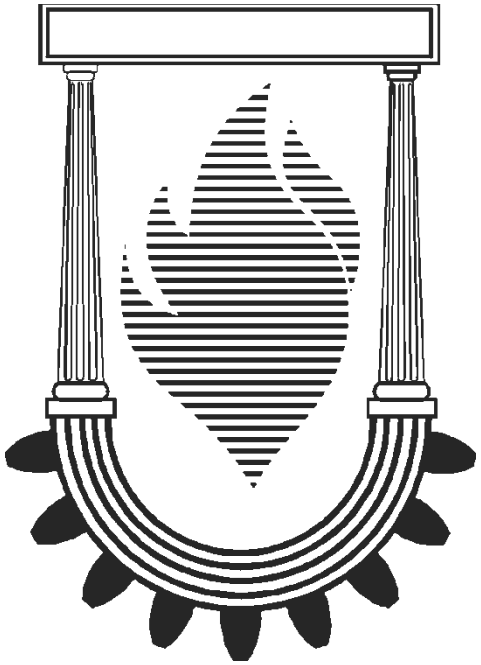
Introducción a C

Con texto de "Caminando junto al Lenguaje C - Martin Goin - Editorial UNRN – 2016"



Definición. Características principales.

- Tipos de datos: int, string, chart, etc.

- Procesamiento: operaciones aritméticas fundamentales.

- Entrada de datos, Salida de información, secuencias de escape

Introducción al lenguaje de la programación

> *Esta máquina puede hacer cualquier cosa que sepamos cómo ordenarle que la ejecute...*
>
> Ada Lovelace (1815-1852)
>
> La frase pertenece a quien es considerada la primera programadora de computadoras de la historia, quien en 1843 escribió y publicó algoritmos para la Máquina Analítica de Charles Babbage.

Conceptos básicos de la programación

Primero vamos a definir programación: es la acción y el efecto de programar. El verbo programar tiene varios usos, se refiere a ordenar e idear las acciones que se realizarán en el marco de un proyecto, como por ejemplo la preparación de máquinas para cumplir con una cierta tarea específica, la preparación de un espectáculo deportivo o artístico, la preparación de datos necesarios para obtener la solución de un cálculo a través de una calculadora, sistema y distribución de materias para una carrera o de temas para un curso o asignatura, etcétera. Pero en la actualidad la noción de programación se encuentra más asociada a la programación en informática. En este sentido, programar es el proceso por el cual un programador escribe, prueba, depura y mantiene un código a partir del uso de un lenguaje de programación.

Lenguaje de programación

Lenguaje artificial que puede ser usado para controlar el comportamiento de una máquina, especialmente una computadora. Éste se compone de un conjunto de reglas sintácticas y semánticas que permite expresar instrucciones que luego serán interpretadas.

Debe distinguirse del lenguaje informático que es una definición más amplia, porque muchas veces es utilizado como sinónimo del lenguaje de programación. Un lenguaje informático no siempre es un lenguaje de programación. Por ejemplo, el html (lenguaje de marca) es un lenguaje informático que describe a la computadora el formato o la estructura de un documento (y no es un lenguaje de programación). Los lenguajes informáticos

engloban a los lenguajes de programación. El programador es el encargado de utilizar un lenguaje de programación para crear un conjunto de instrucciones que, al final, constituirá un programa o subprograma.

En definitiva, los lenguajes utilizados para escribir programas que puedan ser entendidos por las computadoras se denominan lenguajes de programación.

Los lenguajes de programación se clasifican en tres grandes categorías: *lenguaje máquina, lenguaje de bajo nivel y lenguaje de alto nivel.*

- **Lenguaje máquina:** es aquel cuyas instrucciones son directamente entendibles por la computadora. Las instrucciones en lenguaje máquina se expresan en términos de la unidad de memoria más pequeña, es decir el bit (dígito binario 0 y 1). El lenguaje será entendible por el procesador, pero poco claro para el programador. Entonces, para simplificar el lenguaje máquina, aparecieron los lenguajes de bajo nivel.

