
  {
   comp=moles[i];
   elemento=i;
  }
```

```
 }
 printf("El reactivo limitante es %i", elemento);
}
void main()
{
  printf("Reactivo limitante");
  rl();
  getch();
}
```

9.14.7 Calcular la magnitud de un vector

```
/*Cálculo de la magnitud de un vector*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void mag()
{
 int e,i,a[20],aa[20],AA;
 printf("De cuantos elementos es el vector (maximo 20)?");
 scanf("%i",&e);
 for (i=0;i<e;i=i+1)
 {
  printf("Dame el valor del elemento %i del primer vector",i+1);
  scanf("%i",&a[i]);
 }
 for (i=0;i<e;i=i+1) aa[i]=a[i]*a[i];
 AA=aa[0];
 for(i=1;i<e;i=i+1) AA=AA+aa[i];
 printf("%i",AA);
}
void main()
{
 printf("Magnitud al cuadrado de un vector");
 mag();
 getch();
}
```

9.14.8 Calcular el producto cruz de un vector


```
/*Producto cruz*/
#include<stdio.h>
#include<conio.h>
void pc()
{
  int i,a[3],b[3],ab[3],AB;
  printf("Recuerda que son de tres elementos");
  for (i=0;i<3;i=i+1)
  {
    printf("Dame el valor del elemento %i del primer vector",i+1);
    scanf("%i",&a[i]);
  }
  for (i=0;i<3;i=i+1)
  {
    printf("Dame el valor del elemento %i del segundo vector",i+1);
    scanf("%i",&b[i]);
  }
  ab[0]=(a[2]*b[3])-(b[2]*a[3]);
  ab[1]=(a[1]*b[3])-(b[1]*a[3]);
  ab[2]=(a[1]*b[2])-(b[1]*a[2]);
  AB=ab[0];
  for (i=1;i<3;i=i+1) AB=AB+ab[i];
  printf("%i",AB);
 }
 void main()
 {
  printf("Producto cruz de dos vectores");
  pc();
  getch();
}
```

ÍNDICE DE EJERCICIOS RESUELTOS

ÍNDICE DE EJEMPLOS

ÍNDICE DE TABLAS

Este libro fue creado teniendo en mente a aquellos estudiantes que desean adentrarse en el mundo de la programación utilizando el lenguaje de C. El contenido está considerado para desarrollarse en un primer curso.

El método de enseñanza se basa en el análisis de un problema y la descripción de los pasos necesarios para llegar a la solución; el objetivo principal es mostrar con detalle cómo crear un programa y cómo entenderlo.

Todo esto se complementa con ejemplos y ejercicios resueltos, desglosados en cinco partes para una mejor comprensión. También se presenta la descripción de las operaciones, los datos, la codificación y la ejecución del código, además de una explicación detallada del procedimiento.

Los primeros capítulos presentan los fundamentos de la programación en C, la aritmética y la programación estructurada. Luego se muestran las estructuras de control, algunos datos estructurados y funciones, para finalizar con una serie de ejercicios resueltos.

Cada tema se trata con suficiente profundidad y detalle para ser entendido por un estudiante que se inicia en esta área de la computación.

Para mayor información visite la página Web:
www.pearsoneducacion.net/marquez

Prentice Hall
es una marca de

PEARSON

Visítenos en:
www.pearsoneducacion.net

ISBN 978-607-32-0600-6
90000
9 786073 206006