®

Introducción a la Programación Estructurada en C

Teresa Gabriela Marquez Frausto

Sonia Osorio Angel

Agosto de 2008

Indice

Capitulo 1

Introducción al lenguaje C.

Taller de programación estructurada

1. Introducción al lenguaje C

Este es un primer curso de programación estructurada, por lo que se repasarán algunos conceptos de las ciencias de la computación, para luego proceder a describir el contenido de un programa y finalmente la estructura del mismo.

Una **computadora**, es un dispositivo electrónico que procesa información (programa), la computadora esta formado por una unidad de entrada (usualmente el teclado), una unidad de salida (puede ser el monitor), la unidad central de procesamiento (o cpu) y la memoria (también llamada memoria interna o principal).

Es importante recordar que la computadora maneja dos tipos de memoria, interna o principal y memoria externa o secundaria, la interna es la utilizada por el procesador y la memoria externa es la que se usa para almacenar información de manera permanente.

En la memoria interna se almacenan los datos y el programa en lenguaje máquina que ejecutará el CPU, al programa en este estado se le llama **proceso**.

La memoria externa o secundaria es cualquier dispositivo de almacenamiento como el diskette, el CD, el DVD, el disco óptico, la memoria flash, etc., y guarda archivos o información.

1.1 Lenguajes utilizados en una computadora

Lenguaje de alto nivel. Son los programas llamados generalmente lenguajes de programación. Con ellos se escribe en un lenguaje similar al nuestro, aunque la mayoría de estos se encuentran en inglés. Tienen la ventaja de ser portables o sea que diferentes tipos de computadoras los entienden y ejecutan.

Estos lenguajes pueden ser compiladores o intérpretes.

Un **intérprete** toma una instrucción del programa la traduce a lenguaje máquina y la ejecuta, y hace esto con cada una de las sentencias del programa, el intérprete más popular es Basic.

Un **compilador** toma las sentencias, y las traduce a lenguaje máquina, creando un archivo con todo el programa traducido o sea el programa objeto. 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 maquina, es un programa escrito en ceros y unos, es un lenguaje muy lejano a nuestra forma de expresión, pero es el único que entiende el procesador.

Un **programa** es un conjunto de instrucciones con el objetivo de llegar a un fin, las instrucciones están escritas en algún lenguaje de computadora, el programa luego se traduce a lenguaje máquina, que es el único lenguaje que entiende la computadora, y finalmente se carga, en la memoria principal de la computadora (memoria ram), para ser ejecutado por el procesador, el resultado son las acciones para lo que fue escrito el programa.

Hay dos grandes tipos de programas los de **aplicación** y los de **sistemas**. Los primeros son los que se conocen comúnmente y que se utilizan directamente en la computadora como son los procesadores de palabras y las hojas de calculo. Los segundos (los de sistemas) son un conjunto de programas que permiten que exista una comunicación sencilla entre el usuario y la computadora, entre ellos están los sistemas operativos, los compiladores etc.

Programador es el término que se usa para denotar a la persona que se dedica a escribir programas, frecuentemente son equipos de personas que en conjunto realizan un programa.

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

1.2 Etapas de desarrollo del software

El desarrollo y construcción de un programa regularmente sigue los siguientes pasos:

1) Análisis
2) Programación
3) Codificación
4) Prueba
5) Mantenimiento.
6) Documentacion

El *análisis y diseño* depende del tamaño y finalidad del programa, la *programación* se realiza generalmente en pseudocódigo (para nuestro caso en español), para después *codificar* el programa en lenguaje C, una vez escrito en lenguaje C se *compila*, y se resuelven los errores de sintaxis, si no hay errores se procede a la ejecución donde se *prueba y se verifica* si se obtiene lo que se planeó. Finalmente hechos los ajustes un programa puede requerir de *mantenimiento* para realizar cambios o ajustes relativamente pequeños.