- **Lenguaje de bajo nivel** (ensamblador): este lenguaje es generalmente dependiente de la máquina, es decir, depende de un conjunto de instrucciones específicas de la computadora. En este lenguaje las instrucciones se escriben en códigos alfabéticos, conocidos como nemotécnicos (abreviaturas de palabras).

add	Suma
sub	Resta
lda	Cargar acumulador
sto	Almacenar

  Por ejemplo: ADD X,Y,Z (esta instrucción significa que deben sumarse los números almacenados en las direcciones X e Y y el resultado quedará en la dirección Z).

  Lo anterior traducido a lenguaje máquina será: 1110 1001 1010 1011

  El ASSEMBLER es el primer lenguaje de este nivel.

- **Lenguaje de alto nivel** (evolucionado): es un lenguaje cuyas instrucciones se escriben con palabras similares a los lenguajes humanos (por lo general en inglés), para facilitar su comprensión.

Este lenguaje es transportable (con pocas o ninguna modificación), es decir que puede ser utilizado en diferentes tipos de computadoras. Otra propiedad es que es independiente de la máquina, esto es, la sentencia del programa no depende del diseño del *hardware*. Incluye rutinas de uso frecuente como las entradas y las salidas, las funciones matemáticas, el manejo de tablas, etcétera (que figuran en librerías del lenguaje), de manera que puedan utilizarse siempre que se las requiera sin tener la necesidad de programarlas cada vez.

El lenguaje de alto nivel no es entendible directamente por la máquina (se aleja del procesador), entonces necesita ser traducido.

Si volvemos al ejemplo anterior, la suma de los números X e Y que queda almacenado en Z será simplemente como una operación aritmética Z=X+Y (asignación).

Los programas escritos en un lenguaje de alto nivel se llaman *programa fuente*.

Existen muchos lenguajes de alto nivel, los principales son: C/C++, Visual Basic, Pascal, PHP, Python, Matlab, PL/SQL, Java y Fortran.

Los lenguajes de programación pueden, en líneas generales, dividirse en dos categorías:

- Lenguajes interpretados.
- Lenguajes compilados.

Lenguaje interpretado

Un lenguaje de programación es, por definición, diferente al lenguaje máquina. Por lo tanto, debe traducirse para que el procesador pueda comprenderlo. Un programa escrito en un lenguaje interpretado requiere de un programa auxiliar (el intérprete), que traduce los comandos de los programas según sea necesario.

Ejemplos de lenguajes interpretados: ASP, Basic, JavaScrip, Logo, Lisp, Perl, PHP, VBScrip, Python, etcétera.

Lenguaje compilado

Un programa escrito en un lenguaje *compilado* se traduce a través de un programa anexo llamado compilador. El compilador traduce el programa fuente a uno llamado programa objeto. Este programa objeto se utiliza en la fase de ejecución del programa, obteniendo un programa ejecutable (que no requiere de ninguna traducción).

Un programa escrito en un lenguaje compilado posee la ventaja de no necesitar un programa anexo para ser ejecutado una vez que ha sido compilado, entonces se vuelve más rápido.

Sin embargo, no es tan flexible como un programa escrito en lenguaje interpretado, ya que cada modificación del archivo fuente (el archivo comprensible para los seres humanos: el archivo a compilar) requiere de la compilación del programa para aplicar los cambios.

Ejemplos de lenguajes compilados: ADA, C, C++, Cobol, Fortran, Pascal, Algol, etcétera.

Se han propuesto diversas técnicas de programación cuyo objetivo es mejorar tanto el proceso de creación de software como su mantenimiento. Entre ellas, se pueden mencionar las siguientes:

- Programación lineal.
- Programación estructurada.
- Programación modular.
- Programación orientada a objetos (POO).

La mejor forma de explicar dichas técnicas es a través de su evolución histórica en los lenguajes de alto nivel.

