Referencia bibliográfica

Ancona Valdéz, María de los Ángeles y Corona Nakamura, María Adriana. (2008). Los datos y las operaciones básicas. En *Diseño de algoritmos y su codificación en lenguaje C*. (Pp. 15-31). México: Centro Universitario de los Valles, Universidad de Guadalajara.

Diseño de algoritmos y su codificación en lenguaje C

María de los Ángeles Ancona Valdez
María Adriana Corona Nakamura

Diseño de algoritmos y su codificación en lenguaje C

Ejercicios resueltos y complementarios

Universidad de Guadalajara
Centro Universitario de los Valles

Primera edición 2008

ISBN 978-970-27-1374-6

Carretera Guadalajara-Ameca km. 45.5
46600 Apartado Postal núm. 200
Ameca, Jalisco, México

Impreso y hecho en México
Printed and made in Mexico

Contenido

II
Los datos y las operaciones básicas

Identificador

Los identificadores son nombres que se utilizan para nombrar las constantes, variables, funciones y otros objetos definidos por el programador. Un identificador es una secuencia de caracteres alfabéticos, numéricos y el guión bajo en lenguaje C.

Un identificador es una secuencia de caracteres que pueden ser de cualquier longitud; cada lenguaje de programación tiene sus propias características de tamaño. En lenguaje C por ejemplo, pueden ser de cualquier longitud, pero sólo los primeros 32 caracteres son significativos (Turbo C – Microsoft C). El programador tiene libertad para darle cualquier nombre a un identificador, siguiendo las siguientes *normas*:

1. Debe comenzar con una letra (A hasta Z), mayúsculas o minúsculas y no puede contener espacios en blanco. En *lenguaje C* el carácter subrayado "_" es considerado como letra, por lo que se puede utilizar como primer carácter.
2. Lenguaje C distingue mayúsculas de minúsculas.
3. Letras dígitos y caracteres subrayados están permitidos después del primer carácter.
4. No se puede utilizar una palabra reservada como identificador, sin embargo los identificadores estándar se pueden redefinir.

En *lenguaje C* existen identificadores especiales que podrían tener uno o varios puntos, tales como: persona.apellidoPaterno
El punto indica el acceso a un miembro de una estructura.

Sugerencias

1. Los identificadores deben tener un nombre que nos diga o nos dé la idea de qué dato estamos usando.
2. No utilizar nombres demasiado largos, tales como: base_Mayor_Trapecio
   Es mejor utilizar b_Mayor
3. En lenguaje C es usual escribir variables en minúscula, reservando las mayúsculas para las constantes. En los casos de nombres compuestos se suele poner la inicial de la segunda palabra en mayúscula. Ejemplo: totalAlumnos, areaCirculo, numeroPositivo.
4. Para las variables utilizar sustantivos y para los subprogramas (funciones) verbos que indiquen la tarea a realizar.
   Variables: area, lado, base
   Subprogramas: leeMatriz, calcula_Suma, comparaNumeros.

Identificadores válidos:

HOME, year2008, Base_1

Identificadores no válidos:

número, ?precio, año, 2007, 4semestre.

En los tres primeros ejemplos se utilizan caracteres especiales y en los dos últimos el primer carácter no es letra.

Tipos de datos

Los diferentes objetos de información con los que un *programa* trabaja se conocen colectivamente como datos. Todos los datos tienen un tipo asociado con ellos; el tipo de un dato es el conjunto (rango) de valores que puede tomar durante el programa. Por lo tanto los tipos de datos son los valores que una variable puede tomar. En la tabla II.1 apreciamos que cada tipo de dato en *lenguaje C* está delimitado por un rango de valores (en el anexo B se puede ver la tabla completa). Como veremos en los ejemplos resueltos, si utilizamos valores fuera del rango correspondiente, el compilador no sabrá qué hacer con dicho dato e imprimirá en pantalla resultados erróneos. El tipo de dato asociado a una variable limita el conjunto de datos que puede almacenar, así como las operaciones aplicables sobre esa variable. Por lo tanto, una variable que pertenece a un tipo de dato *int* no podrá almacenar datos de tipo *char;* tampoco se podrán calcular operaciones propias de otros tipos de datos.

Las computadoras pueden trabajar con varios tipos de datos; los algoritmos y programas operan sobre éstos.