La compilación consiste en convertir el programa escrito en C a lenguaje maquina, C utiliza un enlazador de bibliotecas (linker), al finalizar se crea un archivo ejecutable, y es el que se utilizara para ejecutar el programa.

1.3 El Lenguaje C: Historia y Características

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

El lenguaje C es un lenguaje de alto nivel, aunque también se utiliza para la programación de sistemas, es una evolución de los lenguajes B y BCPL, los cuales carecian de la capacidad de manejar tipos de datos, lo que era una desventaja para el programador, estos tres lenguajes fueron explotados 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 por Dennis Ritchie, se inicia formalmente la utilización 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 Nacional 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.

1.4 Elementos de un programa en C (conceptos básicos)

Un programa contiene varios elementos, no es obligatorio usarlos siempre todos, a continuación se mencionan los mas usuales.

1.4.1 Identificadores

En un programa siempre se manipularán diversos elementos, los cuales son creados por el programador, por ejemplo variables, constantes, funciones, etc., o bien creados junto con el lenguaje, como la función printf. Cada uno de estos elementos necesita un nombre único, con el cual se diferencie de los demás. A dichos nombres se les llama identificadores, que como la misma palabra lo indica, identifica a un objeto de otros usados en el mismo programa.

En C se siguen ciertas reglas para formar los identificadores, y se enlistan a continuación:

a) Un identificador se forma a partir de dígitos, letras y caracter de subrayado (guión bajo), no se puede utilizar ningún otro carácter.

b) El primer caracter de un identificador siempre debe ser una letra, y aunque también es permitido utilizar el guión bajo como primer carácter, no es muy común usarlo, más bien se utiliza para formar identificadores con más de una palabra.

c) No se puede utilizar un dígito como primer carácter en un identificador.

d) 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 lo más expresivos posible.

e) Se pueden utilizar mayúsculas y minúsculas en un identificador, pero eso sí, se deberá usar consistentemente, ya que el lenguaje C es sensible a mayúsculas y minúsculas, es decir, una ´a´ es diferente a una ´A´.

f) No se pueden utilizar las palabras reservadas de C como identificadores.

Ejemplos de identificadores válidos.

A	Nombre	Nombre_alumno	X1
a	_codigo	CODIGO	resultado_3

Ejemplos de identificadores no válidos y las razones

Identificador	Explicacion
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

1.4.2 Tipos de datos

En un programa siempre se van a procesar datos, los cuales pueden ser de distinta naturaleza o tipo, y dependiendo de esto, es como se representará y se almacenará dicho dato en la memoria de la computadora, es decir, según el tipo de dato, será la cantidad de memoria que se requerirá para almacenarlo. A continuación se enlistan los tipos de datos básicos de C, así como la cantidad de memoria que se requiere para cada uno. Cabe señalar que estas cantidades son las más usuales, aunque pueden variar de un compilador a otro; además el rango de algunos de estos tipos de datos básicos pueden variar si se utilizan los calificadores de tipo como short, long, signed, unsigned, .

Tipo	Descripción	Cantidad de memoria requerida	Rango
int	Valor numérico entero (sin fracción)	2 bytes	-32767 a 32768
unsigned int	Entero sin signo	2 bytes	0 a 65,535
short int	Entero corto	2 bytes	-32767 a 32768
float	Número en coma flotante (con fracción y/o exponente)	4 bytes	$3.4x10^{-38}$ a $3.4x10^{+38}$
double	Número en coma flotante de doble precisión (más cifras significativas para la fracción o mayor para el	8 bytes	$1.7x10^{-308}$ a $1.7x10^{+308}$

	exponente)		
long	Entero largo	4 bytes	-2,147,483,648 a 2,147,483,647
unsigned long	Largo sin signo	4 bytes	0 a 4,294,967,295
long double	Doble largo	10 bytes	$3.4x10^{-4932}$ a $1.1x10^{+4932}$
char	Carácter	1 byte	-128 a 127
unsigned char	Carácter sin signo	1 byte	0 a 255

1.4.3 Variables

Para que un programa pueda ser ejecutado, las instrucciones junto con los datos deberán estar almacenados en la memoria, y muchas veces, dichos datos serán proporcionados por el usuario del programa (que no necesariamente es el programador), o bien serán el resultado del procesamiento de otros datos. Es decir, tal vez no se conozca de antemano el valor de algunos de esos datos. ¿Cómo entonces se puede almacenar información que no se conoce desde el inicio? Con el uso de variables.

