Lecturas de Cátedra

Caminando junto al lenguaje C

Martín Goin

EDITORIAL UNRN

Caminando junto al lenguaje C

Lecturas de Cátedra

Caminando junto al lenguaje C

Martín Goin

EDITORIAL UNRN

Utilice su escáner de
código QR para acceder
a la versión digital

Agradecimientos y dedicatorias

A mis seres queridos: mi hijo Fran, mi amor Ira, mi madre Marilú y mi padre Luis.
A mi entorno laboral: aquellos colegas que luchan por mejorar la educación.
Y a mis queridos/as alumnos/as que tuve, tengo y tendré a lo largo de mi docencia.

Índice

Prólogo

Este material de lectura tiene por objetivo presentar a los estudiantes la posibilidad de aprender y manejar de modo progresivo un lenguaje de programación, atendiendo principalmente a los aspectos prácticos del uso de algoritmos, sin descuidar los conceptos teóricos de cada tema. En particular, se apunta a dar a conocer el *lenguaje C*, muy utilizado en las cátedras universitarias.

El libro, que expone numerosos ejemplos e intenta ayudar y acompañar al lector a resolver los problemas planteados, es ideal para aquellos estudiantes que incursionan por primera vez en el mundo de la programación. El ejemplar funciona como un tutorial que explica paso a paso las bases de la programación de modo muy sencillo y está orientado a los primeros cursos de grado para carreras como ingeniería, profesorados, licenciaturas, tecnicaturas y otras disciplinas que incluyan materias de programación.

Elegir un título para un libro de estas características no fue una tarea sencilla, muchos de los que imaginaba ya habían sido utilizados por otros autores. Pero cierto día, al mirar los cuadros que mi padre pintó en sus tiempos ociosos, encontré la respuesta que estaba buscando. Los paisajes patagónicos aparecen en la mayoría de sus obras, y en todas está la imagen de un camino. Desde mi lugar y los intereses que me convocan, entiendo que esos caminos son semejantes a los que iremos transitando los docentes junto con los estudiantes, en esta tarea de guiarlos en un terreno que para muchos es desconocido, *la programación.*

El material se divide en seis capítulos. Cada capítulo, salvo el primero, contiene ejercicios para que resuelvan los estudiantes. En total son 213 ejercicios y además contiene un total de 93 ejemplos prácticos. El último capítulo es un regalo que hace el autor al lector, donde presenta una guía rápida del programa Calc (planilla de cálculo) muy útil para su formación universitaria. Los contenidos son los siguientes:

Capítulo 1: Introducción al lenguaje de la programación

Este capítulo es el único de carácter teórico y contiene los conceptos básicos de programación, los distintos tipos de lenguajes existentes, el ciclo de vida de un programa y una breve reseña histórica del lenguaje C y sus características principales. Concepto de algoritmo.

Capítulo 2: Pseudocódigo

Herramientas para desarrollar algoritmos en pseudocódigo. Variables y constantes. Estructuras de control de decisión (simple, doble, anidada y múltiple). Operadores de relación. Operadores lógicos (conjunción, disyunción y complemento). Variables contadoras y sumadoras. Estructura de control de repetición (exacta, anidada y condicional). Máximos y mínimos.

Capítulo 3: Codificación

Librerías. Declaración de variables y constantes. Tipos de variables primitivas (numéricas y carácter). Pasar de pseudocódigo a código los algoritmos. Distintas partes de un programa. Inclusión de comentarios en C. Codificación de las estructuras de control de decisión y repetición. La función getchar(). Librería math.h

Capítulo 4: Arreglos, vectores o matrices

Concepto de vectores. Vectores paralelos. Máximos y mínimos con vectores. Búsqueda y edición. Ordenamiento de elementos (método burbuja). Cadena de caracteres. Generación de números aleatorios. Ingreso y funciones para cadena de caracteres. Matrices.

Capítulo 5: Funciones y procedimientos

Descomposición de programas en subprogramas. Modularización. Funciones. Pasaje de argumentos (tipos de parámetros). Ámbito de variables. Variables locales y globales. Procedimientos. Menús. Recursividad. Estructura de datos. Registros de datos. Arreglos de registros.

Capítulo 6: Planilla de cálculo-Calc

Descripción del entorno gráfico y área de trabajo. Algunas herramientas del menú y barra. Principales funciones del Calc. Funciones con operadores lógicos. Ordenamientos de planillas. Referencias relativas y absolutas. Gráficos.

Capítulo 1: 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:

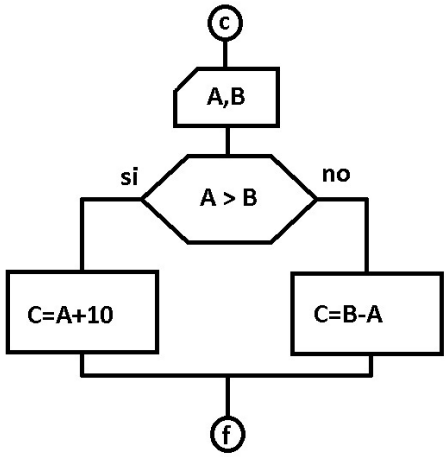
- Los datos de entrada (tipo y cantidad).
- Cuál es la salida deseada (tipo y cantidad).
- Los métodos y las fórmulas que se necesitan para procesar los datos.

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.

Capítulo 2: Pseudocódigo

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 el último.

El pseudocódigo está diseñado para facilitar su comprensión, en lugar de la lectura mediante la computadora.

También se utiliza en la planificación del desarrollo de programas informáticos, para esquematizar la estructura del programa antes de realizar la efectiva codificación.

No existe una sintaxis estándar para el pseudocódigo pero, en definitiva, se parecen.

En nuestro caso, el pseudocódigo que utilizaremos en el libro está muy ligado al lenguaje C, es decir, conserva en gran medida la sintaxis del código. En principio, la escritura será rígida, pero la gran ventaja estará en el pasaje del pseudocódigo al código ya que será casi directo.

Algoritmos (primeros pasos)

Tabla de herramientas básicas

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

Pseudocódigo	Significado	Ejemplo
`comienzo()`	Determina el inicio del algoritmo. Debe estar siempre en todo algoritmo	

ingresar	Sirve para ingresar datos por teclado	`ingresar(A);` → ingresa un dato para la variable A `ingresar(A,B);` → ingresa un dato para cada variable A y B
imprimir	Muestra por pantalla un dato, un cartel o ambas	`imprimir(A);` → imprime por pantalla el dato que tiene la variable A `imprimir("Así es la vida");` → imprime por pantalla el cartel "Así es la vida" `imprimir("El promedio es "P);` → imprime por pantalla "El promedio es 176.34", en este caso, la variable P contiene dicho valor numérico. `imprimir("El área es ", A," y el perímetro es ",P);` → imprime por pantalla "El área es 84.7 y el perímetro es 56", en este caso se pueden alternar cartel y variables.

{ }	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 }
		`comienzo()`
		`{`
		`ingresar A,B);`
		`P = (A+B)/2;`
		`imprimir("El promedio es ",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 pseudocódigo. Todas las palabras clave como comienzo, ingresar e imprimir 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 pseudocódigo:

Ejemplo 1: ingresar tres números y mostrar su sumatoria.

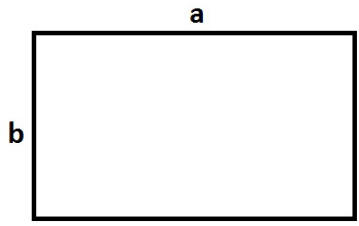
```
comienzo()
{
ingresar(A,B,C);
S = A+B+C;
imprimir("La suma es ",S); → Claramente se ve que se imprime el
cartel "La suma es "y el valor que contiene S
}
```

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

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

```
comienzo()
{
ingresar(C);
F = 9/5*C+32;
imprimir("La temperatura en Fahrenheit es ",F);
}
```

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




```
comienzo()
{
ingresar(a,b);
P = 2*a + 2*b;
A = a*b;
imprimir("El perímetro del rectángulo es ",P);
imprimir("El área del rectángulo es ",A);
}
```

> **Nota:** en el ejemplo tres se ve claramente que las letras A, P, a y b son variables y que en nuestro caso se diferencian las minúsculas de las mayúsculas (esto no sucede en todos los lenguajes de programación).

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.

```
definir  PI  3.1416
comienzo()
{
ingresar(r);
P = 2*PI*r;
A = PI*r*r;   → r2 lo expresamos simplemente como r*r
imprimir("El perímetro de la circunferencia es ",P);
imprimir("El área de la circunferencia es ",A);
}
```

Nota: Las constantes se definen antes de la palabra *comienzo* 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.

Estructuras de control

Las estructuras de control nos permiten controlar el flujo del programa: tomar decisiones, realizar acciones repetitivas, etcétera, según las condiciones que nosotros establezcamos.

El concepto de flujo de control se refiere al orden en que se ejecutan las sentencias o acciones de un programa.

En los anteriores ejemplos hemos visto un flujo lineal también llamado e*structura secuencial,* pero existen también la estructura alternativa y la de repetición, métodos que permiten desviarse de la estructura secuencial.

Estructura de control de decisión

También llamada de alternativa, permite bifurcar el *flujo* del programa en función de una expresión lógica o condición lógica.

Con frecuencia aparecen en algoritmos situaciones donde debe elegirse un camino según los datos de entrada y la condición impuesta.

Existen, a su vez, tres estructuras de decisión: simple, doble y múltiple.

Estructura de control de decisión simple

Veamos un ejemplo de un algoritmo que utilice una decisión simple:

Ejemplo 5: mostrar el perímetro de una circunferencia, siempre y cuando el radio que se ingrese sea mayor a cero.

```
definir PI 3.1416
comienzo()
{
ingresar(r);
si (r > 0)
         {
         Peri = 2*r*PI;
      imprimir("El perímetro de la circunferencia es  ",Peri);
```

```
        }
}
```

Nota: se observa que las dos sentencias que se encuentran dentro del si (el cálculo del perímetro y su impresión) están delimitadas por las llaves { } porque deben escribirse siempre y cuando tengamos más de una sentencia en una decisión.
En este caso si se comete el error de ingresar un radio negativo o cero, el programa simplemente termina, ya que solo funciona cuando la condición r > 0 (expresión lógica) sea verdadera.
Además lo que se encuentra dentro del si está tabulado a derecha. Esta técnica se llama *indentación*.

Indentación: también se lo conoce como sangrado. La palabra es de origen inglés *indentation* y se trata de la sangría y saltos de línea para mejorar la legibilidad de los algoritmos.
Si escribimos el algoritmo de modo compacto, es decir sin tabulaciones, ni saltos, el programa no sufre efecto alguno, pero dificulta la interpretación del programador.
El ejemplo 5 podría escribirse en una línea, de la siguiente forma:

```
comienzo(){ingresar (r);si (r > 0){Peri = 2*r*PI;imprimir (“El
perímetro de la circunferencia es  ”,Peri);}}
```

Estructura de control de decisión doble

Esta estructura es similar a la anterior con la salvedad de que se indican acciones no solo para la rama *verdadera*, sino también para la *falsa*; es decir, en caso de que la expresión lógica sea cierta, se ejecuta una acción o grupo de acciones y, en caso de que sea falsa, se ejecuta el otro grupo de acciones.

Ejemplo 6: ídem al ejemplo anterior, pero en el caso de ingresar un radio erróneo (cero o negativo) indicarlo con el cartel «Error en el ingreso».

```
comienzo()
{
ingresar(r);
```

```
si (r > 0)
         {
         Peri = 2*r*PI;
         imprimir(“El perímetro de la circunferencia es  ”,Peri);
         }
sino
         imprimir(“Error en el ingreso”);
}
```

Nota: cuando es falsa la expresión lógica (r > 0) se ejecuta lo que está dentro del «sino». En la medida en que se van incorporando estructuras de control en el algoritmo, se indentan las sentencias que se encuentran dentro de éstas.
Además se ve claramente que la instrucción *imprimir (“Error en el ingreso”);* que está dentro del “sino” no tiene llaves, porque solo se trata de una sentencia.

Ejemplo 7: mostrar el número más grande (entre dos) ingresado por pantalla.

```
comienzo()
{
ingresar(A,B);
si (A > B)
        imprimir(“El número ”,A,” es el mayor”);
sino
        imprimir(“El número ”,B,” es el mayor”);
}
```

Nota: claro que este algoritmo funciona cuando los valores A y B ingresados sean distintos.
¿Qué sucede si ingresamos dos valores iguales? ¿Cuál sería el resultado?

Atención: se recomienda en una estructura de control de decisión doble no dejar vacío el *si* por ocupar el *sino*.
En el ejemplo 5 podríamos haber hecho la condición del *si* con (r < = 0) en vez de (r > 0), entonces quedaría de la siguiente manera:

```
comienzo()
{
ingresar(r);
si (r <= 0)
          →sin sentencias
sino
    {
    Peri = 2*r*PI;
    imprimir("El perímetro de la circunferencia es  ",Peri);
    }
}
```

Se observa que el *si* no tiene sentencias. En estos casos es conveniente cambiar la condición del *si* para que quede una estructura de control de decisión simple.

Sentencias si anidadas

Existe la posibilidad de tener un si dentro de otro, se llama si anidados. Es muy empleado cuando tenemos más de una alternativa. Se debe tener cuidado con que el si interior esté completamente dentro del si exterior, ya que si se cruzan se produce un error. En pocas palabras, hay que colocar correctamente las llaves que abren y cierran.

Incorrecto	Correcto

`si (condición)`
`{`
`si (condición)`
`{`
`}`
`}`
	`si (condición)`
	`{`
	`si (condición)`
	`{`
	`}`
	`}`

A continuación veremos un caso de anidamiento.

Ejemplo 8: mostrar el número más grande (entre dos) ingresado por pantalla. Si los números son iguales mostrar el cartel «Son iguales».

```
comienzo()
{
ingresar(A,B);
si (A > B)
        imprimir(“El número ”,A,” es el mayor”);
sino
        si (A == B)
                imprimir(“Son iguales”);
        sino
                 imprimir(“El número ”,B,” es el mayor”);
}
```

Nota: dentro del *sino* se encuentra otra estructura *si,* porque si no es A > B caben dos posibilidades, que sean iguales, o que B > A. Es importante respetar la indentación en la medida en que se van agregando estructuras dentro de otras.

El ejemplo anterior podía haberse realizado de otra manera, por ejemplo:

Ejemplo 9:

```
comienzo()
{
ingresar(A,B);
si (A == B)
       imprimir(“Son iguales”);
sino
       si (A > B)
              imprimir(“El número ”,A,” es el mayor”);
       sino
               imprimir(“El número ”,B,” es el mayor”);
}
```

Nota: es decir, comenzar por preguntar si son iguales, o también así podríamos hacerlo de la siguiente manera:

Ejemplo 10:

```
comienzo()
{
ingresar(A,B);
si (A >= B)
       si (A == B)
              imprimir(“Son iguales”);
       sino
              imprimir(“El número ”,A,” es el mayor”);
sino
       imprimir(“El número ”,B,” es el mayor”);
}
```

> **Nota:** en este caso la anidación se produce en el *si* y no en el *sino,* como en los ejemplos 8 y 9.
> Prueba hacer el algoritmo si comenzamos con la expresión B > A y otro algoritmo con B ≥ A.

Estructura de control múltiple:

El problema que presenta el *si* anidado se presenta cuando tenemos numerosas alternativas. Por ejemplo, si nos piden mostrar el nombre del mes ingresando su número, tendremos 12 alternativas, es decir once si anidados (no se cuenta el último por defecto).

Ejemplo 11: ingresar el número del mes para mostrar su nombre, usando anidamiento.

```
comienzo()
{
ingresar (M);
si (M == 1)
       imprimir(“Enero”);
sino
       si (M == 2)
         imprimir(“Febrero”);
      sino
```

```
        si (M == 3)
          imprimir(“Marzo”);
        sino
          si (M == 4)
            imprimir(“Abril”);
          sino
            si (M == 5)
              imprimir(“Mayo”);
            sino
              si (M == 6)
                imprimir(“Junio”);
              sino
                si (M == 7)
                  imprimir(“Julio”);
                sino
                  si (M == 8)
                    imprimir(“Agosto”);
                  sino
                     si (M == 9)
                      imprimir(“Septiembre”);
                     sino
                       si (M == 10)
                        imprimir(“Octubre”);
                       sino
                         si (M == 11)
                          imprimir(“Noviembre”);
                         sino
                            imprimir(“Diciembre”);
}
```

Nota: para anidar N alternativas, se necesitan siempre N-1 *si* anidados.

La solución más económica será la utilización de la estructura de control «según – caso» que trabaja como un distribuidor de casos. Veamos como se implementa en el ejemplo anterior:

Ejemplo 12: ingresar el número del mes para mostrar su nombre, usando control múltiple.

```
comienzo()
{
ingresar (M);
según (M)
       {
       caso 1:
              imprimir(“Enero”);
              salir;
       caso 2:
              imprimir(“Febrero”);
              salir;
       caso 3:
              imprimir(“Marzo”);
              salir;
       caso 4:
              imprimir(“Abril”);
              salir;
       caso 5:
              imprimir(“Mayo”);
              salir;
       caso 6:
              imprimir(“Junio”);
              salir;
       caso 7:
              imprimir(“Julio”);
              salir;
       caso 8:
              imprimir(“Agosto”);
              salir;
       caso 9:
              imprimir(“Septiembre”);
              salir;
       caso 10:
              imprimir(“Octubre”);
              salir;
       caso 11:
              imprimir(“Noviembre”);
              salir;
      defecto:
```

```
                imprimir(“Diciembre”);
                salir;
        }
}
```

> **Nota:** la estructura *según* necesita solo del par de llaves *{}* para delimitar todos los casos. Cada *caso* debe terminar con un *salir* para saltear los casos consecutivos y la última alternativa se obtiene por *defecto*.

Ejemplo 13: ingresar dos números y el operador aritmético (suma “+”, resta “-“, producto “*” o división “/”) para realizar el cálculo y mostrar el resultado.

```
comienzo()
{
ingresar(A,B);
ingresar(operador);
según (operador)
     {
     caso ‘+’:
            R = A + B;
            salir;
     caso ‘-’:
            R = A - B;
            salir;
     caso ‘*’:
            R = A * B;
            salir;
     caso ‘/’:
            R = A / B;
            salir;
     defecto:
            R = 0;
            salir;
     }
imprimir(“El resultado es “, R);
}
```

Nota: en estos casos el *defecto* se utiliza por si no ingresamos correctamente el operador aritmético, en estos casos se determinó que su resultado sea directamente cero.
Además existe un inconveniente en la división si el denominador B es igual a cero. Para poder solucionar el problema podría incorporarse a este caso una decisión como se muestra a continuación:

```
caso “/”:
     si (B ==0)
            R = 0;
     sino
            R = A / B;
     salir;
```

En estos casos el resultado devolverá cero si B es nulo.
Claramente se ve que pueden combinarse los distintos tipos de alternativas.
¿Cómo podríamos mejorar el ejercicio 13 para que devuelva un mensaje de error en el caso de que exista un operador mal tipeado y/o el denominador en cero cuando queremos dividir?

Operadores de relación

A continuación veremos una tabla con todos los operadores de relación que se utilizan para expresar las condiciones entre dos valores:

Operador	Significado	Equivalente en matemática
>	Mayor que	>
<	Menor que	<
>=	Mayor o igual que	≥
<=	Menor o igual que	≤
==	Igual	=
!=	Distinto	≠

Nota: es importante el orden en los símbolos >= y <=.
El doble igual == se diferencia de la asignación =
por ejemplo:

```
si (B==10) → condición
C = A + 5 → asignación
```

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.

```
comienzo()
{
ingresar(N);
si (N % 3 == 0)
     imprimir(N,“ es múltiplo de 3”);
sino
     imprimir(N,“ no es múltiplo de 3”);
}
```

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

```
comienzo()
{
ingresar(N);
si (N % 2 == 1)
     imprimir(N,“ es impar”);
sino
     imprimir(N,“ es par”);
}
```

Tabla de estructura de control de decisión

Pseudocódigo	Significado	Ejemplo

`si (condición)`
`{`
`sentencias;`
`}`
	Decisión Simple: Solamente funciona cuando la alternativa es verdadera	`si (nota >= 4)`
		`imprimir(“Aprobó”);`

si (condición)
{
sentencias;
}
sino
{
sentencias;
}
	Decisión Doble: Existen dos alternativas, cuando la condición sea verdadera y cuando sea falsa	si (nota >= 4)
		imprimir(“Aprobó”);
		sino
		imprimir(“Desaprobó”);

si (condición)
si (condición)
{
sentencias;
}
sino
{
sentencias;
}
	Decisión Anidada: Cuando tenemos más de una condición (una dentro de otra).Existen dentro del si y/o dentro del sino	si (nota >= 4)
		si (nota < 7)
		imprimir(“Aprobó”);
		sino
		imprimir(“Promocionó”);
		sino
		imprimir(“Desaprobó”);

según (variable)
{
caso valor:
sentencias;
salir;
.
.
.
defecto:
sentencias;
salir;
}
	Decisión Múltiple: Cuando tenemos muchas alternativas en la condición. Si no se cumple ninguna de las condiciones la opción es por “defecto”.	según (alerta_de_incendio)
		{
		caso 1:
		imprimir(“Bajo: estado verde”);
		salir;
		caso 2:
		imprimir(“Moderado: estado azul”);
		salir;
		caso 3:
		imprimir(“Alto: estado amarillo”);
		salir;
		caso 4:
		imprimir(“Muy Alto: estado naranja”);
		salir;
		defecto:
		imprimir(“Extremo: estado rojo”);
		salir;
		}

Operadores lógicos

Existen tres operaciones lógicas fundamentales:

Conjunción lógica o producto lógico

El operador correspondiente se representa mediante los símbolos «^» (propio de la lógica proposicional) «AND» y «.». En el lenguaje de programación C se utiliza el símbolo «&&».

El efecto de este operador es la evaluación simultánea del *estado de verdad* de la variables lógicas involucradas. Llamamos e*stado de verdad* a uno cualquiera de los dos valores del conjunto lógico. En tal sentido, a los operadores lógicos en general se los denomina *conectores lógicos*, puesto que *conectan* predicados (o variables lógicas).

Así tendremos, por ejemplo, que la expresión: A && B, será verdadera únicamente si A y B lo son. Cualquier otro arreglo de *estados de verdad* para ambas variables dará como resultado el *valor falso*, puesto que basta con que una de la dos variables tenga valor falso, para que ambas no sean simultáneamente verdaderas.

Variables lógicas		**Resultado P && Q**
P	Q	
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

Disyunción lógica inclusiva o suma lógica

El operador correspondiente se representa mediante los símbolos «v» (propio de la lógica proposicional), «OR» y «+». En el lenguaje de programación C se utiliza el símbolo «||».

El efecto de este operador es la evaluación no simultánea del *estado de verdad* de las variables lógicas involucradas. Esto implica que al tener *estado verdadero* por lo menos una de las variables afectadas, la operación dará un resultado verdadero.

Así tendremos que la expresión: A || B será *falsa* únicamente cuando el e*stado* de ambas variables sea *falso*. En cualquier otro caso, la operación será *verdadera*.

Variables lógicas		Resultado
P	Q	P || Q
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Negación o complemento lógico

Este operador representado por un guión sobre la *variable* a *complementar* ejemplo Ā y también la palabra «NOT» al aplicarse a un predicado lógico (simple o compuesto) devuelve el valor opuesto; es decir, si el predicado en cuestión es *falso*, el resultado será *verdadero* y recíprocamente. En el lenguaje de programación C se utiliza el símbolo «!».

Variable Lógica P	Resultado !P
Verdadero	Falso
Falso	Verdadero

Por ejemplo si tenemos la expresión: !(A || B), devolverá como resultado verdadero, únicamente cuando las variables A y B adopten simultáneamente el estado falso.