La asignación de tipos a los datos tiene dos objetivos principales:

1. Detectar errores de operaciones en programas.
3. Determinar cómo ejecutar las operaciones.

El tipo de un dato determina la naturaleza del conjunto de valores que puede tomar una variable. Los datos que utilizan los programas los podemos clasificar en *simples* o *compuestos*. Un *dato simple* es indivisible, no se puede descomponer. Como su nombre lo dice, un *dato compuesto* está integrado por varios datos. En este capítulo nos enfocaremos en los primeros.

Los tipos de datos simples son: numéricos (enteros y reales), lógicos y caracteres.

Tipos de datos predefinidos:

- Numéricos.
- Lógicos.
- Caracteres.
- Cadenas.

De ellos, tan sólo el tipo cadena es compuesto. Los demás son los tipos de datos simples considerados *estándares*. Esto quiere decir que la mayoría de los lenguajes de programación permiten trabajar con ellos. Por ejemplo, en lenguaje C es posible utilizar datos de tipo entero, real y carácter; sin embargo, los datos de tipo lógico no se pueden utilizar, ya que no existen en este lenguaje.

El tamaño y el rango de algunos tipos de datos pueden variar, dependiendo del compilador utilizado. Algunas arquitecturas implementan tipos de datos de tamaño como lo podemos observar en la tabla II.1; sin embargo en DEV-CPP el tipo *int* y *float* tiene 32-bit, el *char* 8-bit, y el *double* usualmente es de 64-bit; *bool* se implementa con 8-bit. En lenguaje C se puede utilizar el operador *sizeof* para determinar el tamaño de algunos tipos de datos en bytes.

Hay cinco datos básicos del lenguaje C:

Carácter, entero, coma flotante, coma flotante con doble precisión y sin valor.

Tabla II.1
Tipos básicos predefinidos

Tipo	*Tamaño (bytes)*	*Rango*
int (entero)	2	-32,768 a 32,767
float (flotante)	4	3.4 E-38 a 3.4 E+38
double (flotante de doble precisión)	8	1.7 E-308 a 1.7 E+308
char (carácter)	1	-128 a 127
void	0	sin valor

Fuente: elaboración propia.

Datos numéricos
Este tipo de datos se divide en enteros y reales.

Tipos enteros. Los enteros son aquellos números que no tienen fracciones o decimales. Estos números pueden ser negativos o positivos y el rango es de -32,768 a 32,767. Se almacenan internamente en 2 ó 4 bytes de memoria y pueden ser: Unsigned int, short int, int, Unsigned long o long. Cuando el rango de los tipos básicos no es suficientemente grande para sus necesidades, se consideran tipos enteros largos. Ambos tipos long requieren 4 bytes de memoria (32 bits) de almacenamiento.

Tipos reales o de coma flotante (float/double). Los tipos de datos de coma (punto) flotante representan números reales que contienen una coma (un punto) decimal, tal como 3.1416, o números muy grandes, y pueden ser positivos y negativos formando el subconjunto de los números reales. Para representar números muy pequeños o muy grandes se emplea la notación de punto flotante, que es una generalización de la notación científica. En esta notación se considera al número real como mantisa y al exponente la potencia de 10 a la que se eleva este número. C soporta tres formatos de coma flotante. El tipo *float* requiere 4 bytes de memoria, *double* 8 bytes, y *long double* 10 bytes.

Datos lógicos
Aquel que sólo puede tomar uno de dos valores: verdadero *(true)* o falso *(false)*. En lenguaje C no existe el tipo lógico pero se puede implementar con un número entero, conociendo que 0 es falso y cualquier número diferente de cero verdadero. Algunos compiladores, como por ejemplo el DEV-C++ utiliza el *bool*.

Caracteres
El almacenamiento de caracteres en el interior de la computadora se hace en "palabras" de 8 bits (1 byte). Este tipo representa valores enteros en el rango -128 a +127. El lenguaje C proporciona el tipo *unsigned char* para representar valores de 0 a 255 y así representar todos los caracteres ASCII. Los caracteres se almacenan internamente como números y por tanto se pueden realizar operaciones aritméticas con datos tipo *char*.

Una característica de la parte estándar del conjunto de caracteres (los 128 primeros) es que contiene las letras mayúsculas, las minúsculas y los dígitos, y que cada uno de estos tres subconjuntos está ordenado con su orden natural, por lo que podemos manejar rangos de caracteres bien definidos. Así, la siguiente expresión booleana decide si el carácter contenido en la variable *c* es una letra minúscula *('a'<=c && c<='z')*.

Existe también el dato tipo *cadena* (compuesto), que es una sucesión de caracteres que se encuentran delimitados por comillas; la longitud de una cadena es el número de caracteres comprendidos entre los delimitadores "[long_cad]".