Tradicionalmente, la programación fue hecha en una manera secuencial o lineal, es decir, una serie de pasos consecutivos con estructuras consecutivas y bifurcaciones.

Los lenguajes basados en esta forma de programación ofrecían ventajas al principio, pero el problema ocurre cuando los sistemas se vuelven complejos y extensos.

Frente a esta dificultad aparecieron los lenguajes basados en la programación estructurada.

Esta programación estructurada utiliza un número limitado de estructuras de control y reduce así considerablemente los errores.

Esta técnica incorpora:

- Diseño descendente (top-dow): el problema se descompone en etapas o estructuras jerárquicas.
- Recursos abstractos (simplicidad): consiste en descomponer las acciones complejas en otras más simples capaces de ser resueltas con mayor facilidad.

Estructuras básicas: existen tres tipos de estructuras básicas:

- Estructuras secuenciales: cada acción sigue a otra acción secuencialmente. La salida de una acción es la entrada de otra.
- Estructuras selectivas: en estas estructuras se evalúan las condiciones y en función de sus resultados se realizan unas acciones u otras. Se utilizan expresiones lógicas.
- Estructuras repetitivas: son secuencias de instrucciones que se repiten un número determinado de veces.

Estos programas no ofrecen flexibilidad y mantener una gran cantidad de líneas de código en un solo bloque se vuelve una tarea complicada, entonces surgió la idea de programación modular. La misma consiste en separar las partes complejas del programa en módulos o segmentos, esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente. Esta técnica se llama refinamiento sucesivo, «divide y vencerás».

De esta manera tenemos un diseño modular, compuesto por módulos independientes que pueden comunicarse entre sí. Poco a poco, este estilo de programación reemplazó al estilo impuesto por la programación lineal.

Entonces, vemos que la evolución que se fue dando en la programación se orientaba siempre a descomponer aún más el programa. Este tipo de descomposición conduce directamente a la programación orientada a objetos.

Pues la creciente tendencia de crear programas cada vez más grandes y complejos llevó a los desarrolladores a generar una nueva forma de programar que les permitiera producir sistemas de niveles empresariales y científicos muy complejos. Para estas necesidades ya no bastaba con la programación estructurada, ni mucho menos con la programación lineal. Es así como aparece la Programación Orientada a Objetos (POO). Básicamente la POO simplifica la programación con la nueva filosofía y sus nuevos conceptos.

La POO se basa en dividir el programa en pequeñas unidades lógicas de código y en aumentar considerablemente la velocidad de desarrollo de los programas.

A estas pequeñas unidades lógicas de código se las llama objetos. Los objetos son unidades independientes que se comunican entre sí mediante mensajes.

Lo interesante de la POO es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

El principal problema que presentan los lenguajes de alto nivel es la gran cantidad que existe en la actualidad.

No existe el mejor lenguaje (algunos son más apropiados para ciertas cosas).

Fases para la creación de un programa

Definición del problema: esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.

Análisis del problema: esta fase requiere de una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o la solución deseada, entonces es necesario definir:

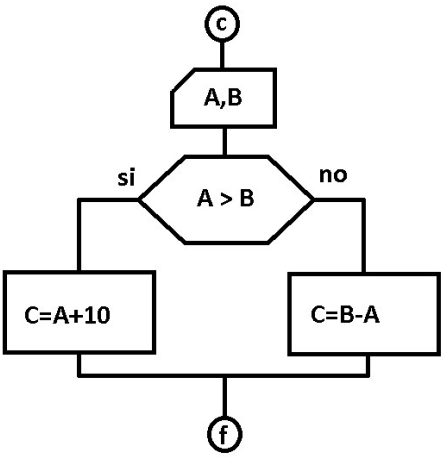
- Los datos de entrada (tipo y cantidad).
- Cuál es la salida deseada (tipo y cantidad).
- Los métodos y las fórmulas que se necesitan para procesar los datos.

En esta etapa se determina *qué* hace el programa.