Ejemplo 16: verificar que el número ingresado esté dentro del intervalo 10 y el 20 (incluidos)

```
comienzo()
{
ingresar(N);
si (N >= 10 && N <=20)
     imprimir(N, " se encuentra en el intervalo");
sino
     imprimir(N," no se encuentra en el intervalo");
}
```

Ejemplo 17: si la temperatura de ambiente es menor a 15º no hay riesgo de incendio, si está entre 15º y 20º hay riesgo bajo, y si hay más de 20º el riesgo es alto. Realizar un algoritmo que comunique el estado según la temperatura ingresada.

```
comienzo()
{
ingresar(T);
si (T < 15)
     imprimir(“No hay riesgo de incendio”);
sino
     si (T >= 15 && T <= 20)
            imprimir(“Hay riesgo bajo de incendio”);
     sino
            si (T > 20)
                   imprimir(“Hay riesgo alto de incendio”);
}
```

Ejemplo 18: mostrar un cartel «Número elegido» para aquellos que cumplen con alguna o las dos condiciones siguientes (que sea par o que sea negativo).

```
comienzo()
{
ingresar(N);
si (N % 2 == 0  || N < 0)
     imprimir(“Número elegido”);
}
```

Ejemplo 19: mostrar un cartel «Número no elegido» para aquellos que no cumplen con alguna de las dos condiciones siguientes (que sea múltiplo de 3 o que sea mayor a 30).

```
comienzo()
{
ingresar(N);
si  ! (N % 3  == 0  || N >=30)
     imprimir(“Número no elegido”);
}
```

Ejercicios: Pseudocódigo-estructura de control de decisión

1. Diseñe un algoritmo que, dados dos números, imprima su suma.
2. Diseñe un algoritmo que imprima el cuadrado y el cubo de un número ingresado.

3. Diseñe un algoritmo que imprima el número siguiente al ingresado.
4. Halle la energía según la masa m y la velocidad de la luz c con la célebre fórmula de Einstein $E=mc^2$
5. Halle el resultado de ex a partir del empleo de la siguiente serie de Taylor: $e^x = 1+\frac{x}{1}+\frac{x^2}{2!}+\frac{x^3}{3!}$, el usuario debe ingresar solo el valor de x
6. Ingrese un número y muestre dos resultados que provengan de éste: aumentado un 10% y disminuido en un 20%
7. Diseñe un algoritmo que imprima el área de un triángulo cuya base y altura se ingresan por teclado. (área = (b * h) / 2)
8. Diseñe un algoritmo que imprima el perímetro de un triángulo cuyos tres lados se ingresan por teclado. (perímetro = a + b + c)
9. Diseñe un algoritmo que, dados tres números, calcule e imprima el promedio.
10. Diseñe un algoritmo que, dadas las longitudes de los lados de un trapecio, calcule e imprima su perímetro.
11. Diseñe un algoritmo que, dado el peso de un objeto en kg, calcule y muestre dicho peso en libras (1 libra es igual a 0.453592 kg).
12. Diseñe un algoritmo que calcule el volumen de un cilindro dado su radio y altura (volumen=2.π.r.h).
13. Diseñe un algoritmo para calcular el porcentaje de hombres y de mujeres que hay en un grupo, dados los totales de hombres y de mujeres.
14. Un profesor corrige un examen y según el puntaje de 1 a 100 (números naturales) debe calificar al alumno respetando la siguiente tabla:

Nota	Puntaje
1	0-29
2	30-47
3	48-59
4	60-65
5	66-71
6	72-77
7	78-83
8	84-89
9	90-95
10	96-100

15. Para que una persona prepare tres equipos de rescate A, B y C. Se sabe que se tarda 15 minutos en preparar el equipo A; 20, con el equipo B y 23, con el equipo C. La cantidad de rescatistas según el equipamiento se ingresa por teclado (cantidad de A, de B y de C). ¿Cuántos minutos tardará dicha persona en preparar los equipos de todos los rescatistas?

16. Ingrese dos números naturales y muestre el menor suponiendo que son distintos.
17. Diseñe un algoritmo que indique con carteles si el número ingresado es negativo, positivo o nulo.
18. Ingrese un número entero y muestre el próximo número par. Por ejemplo si ingresa el 3 (que muestre el 4); si ingresa el 8 (muestra el 10)
19. Diseñe un algoritmo que indique si el número ingresado es par o impar. El n.° ingresado es > 0.
20. Ingrese un número y luego muestre el número consecutivo siguiente al ingresado que sea par.
21. Ingrese un número y luego muestre el número consecutivo siguiente al ingresado que sea impar.
22. Ingrese tres números y muestre el mayor (asuma que los números son distintos).
23. Ingrese tres números y muestre el mayor y el menor (asuma que los números son distintos).
24. Repita el ejercicio anterior pero utilizando el operador lógico Y (&&) en las condiciones para minimizar su cantidad.
25. Diseñe un algoritmo que imprima el número de docena («primera», «segunda» o «tercera») dado el resultado de una jugada de ruleta. Utilice el operador lógico Y (&&).
26. Diseñe un algoritmo que imprima "par" si el valor ingresado es 2, 4, o 6; "impar" si es 1, 3, o 5; y en cualquier otro caso, "error". Utilice el operador lógico O (||).
27. Diseñe un algoritmo que, dado un número, imprima "Verdadero" si está entre 0 y 10, y "Falso" si es mayor a 10 o menor a cero.
28. Ingrese la altura (en cm) y el peso (en kl) de una persona para mostrar su IMC (índice de masa corporal), sabiendo que su cálculo es IMC=100. peso/(altura)^2.
29. Para el ejercicio anterior agregue en la evaluación del IMC un cartel de salida, si es menor a 18,5 "Bajo peso", si se encuentra entre 18,5 y 24.9 (incluídos) "Peso normal", si se encuentra entre 25 y 29,9 (incluídos) "Sobre peso" y mayor a 29,9 "Obesidad".
30. Un empleado cobra según la categoría (A - $ 7.500, B - $ 9.500, C - $ 11.500) y según su antigüedad se le aumenta (<5 años 5 %, ≥5 y <10 años 12 % y el resto, un 20 %). Entonces el programa debe permitir ingresar la antigüedad y la categoría del empleado para mostrar su sueldo definitivo.

31. Se ingresan tres números (A, B, C) para saber si el resultado de la siguiente expresión lógica es verdadero o falso: (A * 3 > B – C) ∧ (C*A = 3). Muestre el resultado ("verdadero" o "falso").
32. Ídem al anterior, pero con la siguiente expresión lógica: ~~(A > 8 - B)~~

Estructura de control de repetición

Hasta el momento hemos visto ejercicios que no presentan repeticiones de sentencias o acciones. Pero son muy comunes estos casos a la hora de programar. Un ejemplo sencillo de repetición podría ser el ingreso de las temperaturas de cada día de un mes para luego hallar el promedio, o la cantidad de venta semanal de un producto en todo el año para hallar la sumatoria, etcétera.

No es lo mismo hallar el promedio de tres números que el de veinte. Veamos justamente cómo podríamos resolver el ejemplo dado con las herramientas que tenemos hasta el momento para tener una mejor idea.

Ejemplo 20: ingresar diez números y hallar su promedio. (Podríamos hacerlo de dos maneras).

Solución 1

```
comienzo()
{
ingresar(A,B,C,D,E,F,G,H,I,J);
P = (A+B+C+D+E+F+G+H+I+J)/10;
imprimir("El promedio es ",P);
}
```

Solución 2

```
comienzo()
{
S = 0;
ingresar (A);
S = S + A;
ingresar (A);
S = S + A;
ingresar (A);
S = S + A;
ingresar (A);
S =S + A;
ingresar (A);
S = S + A;
ingresar (A);
S = S + A;
ingresar (A);
S = S + A;
ingresar (A);
S = S + A;
ingresar (A);
S = S + A;
ingresar (A);
S = S + A;
ingresar (A);
S =S + A;
P = S/10;
imprimir("El promedio es ",P);
}
```

La solución 1 resuelve el problema en pocas líneas pero utiliza una gran cantidad de variables, es decir, mucha memoria, y la solución 2 utiliza pocas variables pero muchas líneas de comando.

Basta imaginar si el algoritmo en vez de resolver el promedio con diez números debiera resolverlo para cien, necesitaríamos para la solución 1 cien variables y para la solución 2, doscientas seis líneas de trabajo.

Para poder revertir dicho problema existen en los lenguajes de programación mecanismos de repetición llamados bucles (*loop*).

Existen tres estructuras o sentencias de control para especificar la repetición: Para, Mientras y Hacer-Mientras.

Estructura de control de repetición (sentencia «para»)

Cuando se desea ejecutar un bucle en un determinado número de veces, usamos la sentencia «para».

En estos casos se requiere que conozcamos por anticipado el número de repeticiones.

Podríamos solucionar el problema del ejemplo 20 al saber que un par de acciones se repite diez veces.

```
comienzo()
{
S = 0;
para (i=0; i<10; i++)
     {
     ingresar(A);
     S = S + A;
     }
P = S/10;
imprimir("El promedio es ",P);
}
```

Al saber que las dos sentencias *ingresar(A) y S = S + A* se repiten diez veces, el bucle «para» envuelve dichas acciones.

La sentencia *para* tiene tres parámetros separados por *;* (punto y coma): el primero *i=0* indica el comienzo con valor inicial cero, el segundo indica la condición que debe cumplir, en estos casos debe ser siempre inferior a diez, y el tercero indica el incremento que debe sufrir la variable *i*, en estos casos *i++* que significa *i = i + 1*, es decir que por cada iteración del bucle se incrementa *i* en uno.

En la primera iteración *i = 0*, luego en la segunda iteración *i = 1*, tercera iteración *i = 2*,.., en la décima *i = 9* y para la decimoprimera vuelta *i = 10*, es decir no cumple con la condición impuesta por el parámetro dos *i<10* de la sentencia *para*, entonces el bucle se termina en ese momento.

Pudo haberse utilizado *para (i=1; i<=10; i++)* siendo equivalente. La variable *i* comienza en uno pero la iteración termina cuando el valor de *i* llega a once.

A dicha variable *i* se la reconoce como *variable de iteración*. No necesariamente debe llamarse así, podría etiquetarse de cualquier otra manera.

También es importante saber que los resultados finales deben ubicarse fuera del ciclo de repetición, por el contrario, estaríamos mostrando en el ejemplo diez veces "El promedio es "...

Ejemplo 21: hallar el promedio de altura de los alumnos del curso A y el promedio de los alumnos del curso B y mostrar sus resultados. El curso A tiene 26 alumnos mientras que el B tiene 30.

```
comienzo()
{
SA = 0;
SB = 0;
para (i=0; i<26; i++)
     {
     ingresar (altura);
     SA = SA + altura;
     }
para (i=0; i<30; i++)
     {
     ingresar (altura);
     SB = SB + altura;
     }
PA = SA/26;
PB = SB/30;
imprimir("El promedio de altura del curso A es ",PA);
imprimir("El promedio de altura del curso B es ",PB);
}
```

Nota: en estos casos se utilizan dos sentencias *para* separadas, por tener distintas cantidades de iteraciones (26 y 30). Puede utilizarse la misma variable de iteración *i* para los dos *para*, ya que su contenido

vuelve a iniciarse con cero. Lo mismo sucede con la variable *altura* que se repite en ambos ingresos, aunque también pudo haberse utilizado diferentes variables.

Las variables *SA* y *SB* del ejemplo 21 son llamadas **variables sumadoras o acumuladoras**. A continuación explicaremos este tipo de variables especiales.

Variables contadoras y sumadoras

Variable contadora

Se trata de una variable que se encuentra en ambos miembros de una asignación a la que se le suma un valor constante (por lo general uno). Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante, cada vez que se produce un determinado suceso, acción o iteración. Los contadores se utilizan con la finalidad de contar sucesos, acciones o iteraciones internas en un bucle, proceso, subrutina o donde se requiera cuantificar; deben necesariamente ser inicializados antes del comienzo de un ciclo de repetición.

La inicialización consiste en asignarle al contador un valor inicial, es decir, el número desde el cual necesitamos que se inicie el conteo (por lo general comienzan en cero).

Variable sumadora

También llamada *acumuladora*, es una variable que se encuentra en ambos miembros de una asignación a la que se le suma un valor variable. Se trata de una variable que, como su nombre lo indica, suma sobre sí misma un conjunto de valores, al finalizar con el ciclo contendrá, en una sola variable, la sumatoria de todos los valores que cumplen una determinada condición (también puede servir para decrementar valores variables). Al igual que el contador es necesario haber inicializado antes del comienzo de un ciclo de repetición.

La inicialización consiste en asignarle al sumador un valor inicial, es decir el número desde el cual necesitamos que se inicie la sumatoria (por lo general comienzan en cero).

Nota: la diferencia entre un contador y un sumador es que mientras el primero aumenta en una cantidad fija preestablecida, el acumulador aumenta en una cantidad o valor variable.
Y la similitud está en que siempre deben tener un valor inicial antes del bucle de repetición.
Seguramente cuando tengamos que hallar promedios o porcentajes necesitaremos de ambas.

A continuación, unos ejemplos:

Variable contadora	Variable sumadora o acumuladora

comenzar()
{
C = 0;
para (i=0; i < 100; i++)
si (i % 2 ==0)
C = C +1; →(lo mismo si C++)
imprimir(“La cantidad de nº pares son “,C);
}
	comenzar()
	{
	S = 0;
	para (i=0; i < 100; i++)
	{
	ingresar(A);
	S = S + A;
	}
	P = S/100;
	imprimir(“El promedio de altura es “,P);
	}

Ejemplo 22: en la salida de un parque nacional se desea saber el promedio de edad de los primeros 100 visitantes que egresan de éste, pero separando los turistas extranjeros de los turistas nacionales.

```
comienzo()
{
SE = 0;
CE = 0;
SN = 0;
CN = 0;
para (i=0; i<100; i++)
     {
```

```
        ingresar (edad,argentino);
        si (argentino == 'S')
                {
                SN = SN + edad;
                CN = CN + 1;
                }
        sino
                {
                SE = SE + edad;
                CE = CE + 1;
                }
        }
   PN = SN/CN;
   PE = SE/CE;
   imprimir ("El promedio de edad de los turistas argentinos es
",PN);
   imprimir ("El promedio de edad de los turistas extranjeros es
",PE);
   }
```

> **Nota:** en este caso dentro del bucle *para* se encuentra la decisión doble *si* para determinar si la edad ingresada es de un turista nacional o extranjero. Entonces se separan los caminos, si es verdadera la condición *(argentino == 'S')* entonces toma el camino del *si*, caso contrario, el camino del *sino*.
> Como se puede observar las variables también pueden guardar datos alfanuméricos ('A'...'Z', 'a'...'z',0...9).

Al no saber de entrada la cantidad de turistas nacionales y extranjeros, fue necesario usar *CE* y *CN* llamadas *variables contadoras*, ya que cada vez que pasan por *CE = CE + 1* y *CN = CN + 1* sus valores se incrementan en uno. Éstas, al igual que las sumadoras, deben inicializarse en cero antes del ciclo de repetición.

> **Aclaración:** en la asignación de variables contadoras puede escribirse de un modo reducido.
> Por ejemplo *CN = CN + 1* es equivalente a *CN++*
> A partir de ahora lo utilizaremos del modo reducido.

Es posible decrementar el valor de la variable de iteración.

Ejemplo 23: mostrar de modo decreciente los números del N al 1. El usuario debe ingresar dicho número.

```
comienzo()
{
ingresar(N);
para (i=N; i>0; i--)
     {
     imprimir(i);
     }
}
```

> **Nota:** por ejemplo, si ingresamos el 8, el resultado del algoritmo será: 87654321, pero si le agregamos a la sentencia imprimir un espacio en blanco *imprimir(i," ");* entonces el resultado será 8 7 6 5 4 3 2 1 y quedará legible.
> Además puede observarse que en la sentencia *para* la variable de iteración i comienza con el valor N (ingresado por el usuario) y se va decrementando con *i--* mientras se cumpla la condición *i>0*.

Bucle para anidado

Al igual que sucede con la sentencias de decisión si, los bucles pueden estar anidados, es decir uno dentro de otro. Se debe tener cuidado con que el bucle interior esté completamente dentro del bucle exterior. Si se cruzan, se produce un error. En pocas palabras, hay que colocar correctamente las llaves que abren y cierran.

A continuación se muestra en un cuadro un ejemplo genérico que compara lo incorrecto con lo correcto para los cierres de llaves *{}* en bucles *para*.

Incorrecto	Correcto

`para (i=0; i<N; i++)`
`    {`
`      para (j=0; j<M; j++)`
`        }`
`        {`
`    }`
	`para (i=0; i<N; i++)`
	`    {`
	`       para (j=0; j<M; j++)`
	`       {`
	`       }`
	`    }`

Ejemplo 24: realizar un algoritmo que produzca por pantalla lo siguiente:

1	→	100	110	120	130	140	150
2	→	100	110	120	130	140	150
3	→	100	110	120	130	140	150
4	→	100	110	120	130	140	150
5	→	100	110	120	130	140	150
6	→	100	110	120	130	140	150
7	→	100	110	120	130	140	150
8	→	100	110	120	130	140	150
9	→	100	110	120	130	140	150
10	→	100	110	120	130	140	150

```
comienzo()
{
para (i=1; i <= 10; i++)
     {
     imprimir(i," ----> ");
     para (j=100; j<=150; j=j+10)
             {
             imprimir(j,"   ");
             }
     imprimir;
     }
}
```

Nota: el primer bucle *para* se incrementa en uno y el segundo, de diez en diez *j = j + 10*, comenzando con *j=100*, mientras sea *j <= 150*. En este caso sí es importante tener distintas variables de iteración *(i , j)*.
También es importante ver cómo fueron cerrados los bucles por sus llaves. Observemos que la primera sentencia *imprimir(i," ---->");* se realiza para la primera columna de la pantalla, el número de iteración y una flecha, luego dentro del segundo bucle *para* se imprimen los cinco números del 100 al 150. El último *imprimir;* que está fuera del segundo pero dentro del primero, sirve para hacer el salto de línea, como si fuera un ENTER. Si no fuera por esta última instrucción, la pantalla tendrá el siguiente aspecto: 1 → 100 110 120 130 140 150 2 → 100 120 130 140 150 3 → 100 ... 10 → 100 110 120 130 140 150
Es decir, todo en una única línea.

El ejemplo anterior muestra cómo se utilizan dos *para* anidados, su uso es frecuente cuando se trabaja con matrices.

Es posible anidar más de dos bucles *para*.

Máximos y mínimos

Para hallar el valor máximo y/o mínimo de una lista de valores numéricos es necesario utilizar una variable adicional. Mostramos un ejemplo a continuación.

Ejemplo 25: hallar la temperatura máxima registrada en una laguna sabiendo que se toman 30 muestras.

```
comienzo()
{
Tmax = -10000;
para (i=0; i < 30; i++)
     {
     ingresar(T);
     si (T > Tmax)
             Tmax = T;
     }
imprimir("La mayor temperatura registrada es ",Tmax);
}
```

> **Nota:** el valor inicial de la variable *Tmax* es un número extremadamente pequeño (un absurdo para la temperatura de una laguna), cuando se ingresa el primer valor de temperatura dentro del bucle, la condición *si* se efectúa al ser verdadera, entonces el valor de *Tmax* cambia por el valor de la temperatura *T*.
> A medida que se ingresan las otras temperaturas, la sentencia *si* evalúa si aparece un valor mayor o no a *Tmax*, en caso afirmativo, se le asigna a éste el nuevo valor de *T*.
> Tomar en cuenta que si se produce el caso de empate la condición del *si* es falsa, por lo tanto, no sufre efecto.

Si por lo contrario, el ejercicio tratara de hallar la mínima temperatura de la laguna, entonces debemos cambiar *Tmax* por *Tmin*, asignarle un

valor extremadamente grande (un absurdo, por ejemplo *Tmin = 10000*) y la condición de la decisión *si* será *si (T<Tmin)*. A continuación, mostramos un ejemplo de cómo se halla ambas.

Ejemplo 26: hallar la temperatura máxima y mínima registrada en una laguna sabiendo que se toman 30 muestras.

```
comienzo()
{
Tmax = -10000;
Tmin = 10000;
para (i=0; i < 30; i++)
     {
     ingresar(T);
     si (T > Tmax)
            Tmax = T;
     si (T < Tmin)
            Tmin = T;
     }
imprimir("La mayor temperatura registrada es ",Tmax);
imprimir("La menor temperatura registrada es ",Tmin);
}
```

> **Nota:** observar que las sentencias ***si*** trabajan de modo independiente al no ser excluyentes. Podría pasar que tanto el máximo como el mínimo coincidan, esto se produce cuando las treinta temperaturas son exactamente iguales.
> ¿Qué sucede si además de registrar la máxima temperatura de la laguna, quisiéramos saber cuál de todas las muestras se registró?

Ejemplo 27: hallar la temperatura máxima registrada en una laguna sabiendo que se toman 30 muestras y en qué muestra fue registrada.

```
comienzo()
{
Tmax = -10000;
para (i=0; i < 30; i++)
     {
```

```
        ingresar(T);
        si (T > Tmax)
                {
                Tmax = T;
                Tmuestra = i+1;
                }
        }
   imprimir("La mayor temperatura registrada es ",Tmax,"  y se re-
gistró en la muestra nº ",Tmuestra);
   }
```

> **Nota:** a la variable *Tmuestra* se le asigna el valor de la variable de iteración *i* del *para* más uno, ya que empieza del cero. Por ejemplo si la temperatura máxima se produce en la vuelta 8 significa que i tiene valor 7 en dicha vuelta, entonces es necesario incrementarlo.
> De esta manera, se tiene el registro de cuando se produce la última condición verdadera de la sentencia *si*.
> En caso de empate el único registro que se produce es el primero que llega.

Por ejemplo, si se ingresan los siguientes valores en *Tmuestra*:

i	0	1	2	3	4	5	...	25	26	27	28	29
Tmuestra	7	4	6	11	9	8	...	5	10	11	8	6

Notamos que en la cuarta muestra (cuando *i* es igual a 3) la temperatura es máxima hasta el momento, pero sucede lo mismo en la vigesimoctava posición (cuando *i* es igual a 27), es decir, tenemos un caso de empate, pero el que se toma en cuenta es el primero de los máximos. Finalmente *Tmuestra* termina con el valor 4, recordando que a *i* se le incrementa de a uno *Tmuestra = i + 1*

Estructura de control de repetición (sentencia «mientras»)

Esta estructura repite una instrucción o grupo de instrucciones mientras una expresión lógica sea cierta.

Cuando no se conoce el número de repeticiones por anticipado, la sentencia *mientras* lo resuelve con el empleo de una determinada condición.

Lo primero que hace esta sentencia es evaluar si se ejecuta el bucle, es decir que es posible que nunca se realice acción alguna.

Mientras la condición se cumpla, el bucle sigue iterando, por eso es importante no caer en ciclos de repetición infinitos.

Ejemplo 28: calcular la suma de números ingresados por teclado hasta que se ingrese un cero.

```
comienzo()
{
S = 0;
ingresar(N);
mientras ( N != 0)
     {
     S = S + N;
     ingresar(N);
     }
imprimir("La sumatoria es ",S);
}
```

> **Nota:** es necesario ingresar *N* antes de arrancar con el *mientras*, porque debe evaluarse la condición al entrar al bucle *mientras*, luego es importante no olvidarse de ingresar *N* dentro del bucle, porque si no lo hacemos caeremos en un ciclo de repetición infinito.
> ¿Cuál será el resultado si arrancamos con el ingreso de un cero?

Dentro del *mientras* pueden utilizarse condiciones compuestas como veremos a continuación.