Ejemplos:

Pseudocódigo	Lenguaje C
caracter letra ← 'b', cadena [25]	`char letra = 'b', cadena [25];`
caracter car ← letra - 32	`char car = letra - 32;`

Tipo void
Son datos vacíos o sin valor. Por ejemplo la función *main* no regresa valor alguno (nada): *void* main() o *void* main(void) porque tampoco tiene parámetros. Debemos tener cuidado, ya que esta característica es propia de algunos compiladores, pero por ejemplo en DEV-C++ el *main* tiene que regresar un entero *(int)*, por el *return* 0, es decir *int* main(*void*) o *int* main(); pero las demás funciones predefinidas sí pueden utilizar el tipo *void;* esto lo analizaremos en detalle en el capítulo último de funciones.

Variables

Una variable es un dato cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa. Es decir, representará un valor almacenado en memoria que se puede modificar en cualquier momento o conservar para ser usado tantas veces como se desee.

Hay diferentes tipos de variables: enteras, reales, caracteres y cadenas. Una variable que es de cierto tipo sólo puede tomar valores que correspondan a ese tipo. Si se intenta asignar un valor de tipo diferente se producirá un error.

El programador de lenguaje C es libre de denominar a sus variables con el nombre que considere más adecuado, siempre que se respeten las normas que mencionamos en la sección respectiva para nombrar un identificador. El lenguaje C acepta letras mayúsculas y minúsculas, siendo distintos los nombres en mayúsculas y minúsculas, es decir los nombres *lado* y *Lado* se refieren a variables diferentes.

Como se mencionaba anteriormente, el uso de nombres largos no es recomendable ya que resultarían más difíciles de teclear y además se utiliza más memoria para almacenar el nombre.

Los nombres de las variables nos deben indicar qué dato almacenan, de manera que resulte más fácil leer el programa. Así, la variable *nomAlumno* indica que almacena el nombre de un alumno.

Declaración de variables

Todas las variables deben ser declaradas antes de ser usadas. Cada variable por lo tanto tiene asociado un *nombre* (identificador), un *tipo* y un *valor.* No se admiten como identificadores palabras reservadas del lenguaje de programación que se esté utilizando. Los nombres de variables que se elijan para el algoritmo o programa deben ser significativos y tener relación con el objeto que representa. En lenguaje C la sintaxis para definir o declarar una variable es:

Pseudocódigo	*Lenguaje C*
tipo_dato variable(s)	`tipo_dato variable(s);`

Ejemplos:

Pseudocódigo	*Lenguaje C*
entero i, j, k	`int i, j, k;`
real si	`float si;`
caracter s, nom[25]	`char s, nom[25];`

Las variables del mismo tipo pueden definirse con una definición múltiple, separándolas mediante " , " : int x, y, z;

Una variable puede declararse en cuatro lugares diferentes del programa:

- Fuera de todas las funciones (global).
- Dentro de una función (local a la función).
- Dentro de un bloque enmarcado por llaves {} (local al bloque).
- Como parámetro formal (local a la función).

Reservación de memoria

Cuando declaramos una variable le estamos diciendo al compilador que debe: *reservar espacio en memoria,* que a cada espacio en memoria le asigne un nombre y un número determinado de bytes, dependiendo del tipo de dato asignado; también se le dice qué tipos de datos puede almacenar. En C una variable es una posición de memoria de la computadora con nombre (identificador) donde se almacena un valor con cierto tipo de dato. Una variable es un lugar donde se puede almacenar temporalmente un dato; las variables nos permiten guardar información.

Inicialzación de variables

Las variables pueden también ser inicializadas en el momento de declararse:

Pseudocódigo	Lenguaje C
tipo_dato variable ← valor	`tipo_dato variable = valor;`

Ejemplos:

Pseudocódigo	Lenguaje C
entero i ← 0	`int i = 0;`
real sal ← 30.5	`float sal = 30.5;`

Constantes

Una constante es un dato que permanece sin cambio durante el desarrollo del algoritmo o durante la ejecución del programa, es decir valores fijos que no pueden ser alterados por el usuario. La mayoría de los lenguajes de programación permiten el manejo de diferentes tipos de constantes; éstas pueden ser enteras, reales, caracteres y cadenas.

En la tabla II.2 vemos que además de tener un valor, una constante también tiene un tipo de dato inherente (entero, real, carácter o cadena); el compilador C las evaluará en el momento de la compilación, en lugar de hacerlo en la ejecución.