Una variable es un espacio en la memoria que el programador reserva con el fin de guardar esos datos que son "desconocidos" cuando empieza la ejecución de un programa o bien que pueden ir cambiando durante el procesamiento. Para poder reservar tantos espacios como se requieran se tiene que hacer la declaración de variables

Declaración de variables

La declaración consiste en reservar los espacios de memoria que requiere el programa para su ejecución, y para esto se tendrá que especificar el tipo de dato, así como el identificador con que se le hará referencia posteriormente.

Sintaxis para la declaración de variables en C.

<Tipo_dato> <identificador>

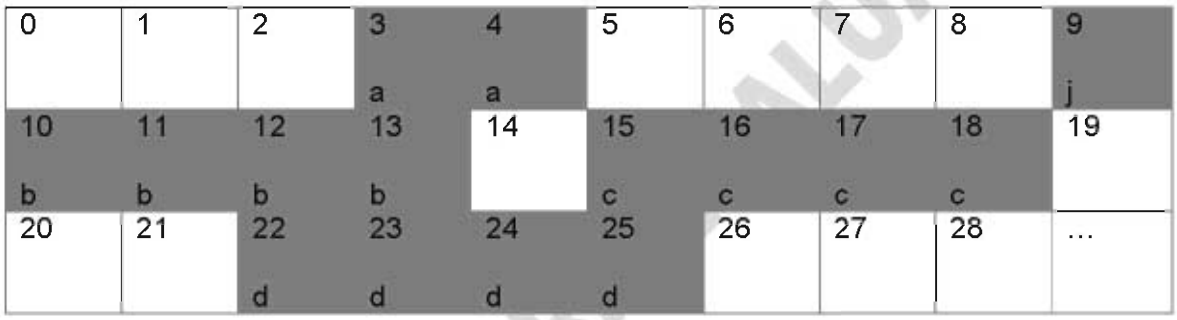
Ejemplos:

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

Reserva de memoria

Recuérdese que la memoria es un conjunto de celdas direccionables, es decir que se puede tener acceso a cada una de las celdas mediante su dirección y que en dichas celdas se pueden almacenar datos.

Cuando se declara una variable en un compilador como C, el programador no necesita conocer cual es la dirección absoluta de la celda o las celdas en las que va a guardar 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) como quedaría la memoria luego de haber declarado las variables de los ejemplos anteriores, gráficamente:



Suponiendo que cada celda midiera un byte, los espacios sombreados representan la reserva de memoria que se ha hecho de acuerdo a 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 18 para `c` y de la 22 a la 25 para `d`; finalmente se reservó la celda 9 (un byte por ser caracter) para `j`.

Nótese que se asignaron los espacios arbitrariamente, esto es porque en la realidad no necesariamente las variables quedarán de manera consecutiva en la memoria, lo que sí es importante es que exista espacio suficiente para las variables que se van 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 guardar en él algún valor que sabemos que puede cambiar durante el procesamiento del programa.

1.4.4 Constantes

Las constantes son elementos frecuentemente utilizados en los programas, y si ya se entendió el concepto de variable, será fácil entender también lo que es una constante. Una constante es un espacio en memoria que recibe un valor por primera vez y generalmente no se modificada 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, como 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 usualmente maneja el valor de 3.1416 y que no puede tener otro; entonces sería ilógico pensar en reservar un espacio para una variable si dicho valor no tiene por qué cambiar en el transcurso del programa; así que lo más conveniente sería definir una constante para almacenar el 3.1416.