Una recomendación muy práctica es la de ponernos en el lugar de la computadora y analizar qué es lo que necesitamos que nos ordenen y en qué secuencia para producir los resultados esperados.

Diseño del algoritmo: en esta etapa se determina *cómo* se hace el programa. Este procedimiento es independiente del lenguaje de programación. Las herramientas son el diagrama de flujo y el pseudocódigo.

El diagrama de flujo es una representación gráfica de un algoritmo.



El pseudocódigo son las instrucciones que se representan por medio de frases o proposiciones en español que facilitan tanto la escritura como la lectura de programas.

Si queremos crear el pseudocódigo del diagrama de flujo anterior, entonces:

```
Ingresar A y B
si A > B entonces
            asignar a C = A + 10
sino
            asignar a C = B - A
fin Si
```

La programación se apoya en los métodos llamados *algoritmos*.

La definición del algoritmo es: una secuencia no ambigua, finita y ordenada de pasos para poder resolver un problema. A continuación enumeramos las características de un buen algoritmo:

- Debe tener un punto particular de inicio (programa principal). El módulo de nivel más alto que llama a (subprogramas) módulos de nivel más bajos.
- Debe ser definido, no debe permitir dobles interpretaciones.
- Debe ser general, es decir, soportar la mayoría de las variantes que puedan presentarse en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.

Adicionalmente, los algoritmos pueden requerir de datos de entrada para producir datos de salida.

- Datos de entrada: un algoritmo tiene cero o más entradas, es decir cantidades que le son dadas antes de que el algoritmo comience, o dinámicamente mientras el algoritmo corre.
- Procesamiento de datos: aquí incluye operaciones aritmético-lógicas, selectivas y repetitivas; cuyo objetivo es obtener la solución del problema.
- Salida de resultados: permite comunicar al exterior el resultado. Puede tener una o más salidas.

Codificación: es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora. A estas instrucciones detalladas se las conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

Prueba y depuración: los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración. La prueba consiste en la captura de datos hasta que el programa no presente errores (los más comunes son los sintácticos y lógicos).

Documentación: describe los pasos a dar en el proceso de resolución de un problema. La importancia de la documentación debe ser destacada en el producto final. Programas pobremente documentados son difíciles de leer, más difíciles de depurar y casi imposibles de mantener y modificar.

A menudo un programa escrito por una persona es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones.

La documentación de un programa puede ser interna o externa. La primera es la contenida en líneas de comentarios y la segunda incluye análisis, diagrama de flujo y/o pseudocódigo, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados.

Mantenimiento: se lleva a cabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementación al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa esté correctamente documentado.

Comencemos a programar

Partimos de la definición del problema. La mejor y la típica manera de empezar es con algo cotidiano y simple como la preparación del mate. Teniendo en cuenta que los ingredientes son la yerba y el agua; y los utensillos son el mate, la bombilla, la pava y el termo:

Agregar yerba al mate

Llenar de agua la pava
Encender la hornalla
Colocar la pava sobre la hornalla
Esperar hasta que tenga la temperatura adecuada
Verter el agua de la pava al termo
Poner la bombilla en el mate
Verter el agua del termo en el mate (cebar el mate)

Observar: en un número finito de pasos se resolvió el problema planteado. En este caso, son ocho los pasos a seguir para que el mate quede listo.

Las instrucciones son precisas. En cada paso es claro qué acción realizar. Las recetas culinarias son un claro ejemplo de algoritmo. Existe un problema, elementos para solucionarlos, un procedimiento a seguir y un resultado que es el plato listo.

Pre y post condiciones

En el ejemplo del mate puede observarse que existen condiciones que deben cumplirse antes de realizar el algoritmo (como por ejemplo, disponer de todos los ingredientes y de los utensilios). A estas condiciones necesarias de cumplir antes de comenzar la ejecución del algoritmo, se las llama precondiciones. Pueden asumirse como verdaderas antes de comenzar con la ejecución de las acciones especificadas en el algoritmo.