Tabla II.2
Tipos de constantes

Tipo de constante	*Descripción y ejemplos*
Enteras	Son una sucesión de dígitos precedidos o no por el signo + o - dentro de un rango determinado. Ejemplos: 234, -456 etc.
Reales	Son una sucesión de dígitos con punto adelante, al final o en medio y seguidos opcionalmente de un exponente: Ejemplos: 82.347, 0.63, 32.4e-05, 7.4e03
Carácter	Una constante carácter *(char)* es un carácter del código ASCII encerrado entre apóstrofos. Ejemplos: 'a', 'b', 'c'
Cadena	Una constante cadena es una secuencia de caracteres encerrados entre dobles comillas. Ejemplos: "123", "26 de noviembre de 1974", "Esto es una cadena"

Fuente: elaboración propia.

En lenguaje C una constante se define por medio de la instrucción *#define* (directiva del procesador) o de la palabra *const*.

Pseudocódigo	Lenguaje C
constante iden_const ← valor	#define iden_const valor Ó const tipo iden_const =valor;

Uso de #define

El compilador C tiene un pre-procesador incorporado. Si las líneas
#define LIMITE 100
#define PI 3.14159

se encuentran en un archivo que se está compilando, el pre-procesador cambia primero todos los identificadores LIMITE por 100 y todos los PI por 3.14159, excepto los que estén en cadenas entre comillas. Una línea *#define* puede estar en cualquier lugar del programa, pero debe empezar en la columna 1 y sólo tendrá efecto en las líneas de archivo que le sigue. El pre-procesador sólo cambiará los identificadores que se escriben con mayúsculas.

Uso de const

El cualificador *const* permite dar nombres simbólicos a constantes. Su valor no puede ser modificado por el programa.
Ejemplos:

Pseudocódigo		Lenguaje C
constante MAX ← 100	#define MAX 100	const int MAX = 100;
constante CAR ← 'a'	#define CAR 'a'	const char CAR = 'a';
constante CAR ← "a"	#define CAR "a"	const char CAR[4] = "a";
constante PI ← 3.1416	#define PI 3.1416	const float PI = 3.1416;
constante NOM ← "Marco"	#define NOM "Marco"	const char NOM[10] = "Marco";

Nota: "a" es diferente de 'a',

La representación de 'a' como carácter y "a" como cadena en la memoria seria:

carácter	cadena
a	a\0

Las constantes se presentan en expresiones como:

Pseudocódigo	Lenguaje C
area ← PI * radio * radio	area = PI * radio * radio;

PI es una constante que ya tiene un valor igual a 3.1416.

Operadores

Un *operador* es un símbolo que permite relacionar dos datos en una expresión y evaluar el resultado de la operación.

Los programas de las computadoras se apoyan esencialmente en la realización de numerosas operaciones aritméticas y matemáticas de diferente complejidad. Los operadores fundamentales son:

- Aritméticos.
- Relacionales.
- Lógicos.
- Asignación.

Operadores aritméticos

Los operadores aritméticos de la tabla II.3 (+, - , * , /, ++, --) pueden ser utilizados con tipos enteros o reales y sirven para realizar operaciones aritméticas básicas. Por ejemplo si a = 15 y b = 3, vemos los resultados de los diferentes operadores aritméticos.

Tabla II.3
Operadores aritméticos

Operador			*Ejemplo*			
Pseudo-código	*Lenguaje C*	*Significado*	*Pseudo-código*	*Lenguaje C*	*Operación*	*Resultado*
+	+	Suma	a + b	a + b	Suma de a y b	18
-	-	Resta	a – b	a – b	Diferencia de a y b	12
*	*	Multiplicación	a * b	a * b	Producto de a por b	45
/	/	División	a / b	a / b	Cociente de a entre b	5
Mod	%	Residuo	a Mod b	a % b	Residuo de a entre b	0
^	pow	Potencia	a ^ b	pow(a,b)	a elevado a la b	3375

Fuente: elaboración propia.

El operador % en lenguaje C, como se mencionó, calcula el residuo que queda al dividir dos números enteros; el siguiente programa muestra lo que realiza ese operador:

```
#include <stdio.h>
#include <conio.h>
main( )
{   int x=7,y=2 ;
    printf("%d",x%y),
}
```

En la pantalla aparecerá:

1

Ya que la división de 7/2 es 3 y el residuo es 1

Incremento y decremento

Estos operadores son propios de lenguaje C. En la tabla II.4 podemos ver cómo funcionan.