Ejemplo 29: hallar el promedio de números ingresados por teclado hasta que aparezca uno que no sea par y mayor a 20.

```
comienzo()
{
S = 0;
C = 0;
ingresar(N);
mientras ( !( (N % 2 == 1) && (N > 20) ) )
     {
     S = S + N;
```

```
     C++;
     ingresar(N);
     }
P = S/C;
imprimir("El promedio es ",P);
}
```

Nota: como no sabemos de entrada la cantidad de números que se van a ingresar, entonces debemos usar un contador *C* para poder finalmente hallar el promedio.
El inconveniente que puede presentar este algoritmo es que de entrada no se cumpla la condición del *mientras*, por lo tanto el valor de *C* terminaría en cero y generaría un error en la división al hallar el promedio *P*. Esto se podría resolver agregando antes de dicho cálculo la siguiente sentencia *si*.

```
comienzo()
{
S = 0;
C = 0;
ingresar(N);
mientras ( !( (N % 2 == 1) && (N > 20) ) )
     {
     S = S + N;
     C++;
     ingresar(N);
     }
si(C == 0)
     imprimir("No hubo números de ingreso");
sino
     {
     P = S/C;
     imprimir("El promedio es ",P);
     }
}
```

Ejemplo 30: contar la cantidad de veces que se ingresan números pares y la cantidad de números impares hasta que se ingrese un número negativo. El cero no se cuenta.

```
comienzo()
{
CP = 0;
CI = 0;
ingresar(N);
mientras ( N > = 0 )
     {
     si ((N % 2 == 0) && (N != 0))
            CP++;
     sino
            si ((N % 2 == 1) && (N != 0))
                   CI++;
     ingresar(N);
     }
imprimir("La cantidad de números pares es ",CP);
imprimir("La cantidad de números impares es ",CI);
}
```

Nota: mientras *N* no sea negativo el bucle sigue iterando. Para incrementar los contadores se les agregó a las condiciones *(N != 0)* porque el enunciado del problema así lo determinaba.
Al igual que la sentencia *para* y la sentencia *si*, pueden anidarse los *mientras*.

Estructura de control de repetición (sentencia «hacer – mientras»)

Esta estructura repite una instrucción o grupo de instrucciones hasta que una expresión lógica sea cierta.

Un aspecto muy importante de la presente estructura de control es que la expresión lógica no se evalúa hasta el final de la estructura con lo cual el bucle se ejecuta al menos una vez, contraria a la estructura «mientras» que podía no ejecutarse nunca. Es decir que después de cada iteración el bucle evalúa la condición, si es verdadera sigue repitiendo y si la condición es falsa termina.

Ejemplo 31: hacer un algoritmo que muestre del 0 al 20 (solo los números pares).

```
comienzo()
{
N = 0;
hacer
```

```
     {
     imprimir(N," “);
     N = N + 2;
     }
mientras ( N < 21 );
}
```

Nota: a medida que itera, el bucle se incrementa *N* de dos en dos, en la última repetición se imprime 20 y se incrementa en 22, por lo tanto, termina porque la condición ahora es falsa.

Ejemplo 32: en un bosque se necesita saber el promedio de diámetro de cada tronco de ciprés y el promedio de su altura. El proceso termina cuando el usuario responde con una 'N', mientras tanto, debe responder con 'S'.

```
comienzo()
{
C = 0;
SA = 0;
SD = 0;
hacer
     {
     ingresar(A,D);
     SA = SA + A;
     SD = SD + D;
     C++;
     imprimir("¿Desea seguir ingresando datos? S/N“);
     ingresar(opción);
     }
mientras ( opción == 'S' );
PA = SA/C;
PD = SD/C;
imprimir ("El promedio de altura de los cipreses es “,PA);
imprimir ("El promedio de diámetro de los cipreses es “,PD);
}
```

Nota: la variable *opción* está reservada para que el usuario responda S o N (si quiere seguir ingresando o no).

Tomar en cuenta que para seguir ingresando datos el usuario debe presionar solo la S (mayúscula). Para resolver este problema podría ponerse la siguiente condición en el *mientras*
(opción == 'S' && opción == 's')
Por otra parte, este tipo de estructura sirve para que al menos se ingresen los datos de un árbol, por eso no es necesario verificar si el denominador es nulo en los promedios.

Al igual que en las anteriores estructuras, es posible crear algoritmos que tengan bucles «hacer-mientras» una dentro de otra.

Tabla de estructura de control de repetición

Pseudocódigo	Significado	Ejemplo

`para (valor inicial; condición;incremento/ decremento)`
`{`
`sentencias;`
`}`
	Bucle de repetición exacto.
	Se sabe de antemano la cantidad de iteraciones que cumple el bucle.
		`para(i=0; i < 10; i++)`
		`{`
		`imprimir("3 X ",i,"→",i*3);`
		`}`
		Imprime la tabla de multiplicación del 3.

`mientras(condición)`
`{`
`sentencias;`
`}`
	Bucle de repetición condicional.
	Mientras se cumpla la condición se repiten las sentencias dentro del bucle. Puede ocurrir que no exista ejecución de sentencias de entrada.
		`i = 1;`
		`mientras (i<=10)`
		`{`
		`imprimir("3 X ",i,"→",i*3);`
		`i++;`
		`}`
		Imprime la tabla de multiplicación del 3.

hacer
{
sentencias;
}
mientras(condición);
	Bucle de repetición condicional.
	Mientras se cumpla la condición se repiten las sentencias dentro del bucle. Pero siempre se ejecuta una vez como mínimo, porque la condición se encuentra al finalizar las sentencias.
		i = 1;
		hacer
		{
		imprimir("3 X "
		,i,"→",i*3);
		i++;
		}
		mientras (i<=10);
		Imprime la tabla de multiplicación del 3.

Atención: en muchos casos da lo mismo utilizar cualquier método de repetición.
Recordar que las variables contadoras y sumadoras deben inicializarse antes del comienzo de la sentencia de repetición y, para hallar los promedios, después de su ejecución.

Ejercicios: Pseudocódigo-estructura de control de repetición

Bucle de repetición para:

33. Imprima los números del 10 al 1.
34. Imprima las tres primeras potencias de los números del 1 al 10.
35. Dado un número imprima su tabla de multiplicar (del 0 a 10).
36. Escriba un programa que pida al usuario un carácter y un número de repeticiones. Luego imprima el carácter el número de veces especificado en una sola línea.
37. Imprima dos columnas, una en centímetros y la otra en su equivalente en pulgadas. Los centímetros se ingresan por teclado. La cantidad total es N.
38. Imprima la cantidad de personas mayores de edad (>=18) de un total de N personas.
39. Imprima la cantidad de mujeres menores de edad de un total de N personas. Muestre también su porcentaje.
40. Imprima la cantidad de mujeres mayores de edad y la cantidad de hombres mayores de edad de un total de N personas. Muestre también sus porcentajes.

41. A un grupo de 10 personas se le consulta la edad y se desea calcular su promedio de edad. Muestre el promedio y cuántas de las 10 personas son mayores de 18 años.
42. Se desea conocer el peso acumulado de 40 personas.
43. Se desea conocer el peso promedio de 20 personas.
44. Se desea conocer el peso acumulado de 15 materiales metálicos en libras (los datos se reciben en kg) Una libra → 0,45359237 kg.
45. En un depósito se reciben 10 barriles de lubricantes para una fábrica de rulemanes y se desea conocer el volumen total que ocuparán. Existen 4 tipos de barriles (de 25, 40, 50 y 100 litros) que se identifican con A, B, C y D, respectivamente.
46. Ingrese 10 temperaturas y muestre la mayor.
47. Se ingresan 10 pares de temperaturas (T1 y T2). Halle el promedio de las temperaturas T1 y el promedio de las temperaturas T2.
48. Se ingresan 10 pares de temperaturas (T1 y T2) para hallar el promedio de las temperaturas que están entre 5° y 15° (incluidos)
49. Se ingresan 10 números para hallar tres datos: a) la cantidad de números negativos, b) la suma de los números que se encuentran entre el 1 y el 10 (no incluidos), c) el promedio de todos los números.
50. Se ingresan 50 datos (peso y edad) de ratones, mostrar la edad del ratón de mayor peso y la edad del de menor peso.

Bucle de repetición mientras o hacer-mientras

51. Se ingresan números enteros al azar para hallar y mostrar lo siguiente:
    a. Cantidad de números múltiplos de 7 que se encuentren entre el 21 y el 63
    b. Promedio total (solo de los números positivos)
    c. Porcentaje de números pares e impares (por separado)

    El algoritmo finaliza cuando ingresamos el 999 y puede que no tengamos ingresos.
52. Ídem al anterior pero tomando en cuenta que al menos existe un ingreso (distinto al 999)
53. En la universidad se registran los siguientes datos por alumnxs: edad, cantidad de materias finales aprobadas y cantidad de materias finales desaprobadas. Se necesita hallar y mostrar lo siguiente:
    a. Cantidad de alumnxs mayores a 21 años de edad que tienen más de dos materias finales desaprobadas.

b. Porcentaje de materias aprobadas y materias desaprobadas (por separado) de todxs lxs alumnxs.
c. Mostrar la edad de las personas que tienen el triple de materias aprobadas con respecto a las que desaprobaron. (se debe hacer mientras se efectúa el bucle)

Elija el método de salida del bucle. Al menos debe registrarse los datos del alumnx.

54. Se ingresan 100 números de la ruleta (0,1,2,...,36), para hallar y mostrar lo siguiente:
    a. Cantidad de números impares.
    b. Promedio de los números pares (no contar los ceros).
    c. Cantidad de números que se encuentran en la segunda docena (13 al 24).
    d. El número más grande.
55. Calcular el promedio de edad de un grupo de personas e indicar cuántas son mayores de 18 años. El algoritmo cierra el grupo cuando hay 5 menores.
56. Hacer un algoritmo que muestre el peso acumulado de un grupo de personas e indique la cantidad de éstas. El grupo se cierra cuando el peso acumulado supera los 500 kg.
57. Hacer un algoritmo que muestre el peso acumulado de un grupo de personas e indique la cantidad de éstas. El grupo nunca debe superar los 500 kg. (Si al ingresar una nueva persona al grupo se superan los 500 kg, se descarta a la persona y se cierra el grupo).
58. Ingresar temperaturas hasta que el promedio sea mayor que 20 grados, y mostrar la menor y la mayor.
59. Ingresar temperaturas mientras el promedio sea menor que 20 grados, y mostrar la menor y la mayor.
60. Se ingresan pares de temperaturas (T1 y T2). Hallar el promedio de las temperaturas T1 y el promedio de las temperaturas T2 hasta que la suma del par ingresado sea mayor que 40 grados.
61. Se ingresan pares de temperaturas (T1 y T2) hasta que T1 sea cero. Hallar el promedio de las temperaturas que están entre 5° y 15° (incluidos).
62. Se ingresan números hasta que la suma alcance los 1.000. Hallar:
    a. La cantidad de números negativos.
    b. La suma de los números que se encuentran entre el 1 y el 10 (incluidos).
    c. El promedio de los mayores a 10.

63. Se ingresan el peso y la edad de ratones, hasta que aparezca un ratón cuya edad sea el doble que el promedio de los ya ingresados (debe ingresar al menos un ratón). Mostrar: a) la edad del ratón de mayor peso; b) la edad del de menor peso, c) el promedio de las edades.
64. Un profesor corrige varios exámenes de sus alumnos y según el puntaje de 1 a 100 (números naturales) debe calificar al alumno respetando la siguiente tabla:

1	1-29
2	30-47
3	48-59
4	60-65
5	66-71
6	72-77
7	78-83
8	84-89
9	90-95
10	96-100

A medida que obtiene el resultado mostrar el puntaje y su condición: Reprobó (menos de 4), Aprobó (mayor o igual a 4 y menor a 7) y Promocionó (mayor o igual a 7).
Además se pide:

a. Los tres porcentajes (reprobados, aprobados y promocionados).
b. El mejor y el peor puntaje.
c. Promedio de nota total.

Nota: cuando aparezca un CERO se termina el proceso.

Ejercicios combinados:

65. Tenemos una lista con 350 triángulos rectángulos. Por cada uno se ingresan sus dos catetos en cm. Finalmente necesitamos saber y mostrar:
    a. Cantidad de triángulos que tienen sus dos catetos iguales
    b. La hipotenusa más grande (mostrar tamaño y número de triángulo según el orden en que ingresó)
    c. Promedio de área de todos los triángulos

    Nota: La hipotenusa no es parte del ingreso, por lo tanto se debe calcular.

66. Un nutricionista registra datos de sus pacientes niñxs para su estadística. Finalmente lo que desea:
    a. Mostrar la/s altura/s de las niñas que pesan entre 35 y 50 kg (incluidos) y son menores a 10 años.
    b. Obtener y mostrar los tres porcentajes de todxs, según su estatura: i) menores a 140 cm, ii) entre 140 y 170 cm (incluidos), iii) más de 170 cm.
    c. Obtener y mostrar el promedio de peso de los niños (varones)

    Nota: No se sabe la cantidad de pacientes.
67. En la caminera de Bariloche se necesita hacer un relevamiento de autos que entran a la ciudad en un día: Se necesita saber y mostrar lo siguiente:
    a. Cantidad de autos con patente Argentina.
    b. Promedio de edad de conductores procedentes de Chile
    c. La mayor estadía (cantidad de días en Bariloche) y el número, según el ingreso del auto

    Nota: para el caso c) es necesario ingresar para cada auto la cantidad de días que se quedan en nuestra ciudad. Al menos un auto ingresa ese día.
68. Un pediatra registra datos de 210 pacientes para su estadística. Finalmente lo que desea:
    a. Obtener y mostrar el porcentaje de niños y de niñas.
    b. Mostrar el peso y la edad de niños (varones) de entre 8 y 11 años de edad (incluidos) con estatura superior a 150 cm.
    c. Obtener y mostrar el promedio del IMC de todos pacientes.

    Nota: En los ingresos el peso es en kilogramos y la estatura en cm. El IMC es la fórmula del índice de masa corporal
69. Tenemos una lista con 120 circunferencias. Por cada una se ingresa su radio en cm. Finalmente necesitamos saber y mostrar:
    a. Cantidad de circunferencias cuyo radio se encuentra dentro del siguiente intervalo (3 y 9]
    b. El promedio total del perímetro.
    c. La sumatoria de áreas (solo de las circunferencias que se ubican en el orden impar en su ingreso)

    Nota: El perímetro y el área de la circunferencia deben ser calculados por el algoritmo

70. En la salida de un museo a cada visitante se les hace una serie de preguntas para saber y mostrar al final del día lo siguiente:
    a. Cantidad de menores de edad (menores a 18 y sin importar el género)
    b. Porcentaje de mujeres.
    c. Promedio de edad de los mayores. (Sin importar el género)
    d. La mayor cantidad de veces que una persona visito el museo y si se trata de un extranjero o argentino.

    Nota: Se debe validar el promedio y el porcentaje (evitar la división por cero)

Capítulo 3: Codificación

Hasta el momento hemos desarrollado algoritmos con pseudocódigo para poder resolver distintos problemas. Para ejecutar los algoritmos en la computadora es necesario codificarlos.

Realizar la conversión del algoritmo a lenguaje de programación implica que no solo se sustituyan las palabras reservadas en español por sus homónimos en inglés, sino también que se agreguen detalles como la declaración de variables, constantes y librerías que son omitidas por el pseudocódigo.

La edición en codificación queda almacenada en lo que llamamos programa fuente, y en particular el Lenguaje C utiliza archivos con extensión .c, es decir *nombre_archivo.c*

Antes de transitar por la codificación, es importante conocer la estructura general en orden de un programa básico en lenguaje C.

Inclusión de librerías
Declaración de constantes
Comienzo de programa
Declaración de variables
Cuerpo del programa

Inclusión de librerías

Las librerías o bibliotecas son archivos que se encuentran en la cabecera de los programas. Básicamente, cada una de las librerías está compuesta por rutinas estandarizadas.

Se trata de funciones que realizan operaciones y cálculos de uso frecuente y son parte de cada compilador.

Las librerías contienen código y datos que proporcionan servicios a programas independientes, es decir, pasan a formar parte de estos.

El programador debe invocar todos aquellos archivos o bibliotecas que necesite.

La extensión de un archivo de librería es .h, es decir *nombre_librería.h*

El modo de incluir una librería es de la siguiente forma:

`#include < librería.h >`

La librería que nunca puede faltar es *stdio.h* que contiene los prototipos de funciones y los tipos de datos para manipular sus entradas y salidas. Es decir que siempre comenzaremos por incluir en nuestros programas la siguiente línea: **#include < stdio.h >**

Existen otras librerías, a continuación veremos algunas de las más importantes:

`#include < stdlib.h >`

Contiene tipos, macros y funciones para la conversión numérica, generación de números aleatorios, búsquedas y ordenación, gestión de memoria y tareas similares.

`#include < string.h >`

Contiene los prototipos de las funciones y macros de clasificación de caracteres.

`#include < math.h >`

Contiene los prototipos de las funciones y otras definiciones para el uso y manipulación de funciones matemáticas.

`#include < time.h >`

Contiene los prototipos de las funciones, macros, y tipos para manipular la hora y la fecha del sistema.

> **Atención:** en un programa puede utilizarse más de una librería y el orden es indistinto, siempre y cuando se ubiquen en la cabecera. En el pseudocódigo no escribíamos este dato.

Declaración de Constantes

La sintaxis para definir las constantes en el lenguaje C es de la siguiente manera:

```
#define nombre_constante  valor_constante
```

Por ejemplo si queremos incluir la constante π en el programa, la sintaxis es la siguiente:

```
#define PI  3.1416
```

Si se tratara de más de una constante, éstas se definen una debajo de la otra.

> **Atención:** la declaración de constantes no necesita ; (punto y coma) para finalizar la línea de comando.

Comienzo del programa

En el pseudocódigo lo llamábamos de la siguiente manera ***comienzo()*** en codificación será:

```
main()
```

Declaración de variables

Este dato en el pseudocódigo era invisible, porque se daba por entendida dicha declaración.

La sintaxis es:

```
tipo_variable   nombre_variable;
```

Ejemplos:

```
int  A;
int A,B;
float suma;
char C1;
float S1,S2;
```

> **Atención:** en este lenguaje no puede quedar ninguna variable sin declarar. Además es importante saber que se diferencian las minúsculas de las mayúsculas.

Tipos de datos

Por *tipo de datos* entenderemos la clase que agrupa datos con idénticas capacidades de operación.

Existen diferencias entre los distintos lenguajes de programación en cuanto a los Tipos de Datos que se utilizan, las operaciones permitidas entre ellos y el modo en que pueden organizarse (estructuras más complejas).

En principio veremos los tipos de *datos primitivos* o *básicos*. Cada *tipo de dato primitivo* o *básico* define, como se mencionó anteriormente, el conjunto de valores que puede asumir una variable, así definida.

Los *tipos de datos básicos* que maneja el lenguaje son: numérico y carácter.

Estos *tipos* se aplican tanto a las variables como a las constantes.

A continuación, se analizará cada uno de los distintos *tipos básicos* mencionados:

Tipo numérico

Básicamente se distinguen dos *subtipos*:

- Tipo numérico entero.
- Tipo numérico real.

Los *datos numéricos enteros* se denotan con *int* (*integer* en inglés, entero en español) y *long* (largo en inglés, es decir entero largo).

Su sintaxis es:

```
int  nombre_variable;
long  nombre_variable; → para valores muy grandes y/o muy chicos
```

La representación de los *datos numéricos reales* apela a un punto decimal que separe la mantisa de la parte entera de la parte decimal. Se denotan con la palabra *float,* que significa punto flotante y *double* que es el punto flotante largo.

Su sintaxis es:

```
float  nombre_variable;
double  nombre_variable; → para valores muy
grandes y/o muy chicos
```

Tipo carácter

Involucra un conjunto ordenado y finito de símbolos que el procesador puede reconocer.

Si bien no existe un conjunto estándar, podemos decir que dicho conjunto está básicamente integrado por:

1. Letras mayúsculas (desde la A hasta la Z), sin incluir la CH y la LL (eventualmente puede no ser incluida la Ñ).
2. Letras minúsculas (desde la a hasta la z), con las mismas restricciones que para las mayúsculas.
3. Dígitos (del 0 al 9).
4. Caracteres especiales. Están incluidos aquí símbolos tales como:

*,+,-,:,;,,,”,/,<,>,=,|,\,~,@,#,$,.,%,^,&,(,),{,},[,],`,!,?,’ y otros. El espacio en blanco, también es considerado como un carácter.

La sintaxis es:

```
char  nombre_variable;
```

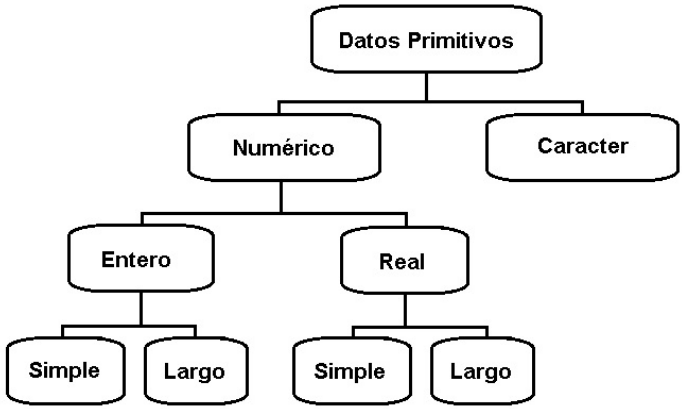
Char es la reducción de la palabra en inglés *character*, que significa carácter.

Atención: en C los caracteres van delimitados por comillas simples como en el lenguaje algorítmico. Por ejemplo: *categoría = ‘A’*

A continuación mostramos una tabla con los tipos de datos básicos y sus rangos de valores:

TIPO	ANCHO EN BIT	RANGO EN PC
char	8	-128 a 127
Int	16	-32768 a 32767
long	32	-2147483648 a 2147483647
float	32	3.4E-38 a 3.4E+38
double	64	1.7E-308 a 1.7E+308

Podemos sintetizar los tipos de datos primitivos mediante el esquema que se presenta a continuación:



Cuerpo del programa

Esta es la sección donde aparece el algoritmo. Es aquí donde se encuentra la traducción del programa escrito en pseudocódigo a código.

El cuerpo del programa se encuentra encerrado entre las llaves { } del main().

A continuación veremos un ejemplo muy sencillo de un algoritmo en pseudocódigo y su codificación:

Ejemplo 1: mostrar el siguiente cartel "Bienvenidos a la Codificación".

Pseudocódigo	Código

comienzo()
{
imprimir ("Bienvenido a la codificación");
}
	#include <stdio.h>
	main()
	{
	printf("Bienvenidos a la Codificación");
	}

Se observa que en el código se agrega la librería **stdio.h** en la primera línea, **comienzo** se reemplaza por **main**, las llaves quedan iguales y la función **imprimir** se traduce al código con **printf**

> **Atención:** Es importante saber que las palabras reservadas del código se escriben en minúsculas.

Ejemplo 2: mostrar la suma de tres números siendo: A = 5, B = 8 y C = 1.

Pseudocódigo	Código

comienzo()
{
A = 5;
B = 8;
C = 1;
suma = A + B + C;
imprimir (“La suma es ”,suma);
}
	#include <stdio.h>
	main()
	{
	int A, B, C, suma;
	A = 5;
	B = 8;
	C = 1;
	suma = A + B + C;
	printf(“La suma es %i”,suma);
	}

En estos casos se observa que en el código se declaran las variables *A, B, C* y *suma* de tipo entero (int), invisible para el pseudocódigo.

La declaración de variables puede realizarse por separado. Para el ejemplo anterior será:

```
int A;
int B;
int C;
int suma;
```

Pero nos resulta más cómodo y económico agruparlo en una sola línea.

Las asignaciones y el cálculo de la suma son idénticos, pero la impresión tiene una diferencia, además de su traducción *imprimir → printf.*

El printf puede mostrar el contenido de una o más variables y, por lo general, va acompañado de carteles. En nuestro ejemplo: *printf(“La suma es %i”,suma);* el cartel es *“La suma es “ ,* luego *%i* que es una máscara de salida y, por último, *suma* que es la variable.

Cada máscara va asociada a una variable, en estos casos *%i* se asocia a *suma,* esto quiere decir que el formato de salida que debe mostrar *S* debe ser un entero.