Las formas para crear constantes en C, son utilizando la directiva del preprocesador **#define** (constantes simbólicas definidas) o bien la palabra reservada **const** (constantes simbólicas declaradas).

Constante simbolica #define

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

#define <identificador> <valor>

Ejemplo

```
#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.

Nótese que a las constantes creadas con ***#define***, no es necesario especificarles el tipo de dato que deberán tener, sin embargo estas constantes adquieren su tipo dependiendo de la naturaleza del valor que se les asignó.

Así pues, 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.

Palabra reservada const

Por otro lado una constante también se puede crear utilizando la palabra reservada ***const,*** si se hace de esta manera el dato si ocupa un espacio en memoria, es como si fuera una variable con la diferencia de que su contenido no cambia.

La sintaxis para declarar una constante utilizando el calificador ***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 es que ***const*** tiene un espacio de memoria reservado para cada dato; además de su sintaxis: observe que ***const*** si especifica el tipo de dato de cada valor; se utiliza el operador = para asignar dicho valor y además cada sentencia termina con ;.

1.4.5 Palabras Reservadas

Las palabras reservadas de C, son aquellas cuyo significado está predefinido en el lenguaje, esto quiere decir que ya tienen un uso específico; éstas se escriben en las sentencias o 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	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Debe tomarse en cuenta que la lista de palabras reservadas está en minúsculas y justamente así es la forma en que se deben de utilizar para que cumplan su propósito dentro de un programa. Por otra parte si se utiliza cualquiera de estas palabras en mayúsculas, bien podría utilizarse como identificador, sin embargo esto no se recomienda en la práctica.

1.4.6 Comentarios

Los comentarios son cadenas de caracteres o texto que describen partes del programa que el programador desea explicar, este texto no es parte del programa fuente, es simplemente 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 utilizarán los símbolos // (doble diagonal) si es que el comentario es de una sola línea, o bien encerrar texto entre los símbolos /*. . . texto. . . */.

1.5 Entrada y salida de datos

Un programa es un conjunto de instrucciones que ejecuta la computadora con el fin de obtener un resultado o bien la solución a un problema determinado. Casi siempre este resultado se obtendrá a partir de la manipulación de datos. Los datos muchas veces dependerán del usuario y por supuesto este tendrá que conocer los resultados del programa que está utilizando, por lo tanto se requiere contar con instrucciones que permitan capturar datos proporcionados por los usuarios 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>***.

1.5.1 Salida de datos

La computadora dispone de diversos medios para proporcionar la salida de datos, como la impresora, archivos y el más utilizado el monitor. Y 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 que se va a visualizar y que obligatoriamente requiere la función printf( ).

Por otro lado argumento o argumentos es el valor o los valores que se pretende mostrar. Los argumentos puede ser variables, constantes, expresiones aritméticas, resultados de funciones o simplemente texto que el programa debe mostrar al usuario, por ejemplo:

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

Uso de printf( )	*Descripción*
`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.

Nótese que cuando se imprime texto no es necesario utilizar alguna cadena de tipo, sólo el texto tal como se desea visualizar encerrado entre comillas(" "); pero para mostrar variables es necesario usar la cadena de tipo adecuada, también dentro de las comillas.

En la siguiente tabla se presentan las diferentes cadenas de tipo que se utilizan en C dependiendo de los tipos de datos que se desea imprimir en pantalla.

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 decimal sin signo (unsigned int)
%c	El dato es un carácter (char)
%e	El dato es un real (float)
%f	El dato es un real (float)
%g	El dato es un real (float)
%s	El dato es una cadena de caracteres seguida de un carácter nulo \0
%lf	El dato es real de doble precisión (double)

Como ya se ha mencionado, printf( ) puede manejar más de un argumento, para ello se tiene que 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, 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 que se presenta a continuación muestra las secuencias de escape que utiliza el lenguaje C, así como su significado.

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	Número octal
\xhh	Número hexadecimal
\0	Carácter nulo

ahora analiza el siguiente ejemplo e identifica cuál será la salida:

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

```
7
8.2
s
```

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

1.5.2 Entrada de datos.

La entrada de datos u operación de escritura, también se puede hacer a través de diferentes dispositivos, como por ejemplo, desde el teclado, desde un archivo, etc. pero 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 en 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

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

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; obviamente luego de ver el mensaje, el usuario tecleará 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, supóngase que el usuario teclea 1.75 que se almacena en *est*; finalmente aparece un mensaje que diría:

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

Generalmente los programas interactúan con el usuario.

Ejercicios

Describa que 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.