Tabla II.4
Operadores incrementales y decrementales

++	++i	Se incrementa *i* en 1 y a continuación se utiliza el nuevo valor de *i* en la expresión en la cual resida i
++	i++	Utiliza el valor actual de *i* en la expresión en la cual reside *i*, y después se incrementa a en 1
--	--i	Se decrementa *i* en 1 y a continuación se utiliza el nuevo valor de *i* en la expresión en la cual reside i
--	i--	Se utiliza el valor actual de *i* en la expresión en la cual reside *i*, y después se decrementa a *i* en 1

Fuente: elaboración propia.

Operadores relacionales

Describen una relación entre dos valores; por lo tanto, se usan para expresar condiciones y para comparar dos valores. El resultado de una expresión relacional es un valor tipo lógico, sólo puede ser *verdadero* o *falso*. El lenguaje C representa como verdadero el valor 1 y como falso el valor 0. En la tabla II.5 se muestran los operadores relacionales y en la tabla II.6 ejemplos de los mismos.

Tabla II.5
Operadores relacionales

Operador		*Significado*
Pseudocódigo	Lenguaje C	
>	>	Mayor que
<	<	Menor que
=	==	Igual que
>=	>=	Mayor o igual que
<=	<=	Menor o igual que
< >	¡=	Distinto a

Fuente: elaboración propia.

Tabla II.6
Ejemplos de operadores relacionales

Expresiones relacionales		*Resultado*	*Valor binario*
Pseudocódigo	Lenguaje C		
'A' < 'B'	'A' < 'B'	V	1
'Z' < 'H'	'Z' < 'H'	F	0
'A' = 'a'	'A' == 'a'	F	0
8 <> 8.0	8 != 8.0	F	0
-124.2 < 0.003	-124.2 < 0.003	V	1
6.73 > 6.75	6.73 > 6.75	F	0
'A' < 'a'	'A' < 'a'	V	1

Fuente: elaboración propia.

Otro ejemplo:

x+y<=3*z La expresión anterior compara el valor de la suma de *x* y *y* con el triple del valor de *z;* si la suma es menor o igual que el triple de *z* entonces el resultado de la expresión es 1, y en el caso contrario es 0.

Operadores lógicos

Las expresiones lógicas pueden combinarse para formar expresiones más complejas utilizando los operadores lógicos. En la tabla II.7 se muestran dichos operadores y en la tabla II.8 ejemplos. Estos operadores se utilizan con constantes lógicas de forma similar al modo en que los operadores aritméticos se utilizan con las constantes numéricas; estos operadores trabajan con operandos que son expresiones lógicas: [condición] *operador:* [condición]

Tabla II.7
Operadores lógicos

Pseudocódigo	Lenguaje C
y	&&
o	||
no	!

Fuente: elaboración propia.

Tabla II.8
Ejemplos de operadores lógicos

Expresiones Lógicas		*Resultado*	*Valor binario*
Pseudocódigo	Lenguaje C		
5 > 3	5 > 3	V	1
8 < 4	8 < 4	F	0
5 >3 y 8 < 4	5 >3 && 8 < 4	F	0
5 >3 y 8 > 4	5 >3 && 8 > 4	V	1
5 >3 o 8 < 4	5 >3 || 8 < 4	V	1
2 >3 o 6 < 9	2 >3 || 6 < 9	F	0
no (4 > 2)	! (4 > 2)	F	0
no (7 < 5)	! (7 < 5)	V	1

Fuente: elaboración propia.

Como se muestra en tabla anterior, se pueden evaluar una o más condiciones en una expresión, y siempre el resultado es un único valor lógico.

Para la evaluación de expresiones lógicas es importante conocer la tabla de verdad. En la tabla II.9 hacemos referencia a los operadores lógicos de lenguaje C.

Tabla II.9
Tabla de verdad

P	*Q*	*!Q*	*P && Q*	*P || Q*
verdad	verdad	falso	verdad	verdad
verdad	falso	verdad	falso	verdad
falso	verdad	falso	falso	verdad
falso	falso	verdad	falso	falso

Fuente: elaboración propia.

Otros ejemplos en lenguaje C:
La expresión !(4>1) al evaluarla el resultado que da es 0.
La expresión: a=!(4>1) almacena 0 en la variable a, debido a que la expresión 4>1 es verdadera y el operador *!* niega la expresión haciéndola falsa o igual a cero.