Otro ejemplo:

`imprimir (“El área es ”,A,” y el perímetro es “,Peri);` →En estos casos se presentan dos máscaras, una asociada a la variable A que es el

área, y la otra variable Peri que es el perímetro, entonces la codificación será:
`printf("El área es %i y el perímetro es %i",A,Peri);`

Esto quiere decir que tanto A como Peri son de tipo entero
En la siguiente tabla mostramos las máscaras más comunes:

Máscara	Imprime
%i	Un entero
%d	Un entero
%c	Un único carácter
%s	Una cadena de caracteres
%(número)s	Una cadena de caracteres limitada por un número, por ejemplo: %5s (en estos casos imprimirá los cinco primeros caracteres)
%%	% Esto es por si queremos imprimir el símbolo de porcentaje
%(número1).(número2)f	Un número con decimales. El tamaño es número1 y la cantidad de decimales es número2. Por ejemplo: %6.2f (el tamaño del número será 6 y tendrá 2 decimales)

Ejemplo 3: (ejercicio 4 del capítulo anterior). Se ingresa el radio para mostrar el perímetro de la circunferencia.

Pseudocódigo	Código

definir PI 3.1416
comienzo()
{
ingresar (r);
P = 2*PI*r;
imprimir ("El perímetro de la circunferencia es ",P);
}
	#include <stdio.h>
	#define PI 3.1416
	main()
	{
	float r,P;
	printf("Ingrese el radio: ");
	scanf("%f",&r);
	P = 2*PI*r;
	printf("El perímetro de la circunferencia es %6.2f",P);
	}

En este ejemplo se usa una constante *PI*, es decir que la codificación de ***definir PI 3.1416*** es prácticamente directa ***#define PI 3.1416***.

La traducción de la sentencia *ingresar* es *scanf*. Éste tiene asociados dos parámetros: la máscara y la variable de ingreso. Delante de la variable hay un «&» que significa la dirección de la variable, donde se va a almacenar el dato. En el ejemplo el radio r es de tipo *float* (decimal).

> **Nota:** da lo mismo usar la máscara %i o %d.

Hasta el momento hemos codificado algoritmos de estructura secuencial, a continuación una tabla como referencia.

Pseudocódigo	Código	Ejemplo
definir	`#define`	`#define G 9.807`

comienzo()	`main()`	`main()`
		`{`
		`sentencias`
		`}`

ingresar	`scanf`	`scanf("%d",&edad);`
imprimir	`printf`	`printf("El promedio es %6.2",P);`
Asignaciones	Se mantienen	`S = A * B + C;`
Llaves { }	Se mantienen	

> **Atención:** recordar que las variables se declaran en el código siendo invisibles para el pseudocódigo.

Comentarios en la codificación

El lenguaje C, como la mayoría de los otros lenguajes, permite al programador introducir comentarios en el programa fuente, pero que el compilador ignora siempre y cuando estén expresados dentro de los símbolos «/*» en el comienzo y «*/» para el final.

```
/*  Esto es un comentario
            que ocupa varias líneas       */
```

La misión de estos comentarios es servir de explicación o aclaración sobre cómo está desarrollado el programa, de forma que pueda ser entendido por cualquier otra persona o por el propio programador un tiempo después.

Por ejemplo, si quisiéramos comentar el ejemplo 3 en la cabecera:

```
/* Ejemplo 3: Se ingresa el radio para mostrar
el perímetro de la circunferencia */

#include <stdio.h>
#define PI 3.1416
main()
{
float  r,P;
printf("Ingrese el radio:  ");
scanf("%f",&r);
P = 2*PI*r;
printf("El perímetro de la circunferencia es %6.2f",P);
}
```

> **Nota:** el comentario puede ocupar las líneas que sean necesarias siempre y cuando estén encerrados por los símbolos /*...*/

El comentario puede utilizarse en cualquier parte del programa, por ejemplo, en el caso anterior podríamos comentar que falta chequear que el radio que ingresa el usuario por teclado no sea ≤ 0, como se ve a continuación:

```
/* Ejemplo 3: Se ingresa el radio para mostrar
el perímetro de la circunferencia */
#include <stdio.h>
#define PI 3.1416
main()
{
float  r,P;
printf("Ingrese el radio:  ");
scanf("%f",&r);  /* falta validar el ingreso por si se ingresa un
valor cero o negative */
P = 2*PI*r;
printf("El perímetro de la circunferencia es %6.2f",P);
}
```

Nota: el comentario es optativo, pero siempre se recomienda usarlo como un simple recordatorio.

Codificación en estructura de control de decisión

En el capítulo de Pseudocódigo aprendimos cómo funciona la estructura de decisión y empezamos por ver la estructura de control de decisión simple. A continuación, la traducción en código y un ejemplo en pseudocódigo y código.

Pseudocódigo	Código

si (condición)
{
sentencias;
}
	if (condición)
	{
	sentencias;
	}

Ejemplo 4: (ejercicio 5 del capítulo anterior). Mostrar el perímetro de una circunferencia, siempre y cuando el radio que se ingresa sea mayor a cero.

Pseudocódigo	Código

definir PI 3.1416
comienzo()
{
ingresar (r);
si (r > 0)
{
Peri = 2*r*PI;
imprimir (“El perí-
metro de la circunferencia
es ”,Peri);
}
}
	#include <stdio.h>
	#define PI 3.1416
	main()
	{
	float r,Peri;
	printf(“Ingrese el radio: “);
	scanf(“%f”,&r);
	if (r > 0)
	{
	Peri = 2*r*PI;
	printf(“El perímetro de la
	circunferencia es %6.2f“,Peri);
	}
	}

Atención: es importante conservar la indentación también en el código.

A continuación vemos el código de la estructura de control de decisión doble en la traducción en código, y un ejemplo en pseudocódigo y código.

Pseudocódigo	Código

si (condición)
{
sentencias;
}
sino
{
sentencias;
}
	if (condición)
	{
	sentencias;
	}
	else
	{
	sentencias;
	}

Ejemplo 5: (ejercicio 6 del capítulo anterior). Mostrar el perímetro de una circunferencia, siempre y cuando el radio que se ingresa sea mayor a cero, caso contrario, informar con el cartel «Error en el ingreso».

Pseudocódigo	Código

definir PI 3.1416
comienzo()
{
ingresar (r);
si (r > 0)
{
Peri = 2*r*PI;
imprimir (“El perímetro de la circunferencia es”,Peri);
}
sino
imprimir (“Error en el ingreso”);
}
	#include <stdio.h>
	#define PI 3.1416
	main()
	{
	float r,Peri;
	printf(“Ingrese el radio: “);
	scanf(“%f”,&r);
	if (r > 0)
	{
	Peri = 2*r*PI;
	printf(“El perímetro de la circunferencia es %6.2f“, Peri);
	}
	else
	printf(“Error en el ingreso”);
	}

Nota: en estos casos dentro del *else* no tenemos llaves *{}* por tratarse de una única sentencia.

Respecto de los ***si*** anidados:

Pseudocódigo	Código

si (condición)
  si(condición)
  {
  sentencias;
  }
sino
  si(condición)
  {
  sentencias;
  }
	if (condición)
	  if(condición)
	  {
	  sentencias;
	  }
	else
	  if(condición)
	  {
	  sentencias;
	  }

Ejemplo 6: (ejercicio 10 del capítulo anterior). Ingresar dos números enteros, mostrar el más grande y, en caso de que sean iguales, mostrar un cartel «Son iguales».

Pseudocódigo	Código

comienzo()
{
ingresar(A,B);
si (A >= B)
  si (A == B)
  imprimir (“Son iguales”);
  sino
  imprimir (“El número ”,A,”
es el mayor”);
sino
  imprimir (“El número ”,B,”
es el mayor”);
}
	#include <stdio.h>
	main()
	{
	int A,B;
	printf(“Ingrese dos números “);
	scanf(“%i”,&A);
	scanf(“%i”,&B);
	if (A >= B)
	if (A == B)
	  printf(“Son iguales”);
	  else
	  printf (“El número %i es
	el mayor”,A);
	else
	  printf (“El número %i es
	el mayor”,B);
	}

Por su parte, la estructura de control múltiple «según» se traduce al código de la siguiente manera:

Pseudocódigo	Código

según (variable)
{
caso valor1:
sentencias;
salir;
caso valor2:
sentencias;
salir;
.
.
.
defecto:
sentencias;
salir;
}
	switch (variable)
	{
	case valor1:
	sentencias;
	break;
	case valor2:
	sentencias;
	break;
	.
	.
	.
	default:
	sentencias;
	break;
	}

Ejemplo 7: (ejercicio 13 del capítulo anterior). Ingresar dos números *A* y *B* y el operador aritmético *op* (suma "+", resta "-", producto "*" o división "/") para realizar el cálculo y mostrar el resultado.

Pseudocódigo	Código

comienzo()
{
ingresar (A,B);
ingresar (op);
según (op)
{
caso '+':
R = A + B;
salir;
caso '-':
R = A - B;
salir;
caso '*':
R = A * B;
salir;
caso '/':
R = A / B;
salir;
defecto:
R = 0;
salir;
}
imprimir ("El resultado es ", R);
}
	#include <stdio.h>
	main()
	{
	int A,B;
	float R;
	char op;
	printf("Ingrese dos números ");
	scanf("%i",&A);
	scanf("%i",&B);
	printf("Ingrese la operación ");
	scanf("%c",&op);
	switch (op)
	{
	case '+':
	R = A + B;
	break;
	case '-':
	R = A - B;
	break;
	case '*':
	R = A * B;
	break;
	case '/':
	R = A / B;
	break;
	default:
	R = 0;
	break;
	}
	printf("El resultado es %6.2f",
	R);
	}

Nota: en este ejemplo no se contempla la posibilidad de dividir por cero. Una propuesta sería informar tal situación y no cometer el error en calcularlo.

Codificación en estructura de control de repetición

En el caso de la sentencia «para» la traducción a la codificación es «for»:

`para(valor inicial; condición; incremento/decremento)`
`{`
`sentencias;`
`}`
	`for(valor inicial; condición; incremento/decremento)`
	`{`
	`sentencias;`
	`}`

Ejemplo 8: hallar el promedio de altura de los alumnos de una comisión sabiendo que son 30.

Pseudocódigo	Código

`comienzo()`
`{`
`S = 0;`
`para (i=0; i<30; i++)`
`{`
`ingresar (A);`
`S = S + A;`
`}`
`P = S / 30;`
`imprimir (“El promedio de altura es “,P);`
`}`
	`#include <stdio.h>`
	`main()`
	`{`
	`int A,i,S;`
	`float P;`
	`S = 0;`
	`for (i=0; i<30; i++)`
	`{`
	`printf(“Ingrese la altura “);`
	`scanf(“%i”,&A);`
	`S = A + S;`
	`}`
	`P = (float) S / 30;`
	`printf (“El promedio de altura es %6.2f“,P);`
	`}`

Nota: en el ejemplo, cuando queremos obtener el promedio con el cálculo *P = (float) S / 30;* la instrucción (*float*) hace que podamos convertir un número entero a decimal, de lo contrario, se truncaría el número si se toma en cuenta solo la parte entera.

A continuación, veremos cómo se codifica el *bucle* para anidado con el ejemplo 24 propuesto en el capitulo anterior.

Ejemplo 9: (ejercicio 24 del capítulo anterior). Realizar un algoritmo que produzca por pantalla lo siguiente:

```
1   →   100  110  120  130  140  150
2   →   100  110  120  130  140  150
3   →   100  110  120  130  140  150
4   →   100  110  120  130  140  150
5   →   100  110  120  130  140  150
6   →   100  110  120  130  140  150
7   →   100  110  120  130  140  150
8   →   100  110  120  130  140  150
9   →   100  110  120  130  140  150
10  →   100  110  120  130  140  150
```

Pseudocódigo

```
comienzo()
{
para (i=1; i <= 10; i++)
 {
  imprimir(i," ----> ");
    para (j=100; j<=150; j=j+10)
         {
         imprimir(j,"   ");
         }
 imprimir;
 }
}
```

Código

```
#include <stdio.h>
main()
{
int i,j;
for (i=1; i <= 10; i++)
  {
   printf("%i ---->", i);
   for (j=100; j<=150;   j=j+10)
     {
      printf("%i ", j);
     }
   printf("\n");
   }
}
```

Para hallar máximos y mínimos con la estructura de control de repetición *para,* veremos a continuación un ejemplo:

Ejemplo 10: hallar la temperatura máxima registrada en una laguna sabiendo que se toman 40 muestras.

Pseudocódigo	Código

comienzo()
{
Tmax = -10000;
para (i=0; i < 40; i++)
{
ingresar(T);
si (T > Tmax)
Tmax = T;
}
imprimir(“La mayor temperatura registrada es “,Tmax);
}
	#include <stdio.h>
	main()
	{
	int i, T, Tmax;
	Tmax = -10000;
	for (i=1; i <= 40; i++)
	{
	printf(“Ingresar la temperatura ”);
	scanf(“%i”,&T);
	if (T >Tmax)
	Tmax = T;
	}
	printf(“La mayor temperatura registrada es %i”,Tmax);
	}

Ejemplo 11: (ejercicio 26 del capítulo anterior). Hallar la temperatura máxima y la mínima registrada en una laguna sabiendo que se toman 30 muestras.

Pseudocódigo	Código

comienzo()
{
Tmax = -10000;
Tmin = 10000;
para (i=0; i < 30; i++)
{
ingresar(T);
si (T > Tmax)
Tmax = T;
si (T < Tmin)
Tmin = T;
}
Imprimir(“La mayor temperatura registrada es “,Tmax);
Imprimir(“La menor temperatura registrada es “,Tmin);
}
	#include <stdio.h>
	main()
	{
	int i, T, Tmax, Tmin;
	Tmax = -10000;
	Tmin = 10000;
	for (i=1; i <= 30; i++)
	{
	printf(“Ingresar la temperatura ”);
	scanf(“%i”,&T);
	if (T >Tmax)
	Tmax = T;
	if (T <Tmin)
	Tmin = T;
	}
	printf(“La mayor temperatura registrada es %i”,Tmax);
	printf(“La menor temperatura registrada es %i”,Tmin);
	}

En el caso de la estructura de repetición «mientras» la traducción es «while».

Pseudocódigo	Código

mientras (condición)
{
sentencias;
}
	while (condición)
	{
	sentencias;
	}

Ejemplo 12: (ejercicio 28 del capítulo anterior). Calcular la suma de números ingresados por teclado hasta que se ingrese un cero.

Pseudocódigo	Código

comienzo()
{
S = 0;
ingresar(N);
mientras (N != 0)
{
S = S + N;
ingresar(N);
}
imprimir (“La sumatoria es “,S);
}
	#include <stdio.h>
	main()
	{
	int N, S;
	S = 0;
	printf(“Ingresar un número ”);
	scanf(“%i”,&N);
	while (N != 0)
	{
	S = S + N;
	printf(“Ingresar un número ”);
	scanf(“%i”,&N);
	}
	printf(“La sumatoria es %i”, S);
	}

Ejemplo 13: (ejercicio 29 del capítulo anterior). Hallar el promedio de números ingresados por teclado hasta que aparezca uno que no sea par y mayor a 20.

Pseudocódigo	Código

comienzo()
{
S = 0;
C = 0;
ingresar(N);
mientras(!((N%2==1)&&(N>
20)))
{
S = S + N;
C++;
ingresar(N);
}
P = S/C;
imprimir (“El promedio es “,P);
}
	#include <stdio.h>
	main()
	{
	int N, S, C;
	float P;
	S = 0;
	C = 0;
	printf(“Ingresar un número ”);
	scanf(“%i”,&N);
	while(!((N%2==1)&&(N>20)))
	{
	S = S + N;
	C++;
	printf(“Ingresar un número ”);
	scanf(“%i”,&N);
	}
	P = (float) S/C;
	printf(“El promedio es %6.2f”,
	P);
	}

Respecto de la sentencia «hacer - mientras» → *do - while,* en general es:

Pseudocódigo	Código

hacer
{
sentencias;
}
mientras (condición);
	do
	{
	sentencias;
	}
	while (condición);

Ejemplo 14: hacer un algoritmo que muestre del 1 al 21 (solo los números impares).

Pseudocódigo	Código

comienzo()
{
N = 1;
hacer
{
imprimir(N," “);
N = N + 2;
}
mientras (N < = 21);
}
	#include <stdio.h>
	main()
	{
	int N;
	N = 1;
	do
	{
	printf(“%i ",N);
	N = N + 2;
	}
	while (N <= 21);
	}

Ejemplo 15: (ejercicio 32 del capítulo anterior). En un bosque se necesita saber el promedio de diámetro de cada tronco de ciprés y el promedio de su altura. El proceso termina cuando el usuario responde con una 'N', mientras tanto, debe responder con 'S'.

Pseudocódigo	Código

comienzo()
{
C = 0;
SA = 0;
SD = 0;
hacer
  {
  ingresar(A,D);
  SA = SA + A;
  SD = SD + D;
  C++;
  imprimir("¿Desea seguir ingresando datos? S/N ");
  ingresar(opción);
  }
mientras (opción == 'S');
PA = SA/C;
PD = SD/C;
imprimir ("El promedio de altura de los cipreses es ",PA);
imprimir ("El promedio de diámetro de los cipreses es ",PD);
}
	#include <stdio.h>
	main()
	{
	int C;
	float SA, SD, A, D, PA, PD;
	char opción;
	C = 0;
	SA = 0;
	SD = 0;
	do
	  {
	  printf("ingresar altura");
	  scanf("%f",&A);
	  printf("ingresar diámetro");
	  scanf("%f",&D);
	  SA = SA + A;
	  SD = SD + D;
	  C++;
	  printf("¿Desea seguir ingresando datos? S/N ");
	  scanf("%c",&opción);
	  }
	while (opción == 'S');
	PA = SA/C;
	PD = SD/C;
	printf("El promedio de altura de los cipreses es %6.2f",PA);
	printf("El promedio de diámetro de los cipreses es %6.2f",PD);
	}

Nota: en este ejemplo los promedios *PA* y *PD* no requieren de (*float*) para la transformación de entero a decimal, porque las sumatorias *SA* y *SD* fueron declaradas de tipo *float*.

Guía rápida para pasar de *pseudocódigo* a *código* las estructuras de control

Tipo de estructura	Pseudocódigo	Código

Decisión Simple	si (condición)
	{
	sentencias;
	}
		if (condición)
		{
		sentencias;
		}

Decisión Doble	si (condición)
	{
	sentencias;
	}
	sino
	{
	sentencias;
	}
		if (condición)
		{
		sentencias;
		}
		else
		{
		sentencias;
		}

Decisión anidada	si (condición)
	si(condición)
	{
	sentencias;
	}
	sino
	si(condición)
	{
	sentencias;
	}
		if (condición)
		if(condición)
		{
		sentencias;
		}
		else
		if(condición)
		{
		sentencias;
		}

Decisión Múltiple	según (variable)
	{
	caso valor:
	sentencias;
	salir;
	.
	.
	defecto:
	sentencias;
	salir;
	}
		switch (variable)
		{
		case 1:
		sentencias;
		break;
		.
		.
		default:
		sentencias;
		break;
		}

Repetición exacta	para (valor inicial; condición; incremento/decremento)
	{
	sentencias;
	}
		for ((valor inicial; condición; incremento/decremento)
		{
		sentencias;
		}

Repetición condicional (puede no iterar)	mientras (condición);
	{
	sentencias;
	}
		while (condición);
		{
		sentencias;
		}

Repetición condicional (itera al menos una vez)	hacer
	{
	sentencias;
	}
	mientras (condición);
		do
		{
		sentencias;
		}
		while (condición);

La función getchar()

En lenguaje C, cuando utilizamos variables de tipo *char* y lo hacemos para ingresar por pantalla dentro de un bucle, queda en el *buffer* de memoria pulsado el ENTER. Entonces el programa lo asume como un carácter de ingreso salteando el próximo *scanf*.

Una solución posible es escribir en el código la función getchar() posterior al ingreso de un carácter. A continuación un ejemplo:

Ejemplo 16: se desea saber cuántas mujeres hay entre las 10 personas.

```
#include<stdio.h>
main()
{
char s;int i,cm=0;
for (i=0; i<10;i++)
     {
     printf(“Ingrese sexo= “);
     scanf(“%c”,&s);
     while(getchar()!=’\n’);
    if(s==’m’)
            cm++;
     }
printf(“\nLa cantidad de mujeres es %i”,cm);
}
```

Si no funciona podemos utilizar la función *setbuf*. La solución en estos casos es la de limpiar el *buffer* con la instrucción **setbuf(stdin,NULL).** De este modo, rellenamos el *buffer* con nada, es decir que lo vaciamos.

A continuación, el ejemplo anterior, pero con otra alternativa.

```
#include<stdio.h>
main()
 {
char s;int i,cm=0;
for (i=0; i<10;i++)
     {
     setbuf(stdin,NULL);
     printf(“Ingrese sexo= “);
     scanf(“%c”,&s);
     if(s==’m’)
            cm++;
     }
printf(“\nLa cantidad de mujeres es %i”,cm);
}
```

Atención: Otra estrategia recomendable para solucionar el problema de modo muy sencillo es utilizando un truco en la función scanf. Cada vez que ingresamos un dato de tipo char por pantalla dejamos un espacio en

blanco entre las comillas y el %, de la siguiente forma: scanf(“ %c”,&s); sin necesidad de agregar otras instrucciones.

Librería <math.h>

Al comienzo del capítulo de codificación mencionamos algunas librerías como: stdio, stdlib, string, time y math. Este último en particular sirve para realizar operaciones aritméticas.

Ejemplo 17: si queremos hallar el coseno de un ángulo, debemos incluir la librería math.h para poder efectuarlo.