```
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);                    _____________________
```

Que dato se almacenaría la variable:

6.

```
int a;
// leer 3
scanf("%d",&a);      _____________
scanf("%f",&a);      _____________
scanf("%c",&a);      _____________
```

7.

```
char x;
// leer  j.
scanf("%d",&x);      _____________
scanf("%f",&x);      _____________
scanf("%c",&x);      _____________
```

8.

```
float w;
// leer 2.5
scanf("%d",&w);      _____________
scanf("%f",&w);      _____________
scanf("%c",&w);      _____________
```

Capitulo 2

Operadores.

2. Operadores

Un operador es un signo (símbolo) que indica al compilador el tipo de operación que se hará sobre los datos. C cuenta con diferentes categorías de operadores, los más elementales son operadores ***aritméticos***, ***relacionales*** y ***lógicos***, *y* los podemos encontrar prácticamente en cualquier otro lenguaje, salvo que tal vez se representen de manera distinta. Pero además de estos operadores existen algunos otros que son propios de C, que se explicarán más adelante.

2.1 Inicialización y asignación de variables

Inicializar una variable ya declarada, consiste en asignarle un valor antes de que la variable sea utilizada en un programa y 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, esto lo veremos en ejemplos más adelante.

El lenguaje C permite inicializar el valor de la variable en el momento de la declaración o bien posteriormente y 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 indica que 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*, 3 variables de tipo real, a las dos primeras se les asigna 4.56 y 7.2 respectivamente; *d* no es inicializada, ya que a ésta se le asigna el resultado de la expresión *a + b*, dando como resultado 11.76.

un ejemplo más:

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

se declara la variable *t* cuyo valor inicial es una *r;*
En los ejemplos anteriores se ha utilizado el operador = (operador de asignación) el cual en el caso del lenguaje C, no significa igualdad como normalmente estamos acostumbrados.

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 anteriormente, éste puede ser otra variable, una constante, una expresión aritmética o el resultado de una función.

Veamos 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 sentencia 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 el cual es 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 lo que utilizamos normalmente para realizar las operaciones básicas aritméticas, como 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 2 operandos (datos) para que funcionen.

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

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
/	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 modulo % 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 obsérvese 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 uno. Veamos esto en el ejemplo siguiente:

$$\begin{array}{r} 1 \leftarrow \text{(cociente)} \\ 2 \;\overline{)\,3} \\ 1 \leftarrow \text{(residuo)} \end{array}$$

Y como el operador residuo sólo maneja datos 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, pero sí obtendrá un residuo, por ejemplo, el resultado de la siguiente expresión