Operadores de asignación

El operador de asignación permite evaluar una expresión y asignar el resultado de la asignación en una variable. Su *sintaxis* es:

Pseudocódigo	Lenguaje C
Identificador ← expresión	`Identificador = expresión;`

Con la asignación anterior le estaremos indicando a la computadora que: evalúe la expresión y la almacene en la variable que se identifica por el identificador. Estos operadores permiten transferir el dato de una variable a otra. Así, la expresión x=a en lenguaje C transfiere el valor de "a" a la variable *x*. En la tabla II.10 se muestran ejemplos de operadores de asignación.

Nunca debe escribirse la expresión a la izquierda del operador de asignación:

Pseudocódigo	Lenguaje C
(cal1 + cal2) ← promedio	`(cal1 + cal2) = promedio;`

Tabla II.10
Ejemplos de operadores de asignación

Pseudocódigo	Lenguaje C	*Significado*
c ← c + 7	`c = c + 7`	Incrementa 7 a la variable c
x ← d – 4	`x = d - 4`	Almacena en x la diferencia de d menos 4
j ← e * 5	`j = e * 5`	Almacena en j el producto de e por 5
f ← f / 3	`f = f / 3`	Divide el valor de f entre 3 y lo almacena en la variable f
g ← g Mod 9	`g = g % 9`	Divide el valor de g entre 9 y almacena el residuo en la variable g

Fuente: elaboración propia.

Otros operadores de asignación en C

En la tabla II.11 vemos otros operadores de asignación propios de lenguaje C: +=, *=, =, /= y %=, donde la sintaxis cuando se quiere utilizar uno de esos operadores es:

Identificador operador_asignación expresión;

Tabla II.11
Otros operadores de asignación en C

Operador	*Ejemplo*	*Equivalencia*
+=	c += 7	c = c +7
-=	d -= 4	d = d - 4
*=	e*=5	e = e* 5
/=	f /= 3	f = f / 3
%=	x %= 9	x = x % 9

Fuente: elaboración propia.

Prioridad de los operadores

Al orden en que la computadora realiza las diferentes operaciones le llamamos *orden de prioridad.*

Operadores aritméticos:

1. *Paréntesis ( ).* Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro hacia afuera; el paréntesis más interno se evalúa primero.
2. *Prioridad de operaciones.* Dentro de una misma expresión o subexpresión, los operadores se evalúan en el siguiente orden:

   * , / ,residuo primero
   +, - último

3. *Regla asociativa izquierda.* Los operadores en una misma expresión con igual nivel de prioridad (tal como * y /) se evalúan de izquierda a derecha.

Operadores lógicos

1.- ! (Negación) 2.- && (y) 3.- || (ó)

En la tabla II.12 apreciamos la prioridad de todos los operadores en lenguaje C.

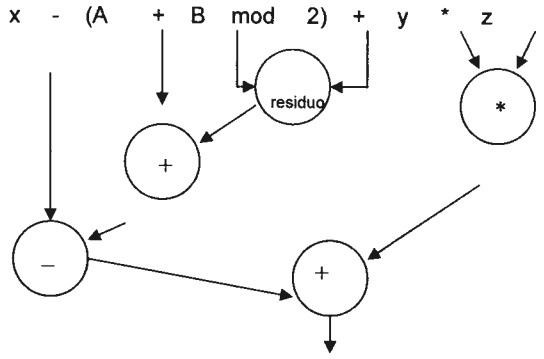
Tabla II.12
Prioridad de los operadores en C

Categoría del operador	*Operadores*	*Asociativa*
Operadores monarios	-,++,--,!, sizeof(tipo)	Derecha a izquierda
Multiplicación, división y residuo	*, /, %	Izquierda a derecha
Suma y sustracción aritmética	+, -	Izquierda a derecha
Operadores de relación	<, <=, >, >=	Izquierda a derecha
Operadores de igualdad	= = , !=	Izquierda a derecha
y	&&	Izquierda a derecha
o	||	Izquierda a derecha
Operadores de asignación	=, +=, - =, *=, /=, %=	Izquierda a derecha