De la misma forma, existen condiciones post-ejecución, las cuales surgen a partir de la ejecución del algoritmo.

Las instrucciones de un algoritmo pueden repetirse un número finito de veces (si ellas indican repetición) pero un algoritmo debe terminar después de ejecutar un número finito de instrucciones, sin importar cuáles fueron los datos o elementos de entrada.

Una instrucción es una acción a ejecutar (la puede ejecutar cualquier entidad: un hombre, una computadora, un auto, etcétera). Dicha instrucción debe poder realizarse o llevarse a cabo en un número finito de pasos. Es necesario que la instrucciones sean precisas, que solo signifiquen una cosa y que estén libres de ambigüedades.

Habíamos dicho que existe una gran cantidad de lenguajes de programación, algunos con propósito específico y otros de uso generalizado como el lenguaje C que es el que vamos a abordar en este libro.

Lenguaje de programación C

El lenguaje C es uno de los lenguajes de programación más populares en el mundo, se ejecuta en la mayoría de los sistemas operativos y puede ser usado en casi todas las plataformas informáticas.

Fue creado en 1972 por el estadounidense Dennis M. Ritchie, en los laboratorios Bell. En un principio se creó para desarrollar softwares de sistemas (conjuntamente con el UNIX) y, más tarde, se amplió su potencial para desarrollar softwares de aplicaciones.

Puede ser que este lenguaje sea más difícil de aprender que otros, pero su ventaja es la versatibilidad que ofrece. Los programas desarrollados en lenguaje C pueden ser más pequeños que los mismos hechos en otros lenguajes y, por ende, más rápidos.

Una desventaja es que se trata de un lenguaje más tolerante a los errores de programación que otros, lo que significa que un descuido puede causar grandes consecuencias.

Uno de los objetivos de diseño del lenguaje C fue que solo sean necesarias unas pocas instrucciones en lenguaje máquina para traducir cada uno de sus elementos, sin que haga falta un soporte intenso en tiempo de ejecución.

Es posible escribir C a bajo nivel de abstracción. De hecho, C se usó como intermediario entre diferentes lenguajes.

Hay una versión muy interesante en el origen del lenguaje que dice que el C fue el resultado del deseo de los programadores Thompson y Ritchie de jugar al *Space Travel* (1969). Habían estado jugando en el *mainframe* de su compañía, pero debido a su poca capacidad de proceso y al tener que soportar 100 usuarios, no tenían suficiente control sobre la nave para evitar colisiones con los asteroides. Por ese motivo decidieron portar el juego a otra máquina ubicada en otro sector de la compañía. El problema era que aquella máquina no tenía sistema operativo, así que decidieron escribir uno. Como el sistema operativo de la máquina que poseía el juego estaba escrito en lenguaje ensamblador ASSEMBLER, decidieron usar un lenguaje de alto nivel y portátil. Consideraron usar B (lenguaje anterior al C), pero éste carecía de las funcionalidades necesarias para aprovechar algunas características avanzadas, entonces comenzaron a crear uno nuevo, el **lenguaje C**.

Ritchie (parado) junto a Thompson, año 1972. Ambos trabajando con la PDP-11
Fuente: Wikipedia.

Las principales características que posee el lenguaje C son:

- Es un lenguaje de programación de nivel medio ya que combina los elementos del lenguaje de alto nivel con la funcionalidad del ensamblador.
- Sirve para crear aplicaciones y software de sistemas.
- Es portable, es decir, posibilita la adaptación de programas escritos para un tipo de computadora en otra.
- Es de fácil aprendizaje.
- Posee un completo conjunto de instrucciones de control.
- Al ser un lenguaje estructurado se divide el programa en módulos, lo que permite que puedan compilarse de modo independiente.
- El código generado por el lenguaje es muy eficiente ya que los datos son tratados directamente por el hardware de números, caracteres y direcciones de las computadoras.