```
1 % 4
```

será 1, ya que

$$\begin{array}{r} 0 \\ 4 \;\overline{)\,1} \end{array}$$

1 ←(residuo)

2.2.1 Prioridad de los operadores aritméticos

Estos operadores siguen las reglas matemáticas en cuanto a precedencia o jerarquía de operadores, es decir, cuando en una expresión existen operadores diferentes, ¿Cuál se evalúa primero?

La tabla siguiente muestra esa precedencia:

( )
*, /, %
+, -

En primer lugar aparece el operador ( ), esto es debido a que en las matemáticas tradicionales se utiliza este símbolo para determinar las prioridades de operadores en expresiones grandes. En segundo lugar se evalúan la multiplicación, división y módulo, los 3 con la misma prioridad y posteriormente se evaluarán la suma y la resta.

Ejemplos 2.1

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 evalúan 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.
c) 6 / 2 + 4 * (5 – 2) 6 / 2 + 4 * 3 3 + 4 * 3 3 + 12 15	En este ejemplo, aunque la resta tiene menor prioridad, se encuentra entre paréntesis, por lo tanto se evalúa en primer lugar, posteriormente división y multiplicación de izquierda a derecha y finalmente la suma.
d) 6 + 7 * ((3 – 1) % 2)	Cuando en una expresión existan paréntesis anidados, se evaluarán de adentro hacia afuera, luego se evalúan los

Expresión	*Reglas*
6 + 7 * (2 % 2) 6 + 7 * 0 6 + 0 6	demás operadores de acuerdo a su precedencia.

NOTA: para definir prioridades de operadores C utiliza únicamente los símbolos ( ). Las { } (llaves) y [ ] (corchetes) en el lenguaje C, no se utilizan para asociar, y 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 y estos son:

+=
-=
***=**
/=

los cuales también sirven para sumar, restar, multiplicar y dividir respectivamente reduciendo las sentencias como se muestra en la siguiente tabla suponiendo que declaramos

```
int a=10;
```

EXPRESION	*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

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. Pero el lenguaje C cuenta con una serie de operadores propios de éste a los que se les llama monarios. Eso significa que únicamente requieren de un operando cuando se utilizan, lo cual proporciona ciertas ventajas.

El **operador 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 sería a `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 decrementa su valor quedando con valor de 8.

Los operadores `++` y `--` se pueden usar como prefijos o postfijos, esto es, antes o después de la variable y aunque en ambos casos se incrementa o decrementa en 1 la variable, 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 únicamente puede ser verdadero o falso.

Los operadores relacionales son:

Operador	*Descripción*

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, los operandos pueden ser variables, constantes, expresiones aritméticas o funciones, y el resultado que se obtiene de estos operadores es un valor entero, ya que verdadero es representado por un 1 y falso por el valor 0. Cuando una expresión sólo puede dar como resultado verdadero o falso, se le llama ***expresión lógica o booleana***.

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 y el operador ! Con un operando, y se requiere que éstos 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 únicamente puede ser cierto o falso.

Operador	*Descripción*
&&	***Y (and)*** El resultado de una operación ***y lógica*** sólo será verdadero si ambos operandos son verdaderos, de lo contrario, será falsa.

Operador	Descripción
||	***O (or)*** El resultado de una operación ***o lógica*** sólo 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

```
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 si 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 un número menor, el resultado será un 0.

En el siguiente ejemplo: *a=3* y *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 los :(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

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

Cabe aclarar que éstos no son los únicos operadores con que cuenta C. Este lenguaje maneja muchos otros que se utilizan para distintos propósitos, pero como es un texto de introducción, únicamente se estudiarán los mencionados.

Ejemplo 2.2

Diga el resultado de las siguientes expresiones

```
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=15-(-a-2)*2/3
X=15- -5*2/3
X=15- -10/3
X=15- -3.333
X=18.333
```

Ejercicios propuestos

I. Que 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>
```

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

4.

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

int 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;
}
```

5.

```
#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;
}
```

6.

Escriba el resultado de la siguiente operación

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

```
A+b*c/d
a % 5 % 2 + c
(a+b)*c % d
(d<65)&&(3==d)
a<=b || (3>a)
(c<=7+d) || (7>c)
(a>=d) && (2-14==c+1)
a % d % c
3+5*d % 10
a/b % 2
-a*7
(a>d) && (b<c) || (d>5)
((a>b)  ||  (a<c))  &&  ((a=c)
||(a>=b))
((a>=b) || (a<d)) && ((a>=d) &&
(c>d))
!(a<=c) || (c>d)
```

Capitulo 3

programación estructurada

3. Programación Estructurada

La programación estructurada es una técnica para crear programas, siguiendo ciertas reglas que permiten que un programa se pueda modificar, leer y mantener facilmente, entre sus características principales es que utiliza 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 sólo programa, sino las veces que se requieran y en cualquier programa.