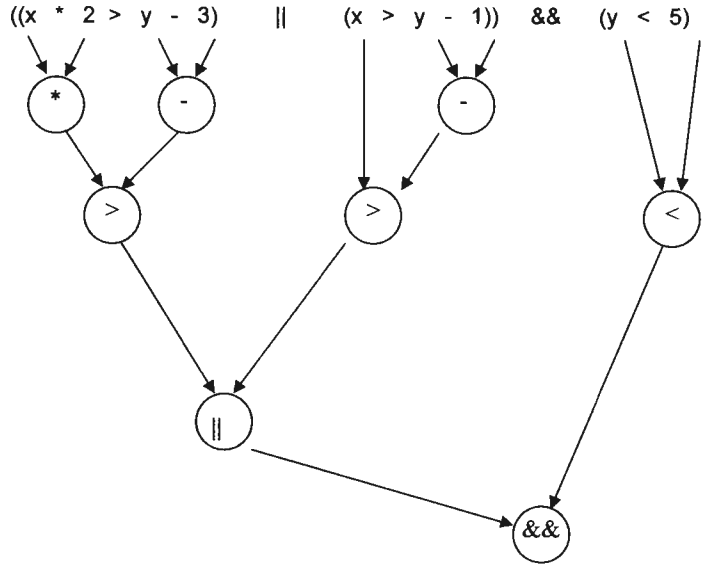
Fuente: elaboración propia.

Los operadores de asignación tienen menor prioridad que todos los otros operadores. Por tanto las operaciones monarios, aritméticos, de relación, de igualdad y lógicos se realizan antes que las de asignación.

El orden de las operaciones en pseudocódigo es:



Otro ejemplo en C:



Expresiones

Una expresión es el resultado de unir *operandos* mediante *operadores*. Los operandos pueden ser variables, constantes u otras expresiones; y los operadores, aritméticos, lógicos o relacionales. El resultado de una expresión es un dato numérico o un valor lógico.

Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

- Aritméticas.
- Relaciónales.
- Lógicas.

Escritura de fórmulas matemáticas

Las fórmulas matemáticas se deben escribir en formato lineal. Esto obliga al uso frecuente de paréntesis que indiquen el orden de evaluación correcto de los operadores.

Ejemplos:

Álgebra	Pseudocódigo	Lenguaje C
$w = \frac{(x+y+z)}{2}$	w ← (x + y + z)/2	`w = (x + y + z)/2;`
$\frac{5-3}{2*4}$	(5-3)/(2*4)	`(5-3)/(2*4)`

Palabras reservadas

Son palabras que tienen un significado especial para el lenguaje y no se pueden utilizar como identificadores. Las más utilizadas en pseudocódigo y lenguaje C las vemos en la tabla II.13. En la tabla II.14 aparecen todas las palabras reservadas de lenguaje C.

Tabla II.13
Palabras reservadas en pseudocódigo y lenguaje C

Pseudocódigo	Lenguaje C	*Función que realiza*
leer	`scanf`	Lee una variable
imprimir	`printf`	Imprime en pantalla
leercad	`gets`	Lee una cadena de caracteres
imprimircad	`puts`	Imprime una cadena
raizcuad()	`sqrt()`	Calcula raíz cuadrada
abs()	`abs()`	Calcula el valor absoluto
inicio	`{`	Inicio del programa o de un bloque
fin	`}`	Fin del programa o de un bloque
si	`if`	Estructura selectiva
sino	`else`	La parte falsa de la selectiva
según_sea	`switch`	Estructura selectiva múltiple
caso contrario	`default`	Ninguna opción de la selectiva múltiple
caso	`case`	Si se cumple un caso
salir o interrrumpir	`break`	Terminar el caso
desde	`for`	Estructura repetitiva
mientras	`while`	Estructura repetitiva
hacer	`do`	Estructura repetitiva
entero	`int`	Tipo de dato entero
real	`float`	Tipo de dato real
caracter	`char`	Tipo de dato carácter
nada	`void`	Valor nulo
regresa	`return`	Regresa valor a otra función

Fuente: elaboración propia.

Las palabras reservadas en C tienen un uso específico y no se pueden utilizar para otros propósitos. Toda palabra reservada se escribe en minúscula.

Tabla II.14
Palabras reservadas de C

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

Fuente: elaboración propia.

Comentarios

Los comentarios son útiles para identificar los elementos principales de un programa o para explicar la lógica subyacente de éstos. Deben ser breves y evitar ambigüedades.

Los comentarios en cualquier lenguaje de programación sirven para que el código fuente sea más entendible, aumentan la claridad de un programa, ayudan para la documentación y bien utilizados nos pueden ahorrar mucho tiempo.

Se deben utilizar comentarios sólo cuando sean necesarios, por ejemplo:

- Al principio del programa: nombre del programa, autor o autores, fecha de elaboración, etcétera.
- En cada sentencia o bloque (bucle, if, switch...) que presenten cierta complejidad (el comentario indicará qué se realiza).
- Al principio de cada función cuyo nombre no haga referencia a la tarea que realiza.
- En la declaración de variables y constantes cuyo identificador no sea suficiente para comprender su utilidad.
- En los cierres de bloques con '}', para indicar a qué sentencias de control de flujo pertenecen, principalmente cuando existe mucho anidamiento de sentencias y/o los bloques contienen muchas líneas de código.