- Trabaja con librerías de funciones en las que básicamente solo se necesita cambiar los valores dentro de una aplicación dada.
- Es uno de los lenguajes más populares. Muy utilizado, especialmente en el campo de la ingeniería y el campo científico.
- Dispone de excelentes compiladores de C gratuitos, para casi cualquier plataforma sobre la que se quiera trabajar y con entornos de programación claros y funcionales.

> *UNIX es simple. Sólo necesita un genio para entender su simplicidad.*
> DENNIS RITCHIE (1941-2011)

Dennis Ritchie fue un científico en computación que colaboró con el desarrollo del sistema operativo UNIX y creador del lenguaje de programación C junto a Ken Thompson.

Código

En el capítulo anterior definimos el algoritmo como «una secuencia no ambigua, finita y ordenada de pasos para poder resolver un problema».

El diseño del algoritmo es independiente del lenguaje de programación, este puede ser usado para cualquier lenguaje de programación.

Anteriormente habíamos mencionado las distintas maneras de expresar un algoritmo (gráfico con diagrama de flujo y textual con pseudocódigo), en nuestro caso trabajaremos con código.

Algoritmos (primeros pasos)

Tabla de herramientas básicas

La siguiente tabla muestra algunas de las herramientas básicas del pseudocódigo y su significado.

Código	Significado	Ejemplo
main ( )	Es la estructura que determina el inicio del algoritmo.	

scanf();	Sirve para ingresar datos por teclado	scanf("%d",&A); → ingresa un dato para la variable A de tipo entera
printf();	Muestra por pantalla un dato, un cartel o ambas	printf("%d",A); → imprime por pantalla el dato que tiene la variable A printf("Así es la vida"); → imprime por pantalla el cartel "Así es la vida" printf("El promedio es %lf",P); → imprime por pantalla "El promedio es 176.34", en este caso, la variable P contiene dicho valor numérico. printf("El área es %d y el perímetro es %lf",A,P); → imprime por pantalla "El área es 84.7 y el perímetro es 56".

{ }	Denota el comienzo → { y el fin →} de un grupo de sentencias y/o el propio algoritmo como se observa en el ejemplo.
	
	Nota: todo algoritmo debe tener el comienzo y las llaves. En un algoritmo puede haber más de { }, siempre y cuando resulten equilibradas, es decir que exista la misma cantidad de llaves que abren { con las que cierran }
		main()
		{
		double A,B,P;
		scanf ("%lf",&A);
		scanf ("%lf",&B);
		P = (A+B)/2;
		printf("El promedio es %lf",P);
		}

=	Asignación	P=A+B; C=8; D=7+A;
;	Fin de sentencia	Se observan en los ejemplos anteriores.

Nota: a medida que avancemos incorporaremos más código. Todas las palabras clave como main, printf y scanf van siempre en minúscula.

Atención. las operaciones aritméticas deben expresarse siempre en una línea, por ejemplo si queremos realizar la siguiente operación: $C = \frac{-B+9}{2A}$ entonces debemos escribirla de la siguiente manera: C = (-B+9)/(2*A)
El asterisco * representa en programación la multiplicación y la barra inclinada a la derecha / , la división.

La mejor manera de empezar a entender los algoritmos es a través de ejemplos prácticos.

A continuación veremos algunos ejemplos de algoritmos expresados en código:

Ejemplo 1: ingresar tres números y mostrar su sumatoria.

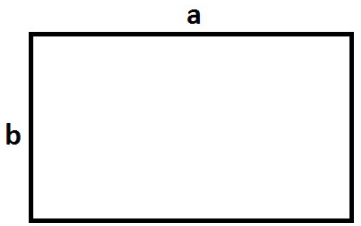
```
main()
{
double A, B,C;
scanf("%d %d %d",&A,&B,&C);
double S = A+B+C;
printf("La suma es %d",S); → Claramente se ve que se imprime el
cartel "La suma es "y el valor que contiene S
}
```

Nota: en la primera línea main() no lleva «;».