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

Funciones internas, son las ya implementadas e incorporadas en el lenguaje de programación. Para poder hacer uso de estas funciones, es necesario incluir el archivo de cabecera estándar correspondiente (a veces llamado también biblioteca) al que corresponda cada función la forma de hacer esto es escribiendo la directiva `#include` generalmente al principio de un programa. Algunos de estos archivos de cabecera que utilizaremos en este libro son los siguientes:

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 esta fuera del alcance de este libro, pero se estudiarán las más usadas y se irán explicando en ejemplos conforme avancemos en los temas, ademas existen otras bibliotecas con funciones que no se trataran en el texto. Este libro contiene un anexo con bibliotecas de C y sus respectivas funciones.

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

En un programa se pueden utilizar 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 es 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 sencillo primer programa:

Ejemplo 3.1

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

La primera línea es un comentario, ya que se encuentra entre los caracteres */** y ***/*. Este es opcional, es decir, puede ir o no; recuérdese que los comentarios son una explicación del programa y son ignorados por el compilador, en este caso se trata 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, obsérvese que los símbolos *<* y *>* no son parte del nombre del archivo; sólo se utilizan 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*, la cual 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.

La cuarta y sexta línea contiene los símbolos *{* y *}* que encierran el cuerpo de la función *main( )* y agrupa las instrucciones a ejecutar, son necesarias 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

```
/*Este es el formato 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 sentencia *return 0*. Esta sentencia indica que termina la ejecución del programa y devuelve el control al sistema operativo de la computadora, el número *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 por defecto toma el valor de entero, es decir, se espera que regrese un entero, pero como en este caso no hay nada que regresar, se usa *return 0*.

Obsérvese también que cada sentencia finaliza con punto y coma *;* ya que indica el final de cada instrucción.

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, utilizando elementos que se denominan directivas o directrices de procesamiento, son instrucciones dirigidas al compilador y éste tiene que realizarlas antes de hacer la traducción del programa principal. Las dos directivas más usuales son *#define* e *#include*.

Ya se han mencionado las dos directivas anteriormente, la primera 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 también archivos de inclusión.

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

Estos archivos están en formato ASCII y residen en disco. En realidad, la directiva del preprocesador mezcla un archivo de disco 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:

Dos ejemplos son:

```
#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 por defecto include. El segundo ejemplo significa que el archivo está en el directorio actual, donde se encuentra el programa fuente. Los dos métodos no son excluyentes y pueden existir en el mismo programa archivos de cabecera estándar utilizando ángulos 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. Se inicia con las primeras.

Las estructuras de control, guían el orden de ejecución de un programa, son tres las estructuras de control básicas: secuenciación, selección e iteración.

3.3.1 Secuenciación.

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:

sentencia 1;
sentencia 2;
.
.
.
sentencia n;

donde n es un número finito; las sentencias se van ejecutando desde la primera hasta la enésima instrucción una a una.

Ejercicios resueltos

Ejercicio 3.1

Se desea calcular el promedio de las edades de 3 personas.

Descripción

- Solicitar las tres edades.
- Aplicar la operación del promedio.
- Mostrar el resultado.

Tabla de 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>

void 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);
 }
```

Explicacion

La primera línea encerrada entre /* */ es un comentario, así que 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*; tambien se incluye la biblioteca *conio.h* aunque en este ejemplo no se utiliza ninguna función de esta, no afecta la ejecución del mismo.

Es muy fácil de 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 lee en *e3*; después se hace el cálculo y se asigna a *prom* (en esta misma línea hay un comentario, recuérdese 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 es una estructura de control tan importante que prevalece en todos los programas por muy grandes o pequeños que éstos sean. Y aquí lo realmente importante consiste en encontrar la secuencia exacta para que los programas funcionen de manera eficiente.

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

Encontrar el área de un trapecio

Descripción

- Solicitar base menor, base mayor y altura del trapecio.
- Aplicar la fórmula del trapecio.
- Mostrar el resultado.

Tabla de 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 área de su trapecio es %.2f",r);
  getch();
}
```