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159265
main ()
{
 double parametro, resultado;
 parametro = 60.0;
 resultado = cos ( parametro * PI / 180.0 );
 printf (“El coseno de %6.2f grados es %6.2f.\n”, parametro,
resultado);
}
```

De esta manera su salida será: «**El coseno de 60.00 grados es 0.50**».

A continuación, mostramos una tabla que contiene algunas de las funciones de la librería math.h:

Función	Tipo de dato	Propósito
sin(x)	double	Devuelve el seno de x
cos(x)	double	Devuelve el coseno de x
tan(x)	double	Devuelve la tangente de x
asin(x)	double	Devuelve el arco seno de x
acos(x)	double	Devuelve el arco coseno de x
atan(x)	double	Devuelve el arco tangente de x
exp(x)	double	Eleva e a la potencia x (e = 2.77182…)

log(x)	double	Devuelve el logaritmo natural de x
log10(x)	double	Devuelve el logaritmo en base 10
pow(x,y)	double	Devuelve x^y
sqrt(x)	double	Devuelve la raíz cuadrada de x
floor(x)	double	Devuelve un valor redondeado al entero más cercano a x
fabs(x)	double	Devuelve el valor absoluto de x

Ejercicios: pasar de pseudocódigo a código

1. Codifique los ejemplos del capítulo anterior (pseudocódigo): 1, 2, 3, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27, 30 y 31.
2. Codifique los ejercicios del capítulo anterior (pseudocódigo): 6,7, 11, 14, 25, 26, 27, 29, 30, 31, 35, 42, 44, 46, 48, 52, 53, 55, 57 y 62.
3. Crear un algoritmo para ingresar 10 pares de números Naturales que representan las coordenadas de un tablero de 10x10 cm. Los números son enteros y representan (x ; y) por ejemplo: (2;7) , (5;1),...
   Se pide:
   a. Hallar y mostrar el promedio de todas las x.
   b. Hallar y mostrar la cantidad de y impares.
   c. Hallar y mostrar la cantidad de pares donde su producto se encuentre en el siguiente rango (8; 20]

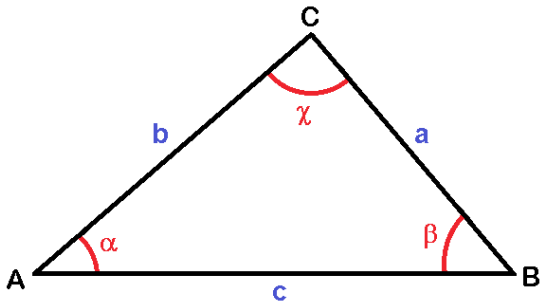
   Ejemplos: (3;2) → No se encuentra por ser 6 su producto , (4;5) → Si se encuentra por ser 20 su producto, (8;2) → Si se encuentra por ser 16 su producto
   d. Hacer y mostrar la prueba escritorio para los siguientes datos: (9;8) , (1;9) , (3;10) , (6;4) , (3;3) , (2;5) , (7;4) , (1;9) , (5;5) ,(3;8)
   e. ¿Qué cambiaría del algoritmo si no sabemos cuántos pares de números son en total? Crear en todo caso otro algoritmo.

Ejercicios: Para usar librería <math.h>

4. Realice en código una tabla con cuatro columnas que muestre los números reales del 0 a π (de 0.1 en 0.1), el seno, la tangente y el coseno correspondientes.
5. Igual que con el anterior pero con arco tangente, arco coseno y arco seno.
6. Halle las raíces del siguiente polinomio de segundo grado: $P(x)=2x^2-x-6$

7. Halle las raíces de un polinomio de grado 2 ingresado por teclado (solo los coeficientes). Contemple si se trata de un número complejo donde la raíz es negativa, anúncielo con un cartel de tal situación.
8. Halle la hipotenusa de un triángulo rectángulo ingresando por teclado sus catetos.
9. Halle el perímetro y el área de un triángulo rectángulo ingresando por teclado la hipotenusa y uno de sus catetos.
10. Usando el teorema del coseno: $c^2=a^2+b^2-2ab.\cos(\gamma)$

    a. Halle *c* cuando se ingresan los siguientes datos: *a*, *b*, γ
    b. Halle *a* cuando se ingresan los siguientes datos: *b*, *c*, γ
    c. Halle γ cuando se ingresan los siguientes datos: *a*, *b*, *c*
    d. Halle *b* cuando se ingresan los siguientes datos α, β, *a*, *c*



11. Hallar la solución del siguiente polinomio: $P(x)=x^4+8x^2-3y^2-2$
    Ingresando por teclado los valores de las variables *x* e *y*

Nota: agregar comentario como título del ejercicio.

Capítulo 4: Arreglos, vectores o matrices

La palabra vector, del latín *vectoris*, derivada del verbo *veho*, cuyo significado es «el que transporta o conduce», es utilizada en diferentes ámbitos.

Por ejemplo, en la Física se utiliza para magnitudes escalares; en Biología, como un agente orgánico; en Geometría significa segmentos rectadirigidos; pero en Informática se llama vector a una zona de almacenamiento continuo que alberga un conjunto de datos del mismo tipo ubicados cada uno en una posición determinada.

Dichos datos están ordenados en filas y columnas, por eso muchas veces los vectores también son mencionados como Matrices.

Para generalizar y facilitar el concepto, los mencionaremos en el libro como *arreglos*.

La traducción de arreglos al inglés es *arrays*.

Cada dato que forma parte del arreglo se denomina *elemento*.

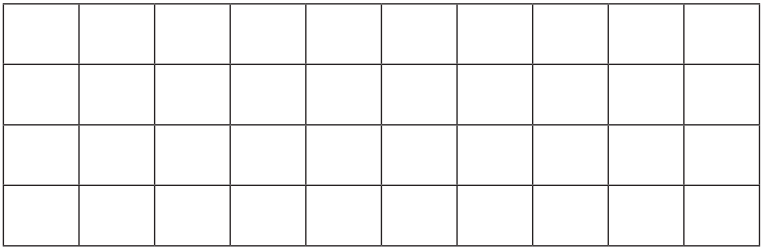
Los arreglos ofrecen una estructura que facilita el acceso a los elementos.

Un arreglo puede ser unidimensional, solo tiene una fila y columnas, llamados vectores:

En este caso tenemos una fila y 10 columnas.

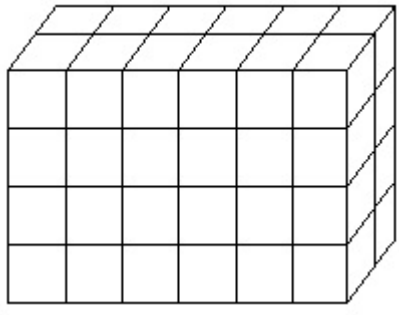
Los arreglos multidimensionales son aquellos que tienen más de una dimensión. En particular, los bidimensionales son también llamados Matrices.

Bidimensional, cuando tienen filas y columnas:



En este caso tenemos 4 filas y 10 columnas.

Tridimensional:



En este caso tenemos 4 filas, 6 columnas, y 2 de profundidad.
Los casos que veremos en el libro son los bidimensionales y los unidimensionales.

Vectores

Como pudo observarse en las gráficas anteriores, los arreglos tienen un número limitado de filas y columnas. En algunos lenguajes estas dimensiones pueden ser cambiadas (redimensionadas) en el algoritmo, pero en el lenguaje C deben ser declaradas de entrada.

Se declaran como las demás variables, pero solo agregando el máximo número de elementos entre corchetes luego del nombre, por ejemplo int miarreglo[10];

Esto quiere decir que el arreglo de nombre «miarreglo» puede tener como máximo 10 números enteros.

Las posiciones del arreglo son llamadas subíndices y siempre empiezan en cero.

Por ejemplo, si tuviéramos en el arreglo los siguientes datos:

4	-3	1	1	0	7	20	-1	10	7

En la posición cero [0] se ubica el número 4, en la [1] el -3 y así hasta llegar a la posición [9] con el número 7. Recuerden que los subíndices comienzan en cero por ese motivo se le resta siempre uno.

Con el ejemplo de la gráfica, si queremos cambiar los números 7 por el 6 usando código, entonces será de la siguiente forma:

```
miarreglo[5] = 6;
miarreglo[9] = 6;
```

Los números 7 estan ubicados en las posiciones 5 y 9.
Si queremos sumar los dos primeros y los dos últimos elementos del arreglo:

```
sum = miarreglo[0]+ miarreglo[1]+ miarreglo[8]+ miarreglo[9];
```

En cambio, si queremos multiplicar por 4 el elemento ubicado en la cuarta posición y a éste restarle la suma entre el primero y el último:

```
sum = 4*miarreglo[3]- (miarreglo[0]+ miarreglo[9]);
```

Si queremos imprimir un elemento en particular del vector hacemos:

`printf("%i",miarreglo[6]);`→ Va a imprimir el elemento que se encuentra en la posición 7.

Cuando queremos recorrer el arreglo unidimensional ya sea para ingresar, mostrar y/o procesar sus datos, el recorrido es siempre secuencial, entonces utilizamos el ciclo de repetición *for*.

> **Atención:** en el área de las Matemáticas y Métodos Numéricos las matrices y vectores se escriben en mayúsculas y los elementos que la componen, en minúsculas.

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0n} \\ a_{10} & a_{11} & a_{12} & \dots & a_{1n} \\ . & . & . & \dots & . \\ a_{m0} & a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Pero para respetar las convenciones del lenguaje C, seguimos con que las variables sean siempre escritas en minúsculas y las constantes que definen las dimensiones de los arreglos, en mayúsculas.

A continuación mostramos algunos ejemplos sencillos en código C.

Ejemplo 1: cómo agregar 20 números enteros por teclado a un arreglo de nombre «arreglo».

```
#include<stdio.h>
#define N 20
main()
{
int arreglo[N] , i;
for (i=0 ; i<N ; i++)
     {
     printf("\n Ingrese un número: ");
     scanf("%d",&arreglo[i]);
     }
}
```

Ejemplo 2: cómo mostrar por pantalla los 20 números enteros que fueron ingresados en el ejemplo anterior.

```
…
for (i=0 ; i<N ; i++)
     {
     printf("\n  En  la  posición  %i  se  cargó  el  número
%i",i,arreglo[i]);
     }
```

Ejemplo 3: la sumatoria de todos los números del arreglo.

```
…
int Sum=0,i;
for (i=0 ; i<N ; i++)
{
     Sum=Sum+arreglo[i];
}
printf("La sumatoria es %i,Sum");
```

Nota: se puede hacer declaración y asignación al mismo tiempo. En el ejemplo se observa que la variable *Sum* es declarada de tipo int y se le asigna cero por ser una variable sumadora.

```
Es decir que: int Sum;      es equivalente a:      int Sum = 0;
              Sum = 0;
```

Ejemplo 4: la cantidad de números negativos del arreglo.

```
…
int Cant=0,i;
for (i=0 ; i<N ; i++)
    {
     if(arreglo[i]<0)
    {
     Cant++;
     }
}
```

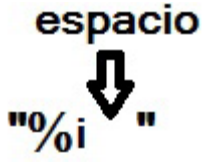
Ejemplo 5: se ingresan al arreglo los 10 primeros números naturales y se los muestra.

```
#include<stdio.h>
#define N 10
main()
{
int arreglo[N] , i;
for (i=0 ; i<N ; i++)
    {
     arreglo[i]=i;
     }
printf("\n El vector resultante es:\n");
for (i=0 ; i<N ; i++)
     {
     printf(" %i  ",arreglo[i]);
     }
}
```

El resultado por pantalla es el siguiente:

El vector resultante es:
0 1 2 3 4 5 6 7 8 9

Es conveniente mostrar los elementos del vector en forma horizontal para una mejor visión, como en el caso anterior. Es importante tomar en cuenta la separación entre elementos con un espacio en blanco en la máscara del print.



Aunque lo más conveniente para separar elementos es utilizar en la máscara lo siguiente:

```
printf("%2i",arreglo[i]); → en estos casos la separación será
                                de 2 lugares
```

Independientemente de la cantidad de dígitos que tenga el elemento, siempre tendremos la separación en dos espacios.

> **Atención:** cuando declaramos un arreglo automáticamente estamos reservando espacio de memoria RAM (memoria central) para luego ser llenado pero, mientras tanto, se ha generado «basura» (números sin sentido).

Probar definir un arreglo y mostrarlo como en el ejemplo 2; se verá por pantalla la «basura».

Si queremos evitar la basura podemos llenar el arreglo en la misma declaración, por ejemplo:

`int arreglo[N]={0}` ; En estos casos todos los elementos del arreglo son ceros. Esto es muy útil cuando tenemos que usar arreglos como contadores o sumadores.

Ejemplo 6: hallar la frecuencia de números de un dado, sabiendo que fueron 100 los tiros.

```
#include<stdio.h>
#define N 100
main()
{
```

```
   int frecuencia[6]={0},i, r;→se inicializa el arreglo
                              «frecuencia» con ceros
   srand(time(NULL));
   for (i=0 ; i<N ; i++)
   {
     r=rand()%6+1;→se obtiene un número aleatorio del 1 al 6
        switch(r)
                {
                case 1:
                  frecuencia[0]++;→cuenta la cantidad de «unos»
                                   en el subíndice 0
                  break;
                case 2:
                  frecuencia[1]++; →cuenta la cantidad de «dos»
                                    en el subíndice 1

                  break;
                case 3:
                   frecuencia[2]++;→cuenta la cantidad de «tres»
                                   en el subíndice 2
                  break;
                case 4:
                  frecuencia[3]++;→cuenta la cantidad de
                                   «cuatros» en el subíndice 3
                  break;
                case 5:
                  frecuencia[4]++;→cuenta la cantidad de
                                   «cincos» en el subíndice 4
                  break;
                case 6:
                  frecuencia[5]++;→cuenta la cantidad de «seis»
                                   en el subíndice 5
                  break;
                }
        }
   printf("\n  La  frecuencia  de  cada  número  del  dado  es  la
siguiente:\n");
   for (i=0 ; i<6 ; i++)
                {
```

```
    printf("\n El número  %i salió %i veces",i+1,frecuencia[i]);
→sumamos un 1 al subíndice i por el corrimiento (porque no existe la
cara cero de un dado)
    }
   }
```

> **Atención:** en este caso nos aseguramos de no mostrar la «basura» cuando un número del dado nunca sale (en dicho caso muestra un cero).

Con arreglos, podemos de entrada definir su contenido. Ejemplos:

```
char vocales[]={'a','e','i','o','u'};→los caracteres deben estar
                                      entre apóstrofes.
int  ruleta[]={0,1,2,3,4,…,35,36};
float  x[]={-10,-9.5,-9,-8.5,8,…,8,8.5,9,9.5,10};
int binario[]={0,1};
```

Como se puede apreciar en cada ejemplo, entre corchetes no hay máximos porque lo establece la misma cantidad de elementos que figuran entre las llaves.

Función rand()

Se trata de una función muy útil para generar números aleatorios.

Por lo general se usa cuando queremos simular el azar. En el ejemplo anterior, el reparto de naipes, el juego de la ruleta, la lotería o el bingo, cara o ceca de una moneda, etcétera.

La función *rand()* en el lenguaje C resuelve todo esto, pero no es útil para aplicaciones en las que se requiere por ejemplo que la secuencia de números aleatorios no se repita hasta pasada una cierta cantidad de veces, o para simular tiempos aleatorios con horas, minutos y segundos (en este último caso será necesario reprogramar).

Esta función, cada vez que la llamamos, nos devuelve un número entero aleatorio entre 0 y el RAND_MAX (número grande).

Según el caso, necesitaremos un rango de números, por ejemplo si se trata de obtener números al azar del 0 al 10 entonces usaremos:

```
n_azar = rand() % 11;
```

Sabemos que la operación % nos devuelve el resto de dividir *rand()* con 11. Este resto puede ir de 0 a 10. Si en cambio queremos obtener números del 0 al 100 tendremos que utilizar:

```
n_azar = rand() % 101;
```

Es decir que siempre será N+1. **n_azar = rand() % (N+1);** para obtener un rango de 0 a N.

Pero para el caso de la obtención al azar de las caras de un dado, su rango será 1..6 descartando el cero. La solución es muy simple, se le suma un 1.

```
n_azar = rand() % 6+1;
```

Pasamos a explicar: con solo *rand()%6* se obtienen números al azar del 0 al 5, pero sumando uno se desplaza el rango obteniendo números del 1 al 6.

Si queremos obtener números del 10 al 20 entonces escribiremos:

```
n_azar = rand () % 11 + 10;
```

En general se obtiene de la siguiente manera: **n_azar = rand () % (M-N+1) + N;**

Siendo M el límite superior del intervalo y N, el inferior.

En el ejemplo anterior será: *N* igual a 10 y *M* a 20.

Ejemplos:

Para la ruleta sabemos que el rango va de 0 a 36 entonces será:

```
n_azar = rand () % 37;
```

Y para el juego de bingo el rango va de 1 a 90 entonces será:

```
n_azar = rand () % 90+1;
```

En el caso de que queramos obtener números decimales, por ejemplo de un dígito decimal entre 0.0 y 10.0, lo hacemos de la siguiente forma:

```
n_azar = (float) ((rand () % 101) / 10);
```

Entonces sabemos que con rand () % 101 obtenemos un rango de 0 al 100, pero al dividirlo por 10 se corre una coma a la izquierda y se genera un dígito decimal. Por ejemplo si se obtiene el número 65, éste pasará a ser el 6.5.

Si quisiéramos simular temperaturas al azar con dos dígitos entre -10.00 y 10.00 entonces:

```
n_azar = (float) ((rand () % 2001) / 100 - 10);
```

En estos casos con rand () % 2001 se obtienen números que van del 0 al 2000, pero dividiendo por 100 se obtienen números del 0.00 al 20.00 (dos dígitos decimales). Si se resta por 10, se obtiene un rango de -10.00 al 10.00.

> **Atención:** cada vez que usemos esta función debemos inicializar el generador de números aleatorios con *srand(time(NULL));* siempre al comienzo del algoritmo y por única vez.

Se recomienda que se encuentre fuera de un bucle. En el ejemplo 6 se observa dónde se ubica correctamente (junto a la declaración de variables).

Vectores paralelos

Como se indicó anteriormente, los vectores están sujetos a un único tipo de elemento. No podemos mezclar en un vector elementos de tipo char carácter con int enteros por ejemplo; entonces recurrimos al uso de vectores paralelos.

Los vectores paralelos permiten almacenar la información en forma de registros, de modo tal que dos o más vectores en la misma posición representan un registro y cada campo es una posición del vector.

Por ejemplo, si queremos representar los datos (legajo, edad y carrera) de cada alumno de la universidad debemos utilizar 3 vectores.

0	1	2	3	...	N-2	N-1
1992	3450	2003	8564	...	6123	8890
19	20	18	25	...	22	20
A	E	E	T	...	E	A

El primer vector de tipo entero representa el número de legajo; el segundo, también de tipo entero, la edad; y el tercero, de tipo carácter, la carrera (A → Ambiental, E → Electrónica y T → Telecomunicaciones).

Los que aparecen en el encabezado son los subíndices y están relacionados con los datos de cada alumno, es decir el alumno n.º 2 tiene legajo 2003, su edad es 18 y estudia Ingeniería Electrónica.

La cantidad de alumnos es N.

Ejemplo 7: el siguiente es el algoritmo que representa el ejemplo dado ingresando y mostrando todos sus datos:

```
#include<stdio.h>
#define N 200
main()
{
int legajo[N] ,edad[N] , i;
char carrera[N];
for (i=0 ; i<N ; i++)
     {
     printf("\n Ingrese el número de legajo : ");
     scanf("%d",&legajo[i]);
     printf("\n Ingrese la edad: ");
     scanf("%d",&edad[i]);
     printf("\n Ingrese la carrera: ");
     scanf(" %c",&carrera[i]);
     }
printf("\n La lista de los datos de los alumnos es la siguiente:\n");
for (i=0 ; i<N ; i++)
          printf("\n El legajo  %i,  la edad  %i y la carrera %c"
,legajo[i],edad[i],carrera[i]);
}
```

> **Nota:** Recuerden que es necesario usar la función getchar() , setbuf o simplemente scanf(" %c",&carrera[i]); →con el espacio entre " y % por tratarse de un ingreso de tipo alfanumérico.

Ejemplo 8: tomando en cuenta el ejemplo anterior, si quisiéramos mostrar solo los datos de un alumno ingresando el número de legajo.

```
…
int numleg;
…
printf(“\n Ingrese el número de legajo: ”);
scanf(“%d”,&numleg);
for (i=0 ; i<N ; i++)
     if(numleg==legajo[i])
                    printf(“\n  La  edad   es  %i  y  la  carrera
%c“,edad[i],carrera[i]);
}
```

Máximos y mínimos con vectores

Hallar máximos y mínimos con vectores es muy sencillo y similar a cuando usábamos variables primitivas.

Acá también utilizamos una variable para ir comparando cada elemento del vector.

Ejemplo 9: para hallar el máximo valor de un vector de N números aleatorios.

```
#include<stdio.h>
#define N 100
main()
{
int números[N] , i , posmax ,max=-100 ; →a la variable max se le
                                          asigna un valor absurdo
srand(time(NULL));
for (i=0 ; i<N ; i++)
     {
     números[i]= rand()%10;→se obtiene un número aleatorio
                           del 0 al 9
     }
for (i=0 ; i<N ; i++)
      if(max<números[i])
     {
      max=números[i];
       posmax=i; →La variable posmax reserva la ubicación del
                  vector donde se encuentra el valor máximo
      }
```

```
   printf("\n El valor máximo es %i y se encuentra en la posición
%i", max, posmax);
   }
```

Ejemplo 10: volviendo al ejemplo 7, si quisiéramos hallar el alumno de mayor edad y además mostrar el legajo y su carrera ¿cómo lo implementamos?

```
   …
   int posmax, edadmax= -10 ;→a la variable edadmax se le
                                asigna un valor absurdo
   …
   for (i=0 ; i<N ; i++)
        if(edadmax<edad[i])
        {
         edadmax=edad[i];
        posmax=i;→La variable posmax reserva la ubicación del
                   vector donde se encuentra la edad máxima
        }
    printf("\n El alumno de mayor edad de legajo %i tiene %i años de
edad y su carrera es %c",legajo[posmax], edadmax, carrera[posmax]);
   }
```

La variable *posmax* reserva el índice donde se encuentra la edad máxima, como tenemos vectores paralelos entonces *legajo[posmax]* y *carrera[posmax]* son los datos del alumno que posee la mayor edad.

> **Atención:** si tenemos más de un valor máximo, los ejemplos expuestos anteriormente sirven para hallar el primer máximo que se encuentra, más adelante mostraremos un ejemplo que tome en cuenta todos los máximos (casos de empate).

Para hallar el mínimo se resuelve cambiando la desigualdad "<" por ">" y la variable *edadmax* se llamaría *edadmin* y su le asignaría de entrada un valor absurdo grande.

Ordenamiento en vectores

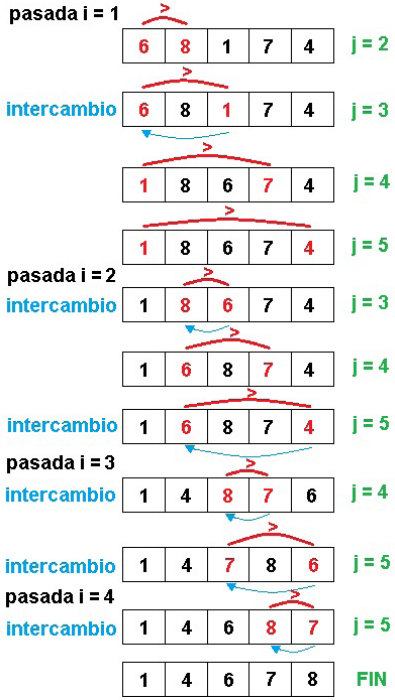
Existen muchos métodos para ordenar los elementos que contiene un vector, el más clásico, fácil de aprender y utilizar se llama burbuja (*Bubble Sort*), también conocido como *intercambio directo*.

Se basa en la comparación e intercambio de elementos adyacentes del vector. De este modo, se dice que los elementos más chicos en su acción burbujean hacia la parte superior de la lista (primer elemento), mientras los valores más grandes se ubican al fondo, esto sucede si su ordenamiento es creciente, es decir, de menor a mayor.

Precisamente este método es lento pero simple, existen otros como: burbuja mejorado (refinado), ordenación por selección, ordenación por inserción, ordenación por casilleros, ordenación por cuentas, ordenación por mezcla, ordenación con árbol binario, ordenación Radix, ordenación por montículos, Quicksort (ordenamiento rápido) y ordenación Shell.

El algoritmo del método burbuja de ordenamiento consiste en realizar varias pasadas sobre el arreglo y lograr que en cada pasada el elemento de mayor valor se coloque al final. Por cada pasada es necesario recorrer el vector realizando comparaciones e intercambios. Por eso, suele implementarse con dos bucles, uno anidado dentro del otro. El bucle exterior realiza las pasadas y el interior recorre el arreglo realizando comparaciones e intercambios.

A continuación vemos un ejemplo sencillo donde se quiere ordenar de modo ascendente (de menor a mayor) 5 elementos (los números enteros 6, 8, 1, 7, 4).



Para cada pasada *i* se efectúa un bucle *j* para preguntar si el elemento de la izquierda es menor ">" al de la derecha (línea roja – ubicada arriba del vector). Si se cumple la desigualdad, entonces se produce el intercambio (línea azul – ubicada debajo del vector).

Para hacer el intercambio no se puede hacer de modo directo, es decir:

```
VEC[i]=VEC[j];
VEC[j]=VEC[i];
```

Porque, finalmente, VEC[j] va a recibir su propio valor (su valor original), entonces para solucionarlo se utiliza una variable auxiliar que sea del mismo tipo de datos que los elementos del vector, de la siguiente manera:

```
AUX= VEC[i];
VEC[i]=VEC[j];
VEC[j]=AUX;
```

> **Nota:** se puede utilizar cualquier nombre para la variable auxiliar.

Ejemplo 11: el algoritmo en lenguaje C para este ejemplo, donde llamamos al vector VEC, es el siguiente:

```
#include<stdio.h>
#define N 5
main()
{
int VEC[]={6,8,1,7,4}, i, j, AUX;
for (i=0 ; i<N-1 ; i++)
     for (j=i+1 ; j<N ; j++)
             if(VEC[i]>VEC[j])
                     {
                     AUX= VEC[i];
                     VEC[i]=VEC[j];
                     VEC[j]=AUX;
                     }
}
```

Atención: para el caso en que necesitemos ordenar de modo descendente, solo debemos cambiar en la condición del *if* el símbolo a "<".

Así como se ordenan vectores con elementos numéricos (*int y float*) también pueden ordenarse caracteres (*char*). A continuación, un ejemplo al respecto:

Ejemplo 12: ordenar de modo descendente el vector VC que contiene 10 letras.