Ejemplo 2: ingresar la temperatura actual en Celsius y mostrarla en Fahrenheit.

```
main()
{
double C;
scanf("%lf",&C);
double F = 9/5*C+32;
printf("La temperatura en Fahrenheit es %lf",F); }
```

Ejemplo 3: hallar el perímetro y el área de un rectángulo ingresando sus lados.




```
main ( )
{
int a,b;
scanf("%d %d",&a,&b);
int peri = 2*a + 2*b;
int area = a*b;
printf("El perímetro del rectángulo es %d",peri);
printf("El área del rectángulo es %d",area);
}
```

Variables y Constantes

Un objeto es una **variable** cuando su valor puede modificarse y además posee un nombre que lo identifica y un tipo que describe su uso.
Un objeto es una **constante**, cuando su valor no puede modificarse y además posee un nombre que lo identifica y un tipo que describe su uso.

Cuando definimos una variable, creamos un identificador (nombre de la variable) que hace referencia a un lugar de la memoria donde se almacena un dato. La diferencia respecto de la definición de una constante, es que en el momento de su creación el valor del objeto es desconocido, mientras que para una constante no solo es conocido, sino que permanece inalterado durante la ejecución del procedimiento resolvente. Recuerde que la definición de los objetos siempre tiene lugar en el ambiente.

Ejemplo 4: hallar el área y el perímetro de una circunferencia ingresando el radio r.

En este caso debemos utilizar π para los dos cálculos (como no podemos denominarlo con el símbolo para usarlo en el algoritmo, lo llamaremos PI). Al mantenerse inalterable lo definimos como una constante.

```
define  PI  3.1416
main ( )
{
int r;
scanf("%d",&r);
int P = 2*PI*r;
int A = PI*r*r;   → r2 lo expresamos simplemente como r*r
printf("El perímetro de la circunferencia es %d",P);
printf("El área de la circunferencia es %d",A);
}
```

> **Nota:** Las constantes se definen antes de *main* y no llevan «=» ni «;».

Identificadores para las variables y las constantes

Los nombres o etiquetas de las variables y las constantes siempre deben empezar con una letra (mayúscula o minúscula) y no pueden contener espacios en blanco, si usamos más de un caracter para su identificación empezamos con la letra y luego podemos seguir con números o letras. Está permitido usar «_» entre medio y no está permitido usar palabras reservadas (reservadas por el propio lenguaje de programación).

Ejemplos válidos: A, a, B1, A20, AA1, Aa1, B2B, Promedio, SUMATORIA, A_1, b1_2
Ejemplos no válidos: 1B , 2c , _S , ¿A, La variable

Se recomienda que el nombre represente el dato en sí y que no sea extenso, algunos ejemplos:

Para una sumatoria → S
Para dos sumatorias → S1 , S2
Para la sumatoria de edades → SE
Para contar → C
Para un promedio → P o Prom
Para el promedio de edades de mujeres → PEM

En el caso del lenguaje C se diferencian los identificadores escritos en mayúsculas con las minúsculas, por ejemplo la variable "a" podría representar la longitud de un lado de un poliedro y la variable "A" el área de dicho poliedro.

Operador aritmético módulo %

Un operador que se utiliza con frecuencia es el *módulo* que se obtiene del resto de una división. En el lenguaje se denota con el símbolo %. Por ejemplo, si queremos saber el resto de la división entre dos números lo hacemos así: (A % B).

Ejemplo 14: verificar que el número ingresado sea múltiplo de 3.

```
main()
{
int N;
scanf("%d",&N);
if (N % 3 == 0)
      printf("%d es múltiplo de 3",N);
else
      printf("%d no es múltiplo de 3",N);
}
```

Ejemplo 15: mostrar si el número ingresado es par o impar.

```
main()
{
int N;
scanf("%d",&N);
if (N % 2 == 1)
      printf("%d es impar",N);
else
      printf("%d es par",N);
}
```