Los comentarios los reconocerá la persona que elaboró el programa o cualquier otro programador, inclusive después de un tiempo. Para el compilador, los comentarios son inexistentes, por lo que no generan líneas de código, permitiendo abundar en ellos tanto como se desee, aunque con medida.

En el lenguaje C se toma como comentario todo carácter interno a los símbolos: /* */ ó iniciarlos con // . Los comentarios pueden ocupar uno o más renglones, por ejemplo:

```
/* Este es un Comentario  */          C estándar
// Este es un Comentario          C++
```

Ejercicios complementarios de los datos y operaciones básicas en pseudocódigo

Ejercicio 1. Realiza un algoritmo para indicar las actividades que realizas el día martes

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.

Ejercicio 2. Escribe válido o inválido en cada identificador para el lenguaje C y por qué

Identificador	Válido o inválido	¿Por qué?
area		
área		
dato 1		
dato_1		
1radio		
radio1		

Ejercicio 3. Añade correcto o incorrecto a cada declaración de variable en pseudocódigo

Declaración de variable	*Válido o inválido*
entero a	
entero _a	
real x	
real x1	
real 1x	
caracter %s	

Ejercicio 4. Añade correcto o incorrecto a cada declaración de constante en pseudocódigo

Declaración de constante				Válido o inválido
constante	MAX	←	20	
constante	MAX	←		
constante	CAR	←	'x'	
constante		←	3.1415	
constante	NOM	←	"Marco"	
constante	CAR	←	c	
constante	G	←	9.8	
constante	NOM	←	Lucy	

Ejercicio 5. De las siguientes expresiones con datos numéricos, cuál es el orden en que se realizan las operaciones según los operadores. Escribe el resultado final según el ejemplo:

a + b * c /d ^ e – f = *, ^, /, +, -

c / d + a * g ^ h – p * y =

(6 * 3 + 8 – 9 / 3 + 2 ^ 4) / 2 =

Ejercicio 6. Convierte la fórmula de la ecuación cuadrática a una expresión algorítmica, es decir que pueda se introducida en un programa de computadora

x1 =

x2 =

Ejercicio 7. Completa la columna de resultado y de valor binario según la expresión

Expresiones lógicas	Resultado	Valor binario
2 > 3		
8 < 15		
7 > 5 y 4 < 9		
12 > 6 y 13 < 9		
7 > 5 o 4 < 9		
12 > 6 o 13 < 9		
no (2 > 4)		
no (5 < 8)		

Ejercicios complementarios de los datos y operaciones básicas en lenguaje C

Preguntas

1. ¿Qué es un identificador?
2. Mencione cuáles son las reglas para nombrar un identificador.
3. Mencione tres tipos de datos en el lenguaje C y sus características.
4. ¿Qué es una variable?
5. ¿Qué es una constante?
6. ¿Qué es una expresión?
7. ¿Qué es un operador?
8. Mencione tres tipos de operadores de lenguaje C.
9. ¿Qué es el orden de prioridad?
10. ¿Cuál es el orden de prioridad de los operadores aritméticos?
11. ¿Cuándo se deben utilizar los paréntesis en una expresión? Mencione un ejemplo.
12. ¿Qué diferencia existe entre los operadores = y == en lenguaje C?
13. ¿Qué es una palabra reservada?
14. ¿Qué es un comentario y cómo se representa en lenguaje C?

Ejercicio 1. ¿De los siguientes identificadores cuál no es válido?

a) _numero
b) número
c) año
d) casa
e) dinero$
f) base_1
g) 2variables

Ejercicio 2. ¿Si en un programa la variable numero tuviera que almacenar la cantidad 40,000, ¿qué tipo de dato recomendarías y por qué?

Ejercicio 3. ¿Escriba las siguientes expresiones en lenguaje C?

a) $\frac{2y+11}{x-3}$

b) $\frac{-b \pm \sqrt{b^2 - 4a}}{2a}$

Ejercicio 4. Si las variables *p, q* y *r* fueran declaradas como tipo *int* y se les asignara los valores p = 25, q = 100 y r = 10, cuál sería el resultado de las siguientes expresiones:

a) !(p < q)

b) q / p

c) 2 * p – 5 + r

d) (-7) % (-3)