```
#include<stdio.h>
#define N 10
main()
{
char VC[]={'a','n','r','i','t','o','n','g','i','m'}, AUX;
int  i, j;
for (i=0 ; i<N-1 ; i++)
     for (j=i+1 ; j<N ; j++)
             if(VC[i]<VC[j])
             {
             AUX= VC[i];
             VC[i]=VC[j];
             VC[j]=AUX;
             }
printf("La cadena de caracteres resulta ser: ");
for (i=0 ; i<N ; i++)
     printf("%c - ",VC[i]); →Recordar que %c en el printf
                              es para imprimir caracteres
}
```

Ordenamiento con vectores paralelos

Cuando tenemos más de un vector asociado, es decir paralelos, en el proceso de ordenamiento debemos realizar el intercambio de elementos según el vector «clave». A medida que se va efectuando el intercambio de posición según el vector que se desea ordenar (vector clave), los restantes vectores también deben sufrir el mismo cambio, de lo contrario se perdería la relación inicial.

Por ejemplo, si tenemos en un vector los números de legajos y en otro los sueldos netos y deseamos ordenar por legajo (vector clave) los sueldos deben acompañar al asalariado, de lo contrario «beneficiaremos a unos y perjudicaremos a otros».

Ejemplo 13: tenemos dos vectores (VL, VS), el primero contiene los números de legajo de 7 empleados y el segundo, sus respectivos sueldos. Ordenar de modo ascendente los legajos y mostrar en dos columnas la lista.

```
#include<stdio.h>
#define N 7
main()
{
int VL[]={69,65,99,30,39,211,91}, i, j, AUXL;
float   VS[]={12000.00, 15550.50, 20480.80, 19000.75, 19000.75,
7600.00, 8000.95}, AUXS;
for (i=0 ; i<N-1 ; i++)
     for (j=i+1 ; j<N ; j++)
            if(VL[i]>VL[j])
            {
            AUXL= VL[i];
            VL[i]=VL[j];
            VL[j]=AUXL;
            AUXS= VS[i];
            VS[i]=VS[j];
            VS[j]=AUXS;
            }
printf("Lista de empleados:\n ");
printf("LEGAJO      SUELDO \n ");
for (i=0 ; i<N ; i++)
     printf("%i → %7.2f\n",VL[i],VS[i]);
}
```

En pantalla se mostrará de la siguiente manera:

Lista de empleados:
LEGAJO SUELDO
30 → 9.000,75
39 → 19.000,75
65 → 15.550,50
69 → 12.000,00
91 → 8.000,95
99 → 20.480,80
211 → 7.600,00

Nota: por cada vector paralelo debemos utilizar una variable auxiliar del mismo tipo que el vector correspondiente.

Ejercicios: Vectores

1. Escriba un programa que genere un vector de 100 números enteros al azar y lo muestre. Calcule e imprima su suma. Los números deben oscilar entre el 0 y el 9.
2. Escriba un programa que genere un vector de 20 enteros al azar, lo muestre y calcule e imprima su promedio. Los números deben oscilar entre el 5 y el 10.
3. Escriba un programa que genere un vector de 20 números decimales al azar y lo muestre. Además, debe calcular e imprimir su promedio (solo los números que se ubican en subíndices impares). Los números deben oscilar entre el -5.0 y el 5.0
4. Escriba un programa que genere 2 vectores con 20 enteros al azar cada uno (del 1 al 9) y cree un tercer vector que guarde las suma de ambos (subíndice por subíndice).
5. Escriba un programa que genere un vector de 30 números al azar (del 1 al 10), lo muestre y cuente la cantidad de números pares e impares. Mostrar los resultados.
6. Escriba un programa que genere 2 vectores con 20 enteros al azar cada uno (del 1 al 10) y cree un tercer vector de 40 elementos que sea el resultado de intercalar los dos vectores anteriores (subíndice por subíndice). Mostrar todos los vectores.
7. Escriba un programa que genere un vector de nombre «todos» con 30 enteros al azar cada uno (del 1 al 10) y crear dos vectores llamados «par» e «impar» para guardar en el primero todos los números pares del vector «todos» y en el segundo, los impares. Mostrar todos los vectores. No debe mostrar «basura».
8. Escriba un programa que genere un vector de 30 vocales al azar, lo muestre y cuente la cantidad de vocales «a» y «o». Mostrar resultado.
9. Escriba un programa que genere un vector de 30 vocales al azar, lo muestre, luego lo ordene de modo ascendente y vuelva a mostrarlo. Por último, cuente la cantidad de vocales «a», «e», «i», «o», «u» usando un vector contador cantvocales[5]={0}, donde la posición 0 esté relacionada con la cantidad de vocales «a», la posición 1, con la «e», etcétera.

10. Ídem al ejercicio anterior, pero contando los números de la cara de un dado.
11. Cree dos vectores paralelos de 30 elementos. Uno debe contener vocales y el otro números del 1 al 10. Ambos generados al azar. Luego mostrar solo las vocales donde su vector paralelo tenga números impares.
12. Escriba un programa que genere un vector de 30 enteros al azar (del -10 al 10). Calcule e imprima su mínimo y máximo.
13. Ídem al anterior pero utilizando el método *burbuja de ordenamiento*.
14. Escriba un programa que genere un vector de 10 números al azar (del 1 al 9) y permita ingresar por teclado un escalar para efectuar el producto con el vector. El vector resultante deberá ser uno nuevo y mostrarlo.
15. Cree dos vectores paralelos de 30 elementos cada uno. Uno debe contener vocales y el otro números del -10 al 10. Ambos generados al azar. Hallar el mínimo y máximo en el vector de números, pero mostrando además las vocales correspondiente al vector paralelo.
16. Cree dos vectores paralelos de 12 elementos c/u. Uno llamado «vocales», que debe contener vocales, y el otro llamado «números» que contenga enteros del 1 al 10. Ambos generados al azar. Hallar la sumatoria del vector «números» siempre y cuando su paralelo «vocales» contenga la letra «e».
17. Escriba un programa que inicialice un vector de 100 elementos con los números de un dado (al azar) e indique la frecuencia de cada número. Se recomienda usar un vector contador cantdado[6]={0}, donde la posición 0 esté relacionada con la cantidad de unos del dado, la posición 1, con las caras «dos», etcétera.
18. Repita el ejercicio anterior pero ahora informe el número que salió más veces y el número que salió menos veces.

Nota: se recomienda ordenar el vector *cantdado* que contiene la frecuencia de cada cara del dado.

19. Repita el ejercicio anterior pero ahora los números deben ingresarse por teclado (solo 15 números).
20. Escriba un programa que genere números al azar del 1 al 7 en un vector de 30 elementos. Éstos representan los números de la semana, en la medida en que van apareciendo los números, deberá agregar a otro vector de tipo char las letras correspondientes al día de semana. Por ejemplo: 1 es D, 2 es L, 3 es M, 4 es I, 5 es J, 6 es V y 7 es S. Mostrar ambos vectores.

21. Tome en cuenta el ejercicio anterior y agregue lo siguiente: mostrar las veces que aparecieron por cada día de la semana. Usar un vector contador.
22. Use dos vectores que contengan x e y y muéstrelos. Tener en cuenta la función cuadrática $\int(x) = x^2 - 3x + \frac{2}{5}$ con intervalos de 0,5. El límite inferior es -5 y el superior, 5 (usar dos decimales), tanto x como y deben estar almacenadas en los vectores vx y vy, respectivamente.
23. Ídem al anterior pero usando la siguiente función: $\int(x) = (2\text{sen}(x) - \cos(x))^2$. La librería correspondiente es math.h, al tratarse de una función trigonométrica.
24. En una competencia de maratón se necesita registrar tres datos de 200 competidores (número de corredor, peso y ubicación). Ingresar los datos en tres vectores paralelos (vn, vp y vu, respectivamente). Luego se pide lo siguiente:
    a. Ordene de modo ascendente por ubicación.
    b. Muestre ubicación y número de competidor (solo los 10 primeros ordenados).
    c. Halle y muestre el promedio de peso de todos.

25. Usando los datos de los tres vectores del ejercicio anterior se pide lo siguiente:
    a. Ordene de modo descendente por peso.
    b. Muestre solo los números de competidores y su peso (ordenado).
    c. Halle y muestre la cantidad de corredores que pesan entre 76 y 80 kg.

Cadena de caracteres

Cadena de caracteres, palabra o frase (en inglés, *strings*) es una secuencia ordenada de caracteres conformada por letras, números y/o símbolos. En el lenguaje C se trata de casos especiales de vectores de tipo *char* (carácter).

La particularidad es que siempre culminan con un '\0' que representa la marca del fin de la cadena.

Veamos algunos ejemplos de declaración:

```
   char cadena1[]="a"; →equivale a → char cadena1[]={'a','\0'} ;
(ambas contienen 2 elementos)
   char cadena2[]="Argentina"; →equivale a → char
   cadena2[]={'A','r','g','e','n','t','i','n','a','\0'} ;
```

Necesariamente, la cadena de caracteres debe tener la marca final '\0'. En el siguiente ejemplo se muestra un vector que no contiene el fin de la cadena:

```
char cadena3[]={'a'} ; → por lo tanto solo contiene un elemento.
```

Puede pasar que utilicemos una porción de la cadena prevista, como:

```
char cadena4[20]="Algoritmo"; → en estos casos se utilizan los 10
primeros lugares de la cadena, el resto no es «basura», son todas
marcas finales '\0'.
char cadena5[]=""; → En estos casos la cadena contiene el carácter
vacío y el fin de marca '\0'.
```

Al usar vectores debemos tener en cuenta el largo que representa la cantidad de elementos, en el caso de las cadenas, al tener una marca de fin podemos prescindir del largo y procesar una cadena hasta llegar a '\0'.

Ejemplo 14: mostrar la palabra «Programación».

```
#include <stdio.h>
main()
{
char cadena[]="Programación";
int i=0;
while (cadena[i]!='\0') → mientras no aparezca el fin de marca '\0'
    {
   printf("%c",cadena[i]); → muestra cada character de la cadena
en la medida en que se incrementa el subíndice i.
     i++;
    }
}
```

Atención: en el ejemplo anterior también podría haberse mostrado por pantalla la cadena de caracteres con un simple printf("%s",cadena);
Importante: Si en el ejemplo 14 declarábamos la variable "cadena" de la siguiente:

```
char cadena[6]="Programación"
```

entonces solo se tomará en cuenta los primeros 6 caracteres, es decir "Progra"

Ingreso de texto por teclado

Hasta ahora habíamos visto el ingreso de datos (enteros, decimales y caracteres) utilizando la función *scanf*, pero es distinto cuando deseamos ingresar una cadena de caracteres, ya que hay diversas maneras de hacerlo.

El caso más sencillo es simplemente con el uso de la máscara "%s" tanto en el *scanf* como en el *printf*.

Ejemplo 15: se ingresa una palabra por teclado (de no más de 30 caracteres) y se muestra por pantalla.

```
#include <stdio.h>
#define N 30
main()
{
char arreglo[N];
printf("\n Ingresar una palabra: ");
scanf("%s", &arreglo);
printf("La palabra ingresada es:  %s", arreglo);
}
```

Se ve claramente que, tanto en el *scanf* como en el *printf*, no tenemos que especificar el subíndice *arreglo[i]* porque cada uno de los caracteres de la palabra está asociado a la secuencia del vector. Por ejemplo, si ingresamos por teclado la palabra «tiempo» entonces la *t* es el elemento de *arreglo[0]*, la *i* al *arreglo[1]* y así hasta llegar al último "o" en *arreglo[5]* ya que la palabra ingresada se asocia automáticamente a cada uno de los caracteres.

```
arreglo[0] → "t"
…
arreglo[5] → "o"
arreglo[6] → "\0"
```

El último subíndice indica la marca final. Es decir que tenemos que saber que el texto ingresado no supere N elementos ya que en el último elemento es la marca final.

> **Atención:** El **`scanf`** define la lectura hasta el primer espacio en blanco, por ejemplo si ingresamos la siguiente cadena: "Universidad Pública" el **`printf`** solo mostrará "Universidad". Para ingresar cadenas de caracteres sin interrupción por el espacio en blanco usamos la función "**`gets`**" Para el ejemplo anterior sustituimos el **`scanf("%s",&arreglo);`** por: **`gets(arreglo);`**

Funciones para cadena de caracteres

Existen herramientas que ofrece el lenguaje C para facilitar el trabajo a la hora de usar cadena de caracteres.Para eso debemos considerar la incorporación de la librería **string.h** en nuestros programas.

La librería *string.h* tiene una gran cantidad de funciones prácticas para trabajar con cadenas, nosotros veremos algunas de las más importantes.

strlen («cadena de caracteres»): sirve para obtener el tamaño o longitud de la cadena o del vector.

Ejemplo 16: se ingresa una palabra y se muestra su longitud.

```
#include <stdio.h>
#include <string.h> → Librería string
#define N 30
main()
{
char cadena[N];
printf("\n Ingresar una palabra: ");
scanf("%s",&cadena);
int lon;
lon=strlen(cadena);
printf("La longitud del texto ingresado es %i ",lon);
}
```

strcpy («variable de arreglo», «cadena a asignar»): para asignar una expresión de cadena a un arreglo de caracteres, no se puede utilizar el operador de asignación (=). Para ello, debe utilizarse la función strcpy.

Ejemplo 17: se asigna la palabra «Argentina» a un arreglo de caracteres llamado «pais».

```
#include <stdio.h>
#include <string.h>
#define N 30
 main()
{
char pais[N];
strcpy(pais,"Argentina");
```

```
printf(“El país es %s “,pais);
}
```

> **Nota:** también se pueden hacer asignaciones entre cadenas de caracteres.

Ejemplo 18: se asigna la palabra «Argentina» al arreglo «cadena1» y luego se le asigna a «cadena2» el contenido de «cadena1». Finalmente, se muestra el contenido de «cadena2».

```
#include <stdio.h>
#include <string.h>
#define N 30
main()
{
char cadena1[N],cadena2[N];
strcpy(cadena1,”Argentina”);
strcpy(cadena2,cadena1);
printf(“El país es %s “,cadena2);
}
```

strcmp («cadena de caracteres 1», «cadena de caracteres 2»): esta función devuelve un valor numérico según la comparación efectuada entre las dos cadenas de caracteres. Los resultados pueden ser 1, 0 o -1.

```
strcmp(“Argentina”,”Argentina”)→el resultado es cero porque son
                                exactamente iguales las cadenas
strcmp(“Argentina”,”Argentinas”)→el resultado es -1 porque no son
                                  iguales
strcmp(“Argentina”,”Argen”)→el resultado es 1 porque si bien no
                       son iguales, sí los 5 primeros caracteres
```

Ejemplo 19: ingresar dos cadenas de caracteres para compararlas y mostrar si «son iguales» o «no son iguales».

```
#include <stdio.h>
#include <string.h>
#define N 30
main()
```

```
{
char cadena1[N],cadena2[N];
printf(“\n Ingresar una palabra: “);
scanf(“%s”,&cadena1);
printf(“\n Ingresar otra palabra: “);
scanf(“%s”,&cadena2);
if(strcmp(cadena1,cadena2)==0)
     printf(“Son iguales”);
else
      printf(“No son iguales”);
}
```

strcat («cadena de caracteres 1», «cadena de caracteres 2»): esta función concatena las dos cadenas y deja el resultados en «cadena de caracteres1».

Por ejemplo si cadena de caracteres 1 contiene el texto «Hola» y la cadena de caracteres 2 contiene «cómo te va», entonces al aplicar la concatenación *strcat* se obtiene «Hola cómo te va» y lo va a contener la primera cadena.

> **Atención:** Solo concatena de a pares. Para hacerlo más de dos veces debemos volver a utilizar las función **strcat**
> Por ejemplo si queremos concatenar al apellido: “Newton” y el nombre: “Isaac” pero separado por una coma entonces:
>
> apellido[]= “Newton”;
> nombre[]= “Isaac”;
> strcat(apellido,”, “);
> strcat(apellido,nombre);
> printf(«%s»,apellido);
>
> Siempre queda concatenado en la primera variable, en este caso “apellido”

strupr («cadena de caracteres»): Convierte las letras minúsculas en Mayúscula (a – z) otros caracteres no se modifican.

Ejemplos: **cadena1[]={‘R’,’í’,’o’, ,’N’,’e’,’g’,’r’,’o’};**
cadena2[]=strupr(cadena1);

strlwr («cadena de caracteres»): Convierte las letras Mayúsculas en minúsculas (A – Z) otros caracteres no se modifican.

Ejemplo 20: Ingresar dos textos y concatenarlos. Mostrar el resultado en Mayúscula y minúscula.

```
#include <stdio.h>
#include <string.h>
#define N 30
main()
{
char cadena1[N],cadena2[N];
printf("\n Ingresar una palabra: ");
scanf("%s",&cadena1);
printf("\n Ingresar otra palabra: ");
scanf("%s",&cadena2);
strcat(cadena1,cadena2);
printf("\n El resultado es: %s",cadena1);
printf("\n En Mayúscula es: %s",strupr(cadena1);
printf("\n El resultado es: %s",strlwr(cadena1);
}
```

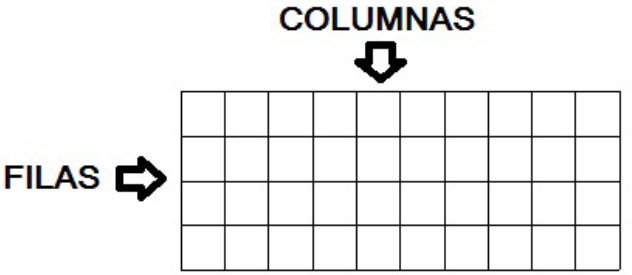
Ejercicios: Cadena de caracteres

26. Escriba un programa que imprima un mensaje de presentación, pregunte cómo se llama el usuario (nombre y apellido) y lo salude. Ejemplo: «Bienvenido Lionel Messi a nuestra casa».
27. Repita el ejercicio anterior pero que ahora también se pida el sexo (M/m/F/f). El programa debe saludarlo anteponiendo «Sr.» o «Sra.». Es decir «Bienvenido Sr.... o Bienvenido Sra....».
28. Escriba un programa que pida el ingreso de un texto por teclado e indique cuántas vocales y cuántas consonantes tiene.
29. Escriba un programa que pida el ingreso de una palabra por teclado y la imprima después de convertir el primer carácter en mayúscula y el resto, en minúsculas.
30. Escriba un programa que pida el ingreso de un texto por teclado e indique la cantidad de palabras que posee.
31. Escriba un programa que pida una cadena por el teclado y la imprima después de convertirla toda a mayúscula.
32. Escriba un programa que pida un texto y luego un carácter (ambos por teclado), e indicar la cantidad de veces que aparece en el texto.

33. Inserción de carácter en cadena. Escriba un programa que pida por el teclado una cadena, un carácter y una posición (número). El programa deberá insertar el carácter ingresado dentro de la cadena en la posición indicada. El programa deberá manejar condiciones de error (ejemplo: si la posición ingresada supera el largo de la cadena).

Matrices

Como se indicó al comienzo del capítulo, las matrices son arreglos. Para el caso de vectores los arreglos son unidimensionales, para las matrices son bidimensionales, es decir tienen filas y columnas.



Se declaran como los vectores pero agregando el máximo número de elementos por fila entre corchetes, por ejemplo int mimatriz[4][10];

Esto quiere decir que el arreglo de nombre «mimatriz» puede tener como máximo 10 columnas y 4 filas, es decir, un máximo de 40 números enteros.

Las posiciones del arreglo son llamadas subíndices y siempre empiezan en cero.

Por ejemplo, si tuviéramos la siguiente matriz:

9	-4	11	8	10	27	2	-9	1	37
0	2	6	-4	9	8	1	2	8	-1
9	2	6	-7	0	21	33	49	2	20
8	1	15	40	45	71	74	-8	4	23

En la posición [0][0] se ubica el número 9, en la [1][3] el -4 y la ultima ubicación es [3][9] con su contenido 23. Recuerden que los subíndices comienzan siempre en cero, por ese motivo se le resta siempre uno.

Atención: el orden de los subíndices es siempre primero la fila y luego la columna. Nosotros llamaremos genéricamente los subíndices como i y j, respectivamente.

Si se aprovecha el ejemplo de la gráfica y queremos cambiar los números 1 por el 13 usando código, entonces será de la siguiente forma:

```
mimatriz[0][8] = 13;
mimatriz[1][6] = 13;
mimatriz[3][1] = 13;
```

Si queremos sumar los dos primeros y los dos últimos elementos del arreglo:

```
sum = mimatriz[0][0]+mimatriz[0][1]+ mimatriz[3][8]+mimatriz[3][9];
```

En cambio, si queremos multiplicar por 4 el elemento ubicado en la fila 2 y la cuarta columna y a este restarle la suma entre el primero y el último elemento de la matriz:

```
sum = 4*mimatriz[1][3]- (mimatriz[0][0]+ mimatriz[3][9]);
```

Si queremos imprimir un elemento en particular del vector hacemos:

```
printf("%i",mimatriz[1][6]);→Va a imprimir el elemento que se
                            encuentra en la fila 2 y la columna 7
```

Cuando queremos recorrer el arreglo bidimensional, ya sea para ingresar, mostrar y/o procesar sus datos, el recorrido es siempre secuencial, entonces utilizamos el doble ciclo de repetición *for* (en estos casos se trata de dos *for*: uno dentro de otro). Hay que tomar en cuenta que por cada fin de fila debe haber un ENTER para bajar el cursor. Se lo hace con un **printf("\n");**

A continuación mostramos algunos ejemplos sencillos en código C.

Ejemplo 21: cómo agregar números enteros por teclado a una matriz llamada «matriz» de 3 filas por 5 columnas.

```
#include<stdio.h>
#define N 3 →para cantidad máxima de filas
#define M 5 →para cantidad máxima de columnas
main()
```

```
{
int matriz[N][M] , i, j;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
             {
             printf(“\n Ingrese un número: “);
             scanf(“%d”,&matriz[i][j]);
             }
}
```

Ejemplo 22: cómo mostrar por pantalla de modo prolijo (fila y columna) la matriz que fue ingresada en el ejemplo anterior.

```
…
for (i=0 ; i<N ; i++)
     {
     for (j=0 ; j<M ; j++)
             printf(“%3i” , matriz[i][j]);
     printf(“\n”);
     }
}
```

Ejemplo 23: la sumatoria de todos los números de la matriz.

```
…
int sum=0,i,j;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
             sum=sum+matriz[i][j];
printf(“\n La suma es %i”,sum);
}
```

Ejemplo 24: la cantidad de números negativos de la matriz.

```
…
int cant=0,i,j;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
             if(matriz[i][j]<0)
                     cant++;
printf(“\n La cantidad es %i”,cant);
}
```

Ejemplo 25: se ingresan a la matriz de 3x5 números consecutivos a partir del cero y mostramos por pantalla.