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 la base mayor, base menor y altura, mismas que se almacenan en sus respectivas variables b,B,h. Se asigna el resultado de la fórmula en la variable r y finalmente se visualiza 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 sea tecleado un carácter; esto hace que el programa no termine intempestivamente, sino que cuando ejecuta el último `printf()` se quedará en pausa o en espera hasta que se pulse 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

de su trapecio es  28.7
```

Ejercicio 3.3

Calcular el salario de un trabajador.

Descripción

- Solicitar cantidad de horas trabajadas.
- Solicitar el sueldo que se paga por cada hora trabajada.
- 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.

Tabla de variables

Nombre	Tipo	Uso
h	Entero	Número 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>

void 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();
}
```

Explicación

Se solicita la cantidad de horas trabajadas y lo que se paga por cada hora trabajada, y se almacenan en `h` y `sh` respectivamente; luego 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

Calcular el promedio de cuatro números.

Descripción

- Solicitar los 4 números a promediar.
- Aplicar la operación para calcular el promedio.
- Mostrar el resultado.

Tabla de variables

Nombre	Tipo	Uso
a	Entero o real	Primer número a promediar
b	Entero o real	Segundo número a promediar
c	Entero o real	Tercer número a promediar
d	Entero o real	Cuarto número a promediar
tot	real	Resultado

Codificación

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

void 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();
}
```

Explicación

Este programa funciona básicamente 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 número a promediar

3

Inserte el 2do número a promediar

5
Inserte el 3er número a promediar

8
Inserte el 4to número a promediar

6

El promedio es 5.50

Ejercicio 3.5

Calcular el monto de las ventas del día de una pastelería.

Descripción

- Suponer 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.

Tabla de variables

Nombre	Tipo	Uso
G	constante real	Precio por pastel grande

Tabla de variables

Nombre	Tipo	Uso
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
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

void 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 endidos\n");
  scanf("%i",&m);
  printf("Introduzca el Vnumero 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();
}
```

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 leen 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 número 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

Realizar las cuatro operaciones básicas con dos numeros.

Descripción

- Pedir dos valores con los que se harán las 4 operaciones.
- Sumarlos, restarlos, multiplicarlos y dividirlos, mostrando cada resultado.

Tabla de variables

Nombre	Tipo	Uso
a	Entero o real	Primer operando
b	Entero o real	Segundo operando

Codificación

```
/*MINICALCULADORA*/
#include<stdio.h>
#include<conio.h>
void 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();
}
```

Explicación

Este ejemplo inicia solicitando dos valores que son leídos 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

Ejercicios propuestos

I. codifique los siguientes programas

calcule el área y el volumen de un cilindro.
reciba una cantidad de segundos y los convierta a su equivalente en el formato hrs: min: seg.

1. calcular el numero con 10 operaciones de esta serie 4/1-4/3+4/5-4/7+4/9 . . .

tres personas deciden invertir su dinero, cada una de ellas invierte una cantidad diferente. obten el porcentaje de cada uno con respecto al total. pide las tres cantidades.
dibuje un cuadro de asteriscos de de 4x4.
de un nombre y escribalo en ASCII.

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

```
/* 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();
}
```

III Complete los siguientes programas.

1.

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

void 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();
}
```

a _____

b _____

c _____

d _____

2.

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

void 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();
}
```

a _____

b _____

c _____

3.

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

int main(int argc, char *argv[])
{
  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();
}
```

a _____

b _____

c _____

d _____

4.

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

int main(int argc, char *argv[])
{
 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();
}
```

a _____

b _____

c _____

5.

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

main()
{
 clrscr();
 printf("calcular el resultado con 10 operaciones de esta serie
");
 printf("1/2-1/3+1/4-1/5+1/6 . . .\n\n");
 float res;
 res= ___________________
 printf(" pi=%.10f ",res);
 getch();
 return 0;
}
```