```
#include<stdio.h>
#define N 3
#define M 5
main()
{
int matriz[N][M] ,i, j,k=0;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
     {
     matriz[i][j]=k;
     k++;
     }
printf("\n La matriz  resultante es:\n");
for (i=0 ; i<N ; i++)
     {
     for (j=0 ; j<M ; j++)
            printf("%3i" , matriz[i][j]);
     printf("\n");
     }
}
```

El resultado por pantalla es el siguiente:

La matriz resultante es:
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14

Es importante mostrar los elementos de una matriz de modo fila x columna para una mejor visión, como en el caso anterior. También, tomar en cuenta la separación entre elementos con un espacio en blanco en la máscara del *printf*.

En el caso anterior fue conveniente utilizar la máscara «%3i» por tratarse de números que tienen como máximo 2 dígitos.

Independientemente de la cantidad de dígitos que tenga el elemento, siempre tendremos la separación en tres espacios.

Atención: cuando declaramos una matriz automáticamente estamos reservando espacio de memoria RAM (memoria central) para luego ser llenado, pero mientras tanto se ha generado «basura» (números sin sentido).

Probar definir un arreglo bidimensional (sin ingreso de datos), mostrarlo como en el ejemplo 22 y se verá por pantalla la «basura».

Si queremos evitar la basura podemos llenar el arreglo en la misma declaración, por ejemplo:

int matriz[N][M]={0} → En estos casos todos los elementos de la matriz son ceros

Con matrices sucede lo mismo que con vectores, podemos de entrada definir su contenido, ejemplo:

`int matriz1[N][M]={{1,1,1,1,1},{2,2,2,2,2},{3,3,3,3,3}}; → a diferencia de los vectores, en la declaración de la matriz necesariamente debemos indicar la cantidad de filas y columnas [N][M]`

Si queremos ingresar un vector columna, por ejemplo:

10
0
-6
8

Lo hacemos de la siguiente manera: **int vecolum[4][1]={{10},{0},{-6},{8}};**

Ejemplo 26: Crear una matriz de 4 x 4 con las vocales A, E, I, O, U al azar y mostrar por pantalla.

```
#include <stdio.h>
#define N 4
#define M 4
main()
{
char matriz[N][M],vocales[]={'A','E','I','O','U'};
int i,j,r;
```

```
srand(time(NULL));
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
             {
             r=rand()%5;
             matriz[i][j]=vocales[r];
             }
printf("\n La matriz  generada es es:\n");
for (i=0 ; i<N ; i++)
     {
       for (j=0 ; j<M ; j++)
             printf("%2c" , matriz[i][j]);
       printf("\n");
     }
}
```

Nota: para agregar a la matriz las letras al azar se utilizó un vector llamado vocales (donde se alojan las 5 vocales) y luego el rand() que permite elegir entre éstas y relacionar índice con vocal. Por ejemplo si el azar arroja el 2 significa que elige la vocal «i».

Ejercicios: Matrices

34. Genere con números al azar del (0 al 9) una matriz de 3 x 3 y muéstrela.
35. Genere con las vocales al azar una matriz de 3 x 5 y muéstrela.
36. Ingrese un valor n y luego un valor m, a continuación generar con números al azar del (0 al 9) una matriz de n x m y muéstrela. Nota: definir como constantes MAX_FILAS y MAX_COLUMNAS para declarar la matriz en memoria.
37. Genere con números al azar del (0 al 9) una matriz de N x M, muéstrela y halle la sumatoria.
38. Genere con números al azar del (0 al 9) una matriz de 4 x 4, muéstrela y halle el promedio.
39. Genere con números al azar del (0 al 9) una matriz de 4 x 4, muéstrela y halle el máximo y el mínimo.
40. Ídem al anterior pero además deberá mostrar su ubicación (fila, columna).
41. Escriba un programa que genere una matriz de N x M con números al azar (del 1 al 9) y permita ingresar por teclado un escalar para efectuar el producto con la matriz. La matriz resultante deberá ser un nueva y mostrarla.

42. Genere con números al azar del (0 al 9) dos matrices de 4 x 4, muéstrelas y halle la suma en una nueva matriz.
43. Genere con números al azar del (0 al 9) una matriz m1 de 3 x 5 y muéstrela. Además se pide hallar los promedios por fila y por columna (mostrar los resultados).
44. Genere con números al azar del (0 al 9) una matriz m1 de 10 x 6, muéstrela y halle los promedios por fila (guardar dichos promedios en un nuevo vector v1 de 10 elementos).
45. Genere con números al azar del (0 al 9) una matriz cuadrada de N x N, muéstrela y halle la sumatoria de la diagonal principal y diagonal inversa.
46. Genere con números al azar del (0 al 9) una matriz de 3 x 4, muéstrela y cree 3 vectores que contengan el contenido de la matriz por fila.
47. Genere con vocales al azar a una matriz de 3 x 4, muéstrela y cree 4 vectores que contengan las vocales de la matriz por columna.
48. Genere con números al azar del (0 al 9) una matriz cuadrada de N x N y halle la sumatoria de sus bordes. (no repetir la suma de sus vértices).
49. Genere con números al azar del (0 al 9) una matriz cuadrada de N x N y hallar la cantidad de números impares y pares que hay en sus bordes (no repetir sus vértices).
50. Genere con números al azar del (0 al 9) una matriz cuadrada de N x N y halle la sumatoria de los números en filas impares.
51. Genere con números al azar del (0 al 9) una matriz cuadrada de N x N y halle la sumatoria de los números en columnas pares.
52. Genere con números al azar del (0 al 9) una matriz cuadrada de N x N y halle la cantidad de números menores a 4 y entre 4 y 7.
53. Genere 3 vectores de 4 elementos cada uno con números al azar del 1 al 9 y cree una matriz de 3 x 4 que contenga los elementos de los vectores.
54. Genere 4 vectores de tamaño 5 con vocales al azar y cree una matriz de 5 X 4 que contenga las vocales de los vectores.
55. Genere con números al azar del (0 al 9) una matriz cuadrada de N x N, muéstrela, halle la transpuesta y muestre el resultado.
56. Tenemos las matrices A y B:

A

3	5	-10	8
0	-9	8	5
7	4	14	11
1	0	3	-2

B

10	0	-3	-8
20	1	4	5
17	-4	4	9
5	1	9	11

Efectúe y muestre las siguientes operaciones:

a. 2A-B
b. (A+B)A (opcional)
c. A2+B2
d. (A-B)2

57. Tomando en cuenta la matriz A del ejercicio anterior y el siguiente vector columna X:

1
-2
6
3

Efectúe y muestre las siguientes operaciones:

a. AX
b. $3A^2X$
c. $(A+A^T)^2X$
d. (X+2X)A

Capítulo 5: Funciones

Modularizar

Hasta el momento, vimos cómo se crea un programa a partir de la utilización de una secuencia de instrucciones armada básicamente de la siguiente forma: primero la entrada de datos, luego el proceso y, por último, la salida de datos.

```
main()
{
Declaración de variables….
Ingreso de datos….
Instrucciones…
Mostrar resultados….
}
```

Esto es muy útil cuando se trata de programas cortos y sencillos, pero cuando tenemos algoritmos largos y complejos, pasa a ser un problema.

Una técnica para resolver dicho problema se basa en dividir el programa grande en subprogramas pequeños, de esta manera, se reduce también la complejidad, atendiendo a la solución por separado y no en su totalidad.

Esta es la forma «divide y vencerás», que se basa en dividir el problema en problemas más pequeños y simples para resolver. Si se resuelve cada uno de estos subproblemas, se resuelve el problema entero.

En programación, a esta técnica se la conoce como **modularización**.

En los distintos lenguajes de programación estos módulos reciben el nombre de: procesos, funciones, rutinas, sub-rutinas, etcétera.

En C, a estos módulos se los llama **función**, y un programa en C está compuesto por una o más funciones: la función *main* y otras (recordar que la función *main* debe estar siempre).

> **Atención:** a partir de ahora el *main* va acompañado del int «*int main()*» porque se toma como función principal y además termina con *return 0;* (devuelve cero). Notarán que los ejemplos que se van a mostrar de ahora en más tienen todos:

```
int main()
{
…
return 0;
}
```

Función

Una función es un conjunto de sentencias que realiza una tarea determinada, responde a un propósito único e identificable. Esto facilita la escritura, evita repetir código, facilidad de mantenimiento, ahorra memoria e independiza los datos.

Generalmente, al llamar a una función se le proporciona un conjunto de datos (parámetros o argumentos) que se procesan con la ejecución de las instrucciones que componen la función.

Una función es una porción de programa, identificable mediante un nombre, que realiza determinadas tareas bien definidas por un grupo de sentencias sobre un conjunto de datos. Las operaciones que realiza la función son siempre las mismas, pero los datos pueden variar cada vez que se llame a la función.

En C, todas las funciones devuelven un solo valor. Algunas funciones reciben parámetros, pero no devuelven nada (por ejemplo *printf*), mientras que otras no reciben parámetros pero sí devuelven un valor (como la función *rand*).

A las funciones que no devuelven valores (o mejor dicho que no tienen como objetivo devolver un dato) las llamamos **procedimientos**. Las explicaremos más adelante.

Definición de una función

Su forma más genérica consta básicamente de dos partes: una línea llamada cabecera, donde se especifica el nombre de la función, el tipo del resultado que devuelve y los parámetros que recibe, y un conjunto de sentencias encerradas entre llaves que forman el cuerpo de la función.

La sintaxis habitual en la definición de una función es:

```
tipo nombre_función (tipo_1 param_1, ..., tipo_N param_N)
{
   Bloque de código
  return(valor de retorno);
}
```

Referencias

Tipo: es el tipo de datos del valor que devuelve la función. Si no se especifica ninguno, C asume que la función devuelve un valor de tipo entero.

Nombre_función: identificador que se usará posteriormente para llamar a la función.

Tipo_i param_i: tipo y nombre de cada uno de los parámetros que recibe la función. Se especifican entre paréntesis y separados por comas. Algunas funciones pueden no tener parámetros (opcional), entonces directamente van los paréntesis (), un claro ejemplo es el *rand()*.

Bloque de código: consiste en la declaración de variables locales e instrucciones necesarias para la realización de la tarea especificada por la función. Si el propósito de la función es devolver un valor, entonces debe incluir la sentencia *return* para devolver un valor al punto de llamada.

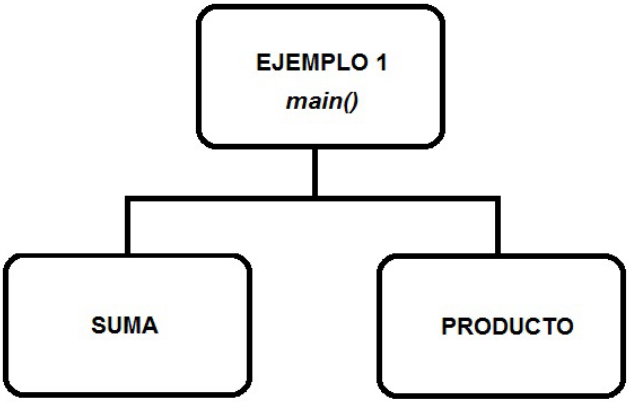
Return: la función genera un valor procesado en la propia función y lo retornará usando la sentencia *return*. Puede retornar un número o un carácter. Esto especifica que la función debe terminar.

Cabe aclarar que el *return* que usamos para concluir con el *main* devuelve siempre 0 (cero).

Entonces, desde la función *main* (y desde cualquier otra función) se puede «invocar» o «llamar» a las funciones definidas. La invocación se realiza utilizando el nombre de la función y sus parámetros.

Ejemplo1: implementar un programa que lea 2 números enteros y calcule la suma y el producto.

Aunque este ejemplo es muy simple, conviene estudiar el diseño *top-down* (de arriba abajo) donde se especifican los módulos en que el problema se divide. Luego, estos módulos se traducen en distintas funciones del programa.



Entonces la implementación es la siguiente:

```
#include <stdio.h>
int suma(int x, int y)
{
    int s;
    s = x + y;
    return(s);
}

int producto(int x, int y)
{
    return(x * y);
}

int main()
{
int a, b, sum, p;
printf("\n Ingrese los dos números: ");
scanf("%d %d", &a, &b);
sum = suma(a,b);
p = producto(a,b);
printf("\n la suma es %d y el producto es %d",sum,p);
return 0;
}
```

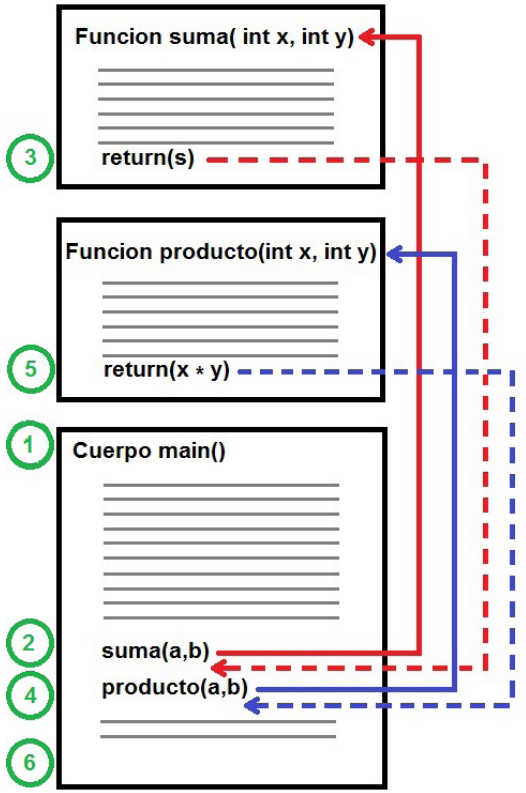
Claramente se ve que en el programa se desarrollan primero las funciones y por último el *main* (cuerpo principal).

Este último se encarga de hacer las llamadas a *suma* y *producto*.

Cada función termina con un *return* (retorno), que permite darle un valor de salida a cada función, de acuerdo al tipo de identificador que ésta tenga. Para nuestro ejemplo 1 ambas funciones son de tipo *int* (entero).

En el caso de la función suma, devuelve el contenido de la variable (definida dentro de la función), en cambio en la función producto no se define ninguna variable porque directamente se realiza el cálculo en el *return*. Es decir que cualquiera de las dos formas es correcta.

A continuación veremos el circuito que establece el ejemplo 1 con sus referencias:



1. Comienzo del programa (siempre en el cuerpo del programa).
2. Llama o invoca a la función *suma* (línea continua roja) con los dos parámetros enteros *a* y *b* que se instancian en cantidad y tipo de parámetro en la llamada. Es muy importante respetar esto, de acuerdo a cómo se ha creado la función y también el orden de los parámetros, aunque en estos casos si se lo hubiera invocado de la siguiente manera: *suma(b,a)* invirtiendo *a,b* por *b,a*, no afectaría el resultado, puesto que la suma es indistinta, pero no así si hubiera pasado con una función de división.
3. Entonces se instancia *a* con *x* y *b* con *y*, es decir que a y b que son variables globales (declaradas en el *main*) pasan a ser *x* e *y*, respectivamente, variables locales (declaradas dentro de la función).
4. La línea punteada roja indica el retorno de la función suma, en estos casos devuelve un número entero, tal como lo establece el tipo de función. El valor es devuelto al mismo lugar donde se realizó la llamada.
5. Llama o invoca a la función *producto* (línea continua azul) con los dos parámetros enteros *a* y *b* que se instancian con *x* e *y*, respectivamente. Acá sucede lo mismo con el orden de los parámetros, si se lo hubiera invocado de la siguiente manera: *producto(b,a)* invirtiendo *a,b* por *b,a*, no afectaría el resultado, puesto que el producto no altera el resultado.
6. La línea punteada azul indica el retorno de la función producto. A diferencia de la función *suma*, ésta no utiliza una variable local, porque directamente realiza en el *return* la operación.
7. Finalmente, termina el programa.

Atención: varios datos para funciones a tomar en cuenta: se recomienda elegir nombres acordes a la acción de la función. Respetar la cantidad, el tipo y el orden de los parámetros de la función, como así también el nombre de ésta. Recordar el retorno, *return*.
Las variables locales de una función pueden repetirse en otras funciones ya que su alcance comienza y termina en el propio módulo. En el ejemplo 1 se ve claramente que *x* e *y* se utilizaron en las dos funciones. El orden de las funciones es indistinto, pero sí deben estar antes que el *main*.

En el ejemplo 1 se ve claramente en el cuerpo del programa *main* que las llamadas a las dos funciones son asignaciones a variables *sum* y *p*, que luego serán impresas en el *printf*

```
 sum = suma(a,b);
 p = producto(a,b);
 printf("\n la suma es %d y el producto es %d",sum,p);
```

Pero también podría haberse evitado si se ahorraban líneas de código y variables de la siguiente forma:

```
printf("\n la suma es %d y el producto es %d", suma(a,b),
producto(a,b));
```

Ejemplo 2: implementar un programa que lea un número entero y calcule su factorial y su cuadrado.

```
#include <stdio.h>
int factorial(int x)
{
 int i, aux=1;
 for (i=2; i<=x; i++)
     aux = aux * i;
return(aux);
}

int cuadrado(int x)
     {
    return(x * x);
     }
int main()
{
int a;
printf("\n Ingrese el número: ");
scanf("%d", &a);
printf("\n El factorial del número %d es %d", a, factorial(a));
printf("\n El cuadrado del  número %d es %d", a, cuadrado(a));
return 0;
}
```

En el ejemplo anterior no se usaron variables de asignación en el *main* para las funciones, porque directamente se las invocó en el *printf*. Cabe aclarar que cualquiera de las dos maneras es correcta.

Atención: es conveniente que las variables globales sean distintas a las locales, es decir que para el ejemplo 2, se usó la variable global *a* para el *main,* que instancia con la variable local *x* de cada función.

Las funciones pueden ser llamadas dentro de una operación aritmética:

```
T = suma(a,b) / producto(a,b); → T=(x+y)/(x.y)
T = suma(a,a) / producto(b,b); → T=(x+x)/(y.y)
```

O también una dentro de otra:

```
T = producto(a, suma(a,b)); → T=x.(x+y)
T = suma(producto(b,b)+producto(a,a)); → T= y²+x²
T= producto(producto(suma(a,b),b),a); → T=((x+y).y).x
```

A continuación, mostraremos un ejemplo donde una función llama a otra.

Ejemplo 3: implementar un programa que lea los dos catetos de un triángulo rectángulo para hallar su perímetro, usar las funciones «cuadrado», «suma» e «hipotenusa».

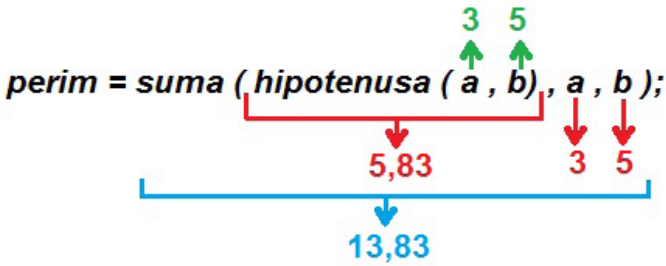
Sabemos que el perímetro de un triángulo es la suma de sus lados.

```
#include <stdio.h>
#include <math.h> →Se debe incluir la librería math.h para
                   utilizar la raíz cuadrada sqrt.
float cuadrado(float x) →Esta función debe estar necesariamente
                         antes que la función hipotenusa, quien la
                         llama o invoca.
     {
    return(x * x);
     }
float hipotenusa(float x,float y)
     {
    float h;
    h=sqrt(cuadrado(x)+cuadrado(y));
    return(h);
     }
float suma(float x,float y, float z)
```

```
    {
    return(x+y+z);
    }
int main()
{
float a,b,perim;
printf("\n Ingrese los catetos del triangulo: ");
scanf("%f %f", &a, &b);
perim=suma(hipotenusa(a,b),a,b);
printf("\n El perímetro del triángulo es %0.2f",perim);
return 0;
}
```

> **Nota:** las tres funciones devuelven un *float*, por tratarse de números decimales.
> Por otra parte, la librería que se utiliza para una de la funciones se invoca al principio con *#include <math.h>*
> En el *main* aparece la asignación *perim=suma(hipotenusa(a,b),a,b);*

Veamos un ejemplo para comprender un poco más su funcionamiento. Si a = 3 y b = 5 entonces:



Es decir que, en estos casos, tenemos una función dentro de otra.
Veamos otro ejemplo donde se utiliza el retorno de una función de tipo *char*.

Ejemplo 4: implementar un programa que muestre si la resta de dos números devuelve un número negativo, positivo o nulo usando una función cuyo retorno sea un carácter ("-","+","N"), respectivamente:

```
#include <stdio.h>
char resta(int x,int y)
```

```
{
    int res;
    res=x-y;
    if(res==0)
        return('N');
    else
        if(res>0)
            return('+');
        else
            return('-');
}
int main()
{
int a,b;
printf("\n Ingrese los dos números: ");
scanf("%d %d", &a, &b);
printf("\n La diferencia entre %i y %i es %c", a, b, resta(a,b));
return 0;
}
```

Nota: además se puede observar que en el retorno de la función *resta* existen tres *return*, pero solo uno de éstos es el que debe funcionar (depende de la respuesta del *if*).

Atención: las funciones no devuelven arreglos, solo datos de tipo escalar, es decir punteros, numéricos (*float* e *int*) y caracteres (*char*).
Si quisiéramos retornar un arreglo, entonces tendríamos que utilizar procedimientos.

Variables globales y locales

Variables globales

Se crean durante toda la ejecución del programa y son globales ya que pueden ser llamadas, leídas, modificadas, etcétera, desde cualquier función, incluso del mismo *main*. Las variables se definen antes del *main()*.

Variables locales

Pueden ser utilizadas únicamente en la función que hayan sido declaradas.

A continuación mostraremos un ejemplo que utilice variables globales y locales.

Ejemplo 5: se genera un vector de 100 números aleatorios (del 0 al 10), luego el usuario elige una cantidad menor o igual a 100 para hallar su promedio. Para el promedio debe utilizarse una función.

```
#include <stdio.h>
#define N 100
int c;  →Esta es la variable global (puede utilizarse tanto en el
         main como dentro de las funciones)
float promedio(int v[])
     {
     int i,s=0;  →s es una variable local por estar definida dentro
                  de la función
     for (i=0;i<c;i++) → c mantiene el valor global
            s=v[i]+s;
     return((float) s/c);
}
void mostrar(int v[])
     {
     int i;
     for (i=0;i<N;i++)
       printf("%3d",v[i]);→ %3d significa que va a dejar siempre 3
                          lugares independientemente de la cantidad de
                          dígitos de cada elemento del vector
     }
int main()
{
int i,vec[N];
srand(time(NULL));
for (i=0;i<N;i++)
     vec[i]=rand()%11;
printf("\nEl vector\n");
mostrar(vec);
printf("\nIngrese la cantidad de elementos para el promedio
del vector: ");
```

```
scanf(“%d”,&c);
printf(“\nEl promedio es: %0.2f”,promedio(vec));
return 0;
}
```

Procedimiento

Existen funciones que no utilizan la instrucción *return*, es decir no devuelve ningún valor, entonces pasan a ser **procedimientos**. En muchos lenguajes de programación separan totalmente el concepto de funciones del de procedimientos, pero C los trata de igual forma. En pocas palabras, el procedimiento es una función sin valor de retorno, pero que pueden realizar una tarea, por ejemplo modificar los elementos de una matriz, crear un vector de datos, etcétera.

Un claro ejemplo es el procedimiento *printf* que simplemente imprime lo que quisiéramos entre comillas.

La sintaxis habitual en la definición de un procedimiento es:

```
void nombre_procedimiento (tipo_1 param_1, ..., tipo_N param_N)
    {
    Bloque de código
    }
```

Referencias:

Void: es la palabra cabecera que no debe faltar en los procedimientos.

Nombre_procedimiento: identificador que se usará posteriormente para llamar al procedimiento.

Tipo_i param_i: tipo y nombre de cada uno de los parámetros que recibe el procedimiento. Se especifican entre paréntesis y separados por comas. Algunos procedimientos pueden no tener parámetros (opcional), entonces directamente van los paréntesis *( )*, un claro ejemplo es el *main() y el rand()*.

Bloque de código: consiste en la declaración de variables locales e instrucciones que son necesarias para la realización de la tarea especificada por el procedimiento.

Pero para aclarar aun más el concepto veamos el siguiente ejemplo:

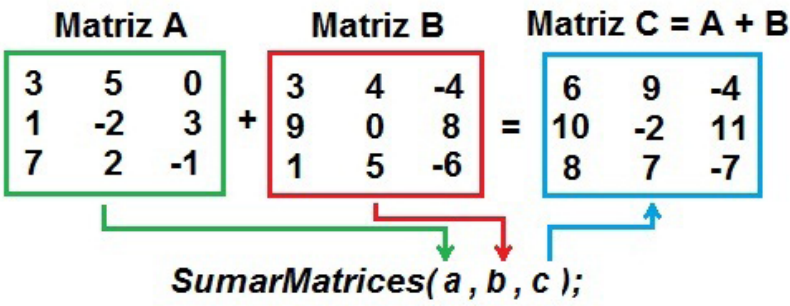
Ejemplo 6: implementar un programa que muestre un cartel según si la resta de dos números devuelve un número negativo, positivo o nulo usando un procedimiento.

```
#include <stdio.h>
void  resta(int x, int y)
{
    int res;
    res=x-y;
    if(res==0)
            printf("\n El resultado es nulo");
    else
            if(res>0)
                    printf("\n El resultado es positivo");
else
            printf("\n El resultado es negativo");
}
int main()
{
int a,b;
printf("\n Ingrese los dos números: ");
scanf("%d  %d", &a, &b);
resta(a,b);
return 0;
}
```

Nota: claramente se observa en el *main* del ejemplo la llamada al procedimiento *resta(a,b)*.

Los procedimientos, a diferencia de las funciones, son invocados en el *main* sin utilizar asignación, además son muy útiles para trabajar con arreglos (vectores y matrices), porque permiten crear, mostrar y modificarlos. Recordemos que las funciones no pueden devolver arreglos.

La siguiente es una gráfica que representa un ejemplo de la función de los parámetros de un procedimiento:



El procedimiento *sumar matrices* contiene tres parámetros, los dos primeros de entrada y el tercero de salida. Todas las matrices fueron definidas en el cuerpo del programa, pero solo a *a* y *b* se les agregaron elementos (números enteros). Dentro del procedimiento se realiza la suma para obtener un nuevo *c*.

```
void SumarMatrices(int m1, int m2, int m3)
{
int i,j;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<N ; j++)
            m3[i] [j]=m1[i] [j] + m2[i] [j];
}
```

Sabiendo que *a* instancia con *m1*, *b* con *m2* y *c* con *m3*.

A continuación veremos un ejemplo relacionado.

Ejemplo 7: implementar un programa que genere con números aleatorios dos vectores v1 y v2 de N enteros que oscilan entre (0 y el 10), luego mostrarlo.

```
#include <stdio.h>
#define N 30
void generar(int v[])
{
     int i;
     for (i=0;i<N;i++)
            v[i]=rand()%11;
```

```
}
void mostrar(int v[])
{
     int i;
     for (i=0;i<N;i++)
            printf("%i\t",v[i]);
}
int main()
{
int v1[N],v2[N];
srand(time(NULL));
generar (v1);
generar (v2);
printf("\n El Vector 1\n");
mostrar(v1);
printf("\n El Vector 2\n");
mostrar(v2);
return 0;
}
```

El procedimiento *generar* recibe el parámetro *v1* (vector declarado en el *main*), que en un principio contiene «basura», pero al terminar el procedimiento cambia al tener números aleatorios generados por el primero.

Además se puede observar que un solo procedimiento puede generar dos vectores (*v1* y *v2*), lo mismo sucede con *mostrar*.

Supongamos que al ejercicio anterior le agregamos una consigna. Crear un tercer vector *v3* que sea el resultado del promedio entre los vectores *v1* y *v2* (índice por índice). Es decir que *v3* también será de dimensión *N* pero al ser un promedio los elementos que la integran son del tipo *float*.

Entonces el procedimiento será el siguiente:

```
void promedio(int va[], int vb[], float vc[])
     {
     int i;
     for (i=0;i<N;i++)
            vc[i] =(float) (va[i]+vb[i])/2;
     }
```

Nota: el tercer vector que internamente llamamos *vc* debe ser de tipo *float* porque se trata de un promedio.

Y la llamada o invocación en el *main* será:

```
promedio(v1,v2,v3);
```

Siendo *v3* declarado en el cuerpo del programa como *float v3[N];*

Atención: en estos casos no podemos utilizar el procedimiento mostrar, porque su parámetro es de tipo *int* y para el caso del vector v3 necesitamos un nuevo procedimiento que reciba el parámetro adecuado, lo llamaremos *mostrarf* para diferenciarlo del procedimiento *mostrar*.

```
void mostrarf(float v[])
{
int i;
for (i=0;i<N;i++)
     printf("%0.2f\t",v[i]);
}
```

Atención: recordemos que los parámetros de una función pueden ser opcionales, es decir que pueden estar o no y si están se los identifica a cada uno separado por comas.

A continuación mostramos un ejemplo de uso de matrices combinando funciones, procedimientos variables locales y globales.

Ejemplo 8: crear y mostrar una matriz de N X M con las vocales aleatoriamente. Luego el usuario debe ingresar una vocal para hallar la cantidad dentro de la matriz. Usar funciones y procedimientos.

```
#include <stdio.h>
#define N 3
#define M 4
char vocales[]={'A','E','I','O','U'};→ El vector «vocales»
```

```
                                        declarado como Variable Global
void crearmatriz(char ma[N][M])
{
int i,j,r;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
     {
                        r=rand()%5;
ma[i][j]=vocales[r];→Utiliza el vector global «vocales» para generar
                              la matriz aleatoriamente
     }
}
void mostrarmatriz(char ma[N][M])
{
int i,j;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
             printf("%2c" , ma[i][j]);
     printf("\n");
}
int cantidadvocales(char ma[N][M],char v) →La función recibe los
                                            parámetros Ma y V (la
                                            matriz y la vocal elegida
                                            por el usuario
                                            respectivamente)
{
int i,j,c=0;
for (i=0 ; i<N ; i++)
     for (j=0 ; j<M ; j++)
     {
             if(ma[i][j]==v)
                     c++;
     }
return(c);
}
int main()
{
char m1[N][M],vocal;
srand(time(NULL));
```

```
crearmatriz(m1);
printf(“\n La matriz  generada es es:\n”);
mostrarmatriz(m1);
printf(“\nIngrese la vocal: “);
scanf(“%c”,&vocal);
printf(“\nLa cantidad de vocales %c es %i”,vocal,
                    cantidadvocales(m1,vocal));
return 0;
}
```

Menús

En programación el menú aparece cuando el usuario debe optar por distintas aplicaciones.

Estas opciones se presentan por pantalla, preferentemente de modo vertical y están dentro de un bucle con la posibilidad de terminar y salir de este.

Por ejemplo, si se tratara de una mini calculadora, cuyas operaciones básicas son sumar, restar, dividir y multiplicar, la pantalla deberá presentar el siguiente aspecto:

```
  1 - SUMAR
  2 - RESTAR
  3 - DIVIDIR
  4 - MULTIPLICAR
  5 - SALIR
--------------------------------
Ingrese los dos números:
```

Luego de ingresar los dos números aparece el siguiente mensaje:

```
Ingrese la opción:
```

Ejercicio 9: realizar una mini calculadora con las operaciones básicas (suma, resta, división y multiplicación) usando menú.

```
#include <stdio.h>
float sumar(float a,float b)
{
```

```
    return(a+b);
}
float restar(float a,float b)
{
    return(a-b);
}
float multiplicar(float a,float b)
{
    return(a*b);
}
float dividir(float a,float b)
{
    return(a/b);
}
int main()
{
int opcion;
float n1,n2;
do
{
     printf("\n   1 - SUMAR");
     printf("\n   2 - RESTAR");
     printf("\n   3 - DIVIDIR");
     printf("\n   4 - MULTIPLICAR");
     printf("\n   5 - SALIR");
     printf("\n---------------------");
     printf("\n Ingrese los dos números: ");
     scanf("%f %f",&n1,&n2);
     printf("\n Ingrese la opción: ");
     scanf("%i",&opción);
     switch(opción)
            {
            case 1:
               printf("\nEl resultado es %0.2f",sumar(n1,n2));
               break;
            case 2:
               printf("\nEl resultado es %0.2f",restar(n1,n2));
               break;
            case 3:
```

```
            if(n2!=0.0) →Se asegura de que el numerador sea
                              distinto de cero para la división
            printf("\nEl resultado es %0.2f",dividir(n1,n2));
             else
            printf("\n ERROR"); → Si es cero muestra el error
                    break;
            case 4:
            printf("\nEl resultado es %0.2f",multiplicar(n1,n2));
                    break;
            default:
                    opcion=5;→Cualquier otro número fuera de las
                              opciones se le asigna 5 para salir
                    break;
             }
}while (opcion!=5);
printf("\n Gracias y Buena Suerte!!! ");
return 0;
}
```

Atención: el único inconveniente que presenta el ejemplo anterior es que antes de salir del menú el usuario debe ingresar los dos números. La intención es que se muestre el funcionamiento del menú. Intente resolver el ejercicio de la calculadora con un *while* (y no con un *do – while*) para evitar dicho inconveniente. Por lo general, los menús con muchas opciones van acompañados de un *switch*.

Ejercicios: Funciones y procedimientos

1. Escriba un programa que solicite el ingreso del grado de un ángulo y con un menú por pantalla que permita optar por hallar el A - seno, B - coseno y C - tangente de dicho ángulo. Recuerde que con D sale y que antes de resolver trigonométricamente debe hacer el pasaje de grados a radianes. Incluir la librería *math.h*.
2. Realice un menú que tenga las siguientes opciones: 1-Área 2-Perímetro 3-Hipotenusa 4-Salir. Ingresando los dos catetos a y b de un triángulo rectángulo. Incluir la librería *math.h*.
3. Realice un programa que ingrese a un vector de 20 posiciones, números al azar del 1 al 10, muestre dichos números, halle la cantidad

de números mayores a 5 y halle la cantidad de números menores o iguales que 5. Mostrar ambos resultados.

4. Ingrese un vector de números binarios (0 y 1) y cree otro vector invirtiendo los números del primer vector. Mostrar ambos vectores.
5. Ingrese un vector con 20 números enteros al azar del 1 al 10. Muestre la cantidad de números impares. Calcule la cantidad de números 1 y 3.
6. Ingrese a un vector de 20 posiciones números de un dado al azar, cuente la cantidad de 6 y sustituya el 5 por el 3.
7. Ingrese un vector con 20 números enteros al azar del 1 al 10. Halle el promedio. Cantidad de pares y cantidad de impares.
8. Ingrese números al azar (del 1 al 9) a dos vectores de nombre v1 y v2. Muéstrelos y luego halle y muestre la cantidad de números pares que se encuentran en cada vector.
9. Ingrese un vector con 20 números enteros al azar del 1 al 10. Cambie los números pares por 0. Cambie los impares por 1. Muestre el vector resultante. Muestre la cantidad de impares y pares.
10. Ingrese un vector «v» con 20 números enteros al azar del 1 al 10. Y usando dos vectores «vi» y «vp» de 20 elementos, llene al primero con los números impares de «v», y al otro «vp» los pares (sin respetar la posición original, es decir que posiblemente quede «basura» en los elementos finales de cada vector).
    Muestre los tres vectores (sin la basura).
11. Ingrese dos vectores de 20 números enteros al azar del 1 al 10. Sume cada elemento de ambos vectores para agregarlo en otro nuevo vector. Halle de ambos vectores el promedio en cada posición y agregue el resultado en otro nuevo vector. Finalmente, muestre los cuatro vectores resultantes.
12. Ingrese dos vectores con 20 números al azar del 0 al 10 («va» y «vb»). Cree otro vector «vc» que tenga en cada posición la letra «A» o «B» según el mayor de ambos (pero si son iguales se coloca el "=") y mostrar el vector «vc». Muestre la cantidad de «A» y de «B». Por ejemplo:

va	→	7	5	2	8	...	10
vb	→	1	5	7	9	...	0
vc	→	A	=	B	B	...	A

13. Ingrese tres vectores v1, v2 y v3 con 30 números al azar del 1 al 10. En otro vector p debe tener el promedio (de cada posición de los tres vectores), es decir que finalmente nos quedará un vector de 30 elementos reales. Mostrar el vector p y su valor máximo.
14. Llene una matriz de 4 X 4 con números de un dado al azar. Muéstrelo y luego halle la sumatoria de los números que se encuentran en las filas impares.
15. Llene una matriz de 4 X 4 con números de un dado al azar. Muéstrelo y luego halle la sumatoria de los números que se encuentran en las columnas pares.
16. Ingrese a una matriz de 3 X 4 con números al azar (del 0 al 9), muéstrela y halle la cantidad de números pares.
17. Ingrese a una matriz de 4 X 4 con números al azar (del 0 al 9), muéstrela y cree 4 vectores conteniendo las filas de la matriz, muétrelos.
18. Ingrese a una matriz de 3 X 4 con números al azar binarios (0 y 1), muéstrela y cree otra matriz traspuesta, muéstrela.
19. Cree un programa con un menú de opciones para 1 - Mostrar la matriz; 2- Mostrar la matriz traspuesta; 3-Mostrar el inverso de la matriz (0 por 1 y viceversa); 4-Generar una nueva matriz y 5-Salir. La matriz de 5 X 5 debe contener números binarios generados aleatoriamente. En un principio, ésta debe contener ceros.
20. Ingrese 3 matrices (m1, m2 y m3) de 4 X 4 con números al azar (del 0 al 9), halle la suma de m1+m2, m2+m3 o m1+m3 y muestre cada resultado. Use el menú de opciones.
21. Ingrese una matriz de 5 X 5 de números aleatorios del 0 al 9. Use un menú de opciones para: A – sumar; B – restar; C – Multiplicar y D – Dividir, un escalar ingresado por el usuario. Con X – Salir. En el caso de la suma, el escalar debe sumarse con cada elemento de la matriz. Es decir que el resultado será una matriz de 5x5 con cada elemento sumado con el escalar. En caso análogo pero con la resta.
22. Desarrolle las funciones y procedimientos de acuerdo y acorde al siguiente cuerpo del programa:

```
#include<stdio.h>
#define N 20
int main() {
int v1[N], v2[N], v3[N], m[3][N];
srand(time(NULL);
CargarVector(v1); →Carga a un vector de N elementos números
                    aleatorios enteros de [-10,10]
MostrarVector(v1);
CargarVector(v2); →Carga a un vector de N elementos números
                    aleatorios enteros de [-10,10]
```

```
MostrarVector(v2);
CargarVector(v3); →Carga a un vector de N elementos números
                    aleatorios enteros de [-10,10]
MostrarVector(v3);
ArmarMatriz(v1,v2,v3,m); →Con los tres vectores se arma una
                           matriz de 3 X N
MostrarMatriz(m);
printf("\n El Promedio es %0.2f ",PromedioMatriz(m));
return 0;
}
```

Recursividad

También conocida como *recurrencia* o *recursión*, es una técnica utilizada en muchos lenguajes de programación estructurada. Habíamos visto la posibilidad en programación de que una función llame a otra función, pues en estos casos y para su mayor entendimiento, en la recursividad las funciones se llaman o invocan a sí mismas.

En muchos casos, esta técnica es una alternativa de la implementación de bucles en el algoritmo.

En la recursividad hay que tener mucho cuidado de no meterse en problemas sin llegar a un punto final, haciendo que el proceso resulte infinito.

Un típico y claro ejemplo es la obtención del factorial de un número. Sabemos que el factorial de cero es 1 y que el factorial de cualquier otro número natural N es:

```
N x (N-1) x (N-2) x (N-3) x… x 1
```

Por ejemplo, el factorial de 5 es 120 porque: *5 x 4 x 3 x 2 x 1 = 120*

Veamos a continuación el algoritmo relacionado con esta técnica, para hallar el factorial de un número.

Ejemplo 10: factorial de un número.

```
#include <stdio.h>
int factorial(int n)
{
  int r;
  if(n==0)
```

```
     return(1);
   else
     r=n*factorial(n-1); →A medida que se va llamando a la función
                           factorial el número n se reduce uno n-1
     return(r);
}
int main()
{
int n,f;
printf("\nIngrese un número: ");
scanf("%d",&n);
f=factorial(n);
printf("El factorial de %d es %d",n,f);
return 0;
}
```

Nota: claramente se ve cómo la función se llama a sí misma en el *else* de la condición *if*, frenando este proceso de recursión cuando el valor o parámetro de la función llega a cero.
Atención: para usar la técnica de recursión siempre debe haber una condición de salida y una de llamada a sí mismo, ambas deben tener el retorno *return*.

Ejemplo 11: la sumatoria de un número. Por ejemplo de 5 es 5 + 4 + 3 + 2 + 1 = 15.

```
#include <stdio.h>
int sumatoria(int n)
{
     int s;
     if(n==1)
            return(1);
     else
           s=n+sumatoria(n-1); →A medida que se va llamando a la función
                                sumatoria el número n se va decrementando
                                en 1 con n-1
     return(s);
}
int main()
{
```

```
int n,sum;
printf("\nIngrese un número: ");
scanf("%d",&n);
sum=sumatoria(n);
printf("La sumatoria de %d es %d", n, sum);
return 0;
}
```

Nota: claramente se ve cómo la función se llama a sí misma en el *else* de la condición *if*, frenando este proceso de recursión cuando el valor o parámetro de la función llega a uno.
También se pudo haber puesto en el *printf* lo siguiente: *printf("La sumatoria de %d es %d, n, sumatoria(n));*

Ejercicios: Recursividad

23. Realice un algoritmo usando recursividad para el producto a x b. Una ayuda usar → producto(a,b-1). Los números a y b ingresados por el usuario deben ser naturales.
24. Realice un algoritmo con recursividad para la potencia a^b. Los números a y b ingresados por el usuario deben ser naturales.
25. Realice la siguiente serie: $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots\frac{1}{n}$ sabiendo que n (natural) debe ingresarlo el usuario.
26. Realice la siguiente serie: $\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+\cdots\frac{n-1}{n}$ sabiendo que n (natural) debe ingresarlo el usuario.

Estructura de datos

En la creación de soluciones para algunos problemas surge la necesidad de agrupar datos de diferente tipo o de manejar datos que serían muy difíciles de describir en los tipos de datos primitivos, esta es la situación en la que debemos aprovecharnos de las características que hacen del C un lenguaje especial, o sea el uso de estructuras de datos.

Una estructura puede contener varios datos (incluso de distintos tipos). La forma de definir una estructura es haciendo uso de la palabra clave *struct*.

```
struct miestructura
```

```
{
int legajo,edad;
float sueldo;
char categoría; } ;
```

Más adelante se declara una variable con el nombre «datos» de tipo *miestructura* de la siguiente manera:

```
struct miestructura datos;
```

Entonces para referirnos a cualquier dato de una estructura:

```
nombre_de_ estructura.nombre_ de_ variable
```

Para llenar los datos del tipo de estructura del ejemplo:

```
datos.legajo=100;
datos.edad=23;
datos.sueldo=5500.00;
datos.categoría='b';
```

Si deseamos ingresar por teclado los datos usamos lo siguiente:

```
printf("\nIngresar nº de legajo ");
scanf("%i",&datos.legajo);
printf("\nIngresar edad ");
scanf("%i",&datos.edad);
printf("\nIngresar sueldo ");
scanf("%f",&datos.sueldo);
printf("\nIngresar categoría ");
scanf(" %c",&datos.categoría);
```

> **Nota:** recuerden que el espacio en blanco en el scanf cuando usamos datos de tipo caracter sirve para obviar el "enter" → "\0" que se ingresó en el último scanf.

Ejemplo 12: el siguiente es un algoritmo de ejemplo para ingresar un registro de un empleado: (n.º de legajo, edad, sueldo y categoría), y luego muestra por pantalla el registro en una línea.

```
#include <stdio.h>
int main()
{
struct miestructura
 {
 int legajo,edad;
 float sueldo;
 char categoría;};
struct miestructura empleado;
printf(“\nIngresar num de legajo “);
scanf(“%i”,&empleado.legajo);
printf(“\nIngresar edad “);
scanf(“%i”,&empleado.edad);
printf(“\nIngresar sueldo “);
scanf(“%f”,&empleado.sueldo);
while(getchar()!=’\n’);
printf(“\nIngresar categoría “);
scanf(“%c”,&empleado.categoría);
printf(“\nDatos del empleado\n”);
printf(“\nLegajo %i ,la edad %i , Categoría %c y sueldo %0.2f\n”,
empleado.legajo, empleado.edad, empleado.categoría, empleado.suel-
do);
return 0;
}
```

Pero para ingresar más de un registro, debemos utilizar *arrays* (vector de datos) y agregar [i] al dato de la siguiente manera:

```
nombre_de_ estructura[i].nombre_ de_ variable
```

Ejercicio 13: modificar el ejemplo anterior, llenar y mostrar N registros de la estructura *miestructura* será:

```
#include <stdio.h>
#define N 30
int main()
{
struct miestructura
 {
```

```
 int legajo,edad;
 float sueldo;
 char categoría;};
struct miestructura empleado [N];
int i;
for(i=0;i<N;i++)
     {
     printf(“\nIngresar nº de legajo “);
     scanf(“%i”,& empleado [i].legajo);
     printf(“\nIngresar edad “);
     scanf(“%i”,& empleado [i].edad);
     printf(“\nIngresar sueldo “);
     scanf(“%f”,& empleado [i].sueldo);
     while(getchar()!=’\n’);
     printf(“\nIngresar categoría “);
     scanf(“%c”,& empleado [i].categoría);
     }
printf(“\nDatos de los empleados\n”);
for(i=0;i<N;i++)
    printf(“\nLegajo %i y edad %i Categoría %c y sueldo %0.2f\n”,
empleado [i].legajo, empleado [i].edad, empleado [i].categoría, em-
pleado [i].sueldo);
return 0;
}
```

Si tenemos que aplicar ordenamiento en una estructura de datos debemos utilizar una única variable auxiliar para intercambiar todos los datos. Si aplicamos el método de burbuja siguiendo el ejemplo anterior, debemos definir la variable *aux* de la siguiente manera:

```
struct miestructura empleado [N],aux;
```

y el procedimiento de intercambio será de la siguiente manera:

```
for(i=0;i<N-1;i++)
    for (j=i+1;j<N;j++)
         if (empleado [i].legajo> empleado [j].legajo)
            {
            aux= empleado [i];
```

```
empleado [i]= empleado [j];
empleado [j]=aux;
}
```

Para poder combinar estructura de datos con funciones y procedimientos, la definición struct debe figurar al inicio (luego de las librerías y constantes y antes de las funciones y procedimientos). A continuación veremos un ejemplo relacionado.

Ejercicio 14: Realizar el ejercicio 12 y 13 utilizando funciones y procedimientos

```
#include <stdio.h>
#define N 30
struct miestructura
 {
 int legajo,edad;
 float sueldo;
 char categoria;};
void generar(struct miestructura emp[N])
{
int i;
for(i=0;i<N;i++)
    {
    printf("\nIngresar nº de legajo ");
    scanf("%i",&emp[i].legajo);
    printf("\nIngresar edad ");
    scanf("%i",&emp[i].edad);
    printf("\nIngresar sueldo ");
    scanf("%f",&emp[i].sueldo);
    printf("\nIngresar categoría ");
    scanf(" %c",&emp[i].categoria);
    }
}
void mostrar(struct miestructura emp[N])
{
int i;
printf("\nDatos de los empleados\n");
for(i=0;i<N;i++)
   printf("\nLegajo %i y edad %i Categoría %c y sueldo %0.2f\n",emp[i].
legajo, emp[i].edad, emp[i].categoria, emp[i].sueldo);
```

```
}
void ordenar(struct miestructura emp[N])
{
int i,j;
struct miestructura aux;
for(i=0;i<N-1;i++)
    for (j=i+1;j<N;j++)
        if (emp[i].legajo> emp[j].legajo)
            {
            aux= emp[i];
            emp[i]= emp[j];
            emp[j]=aux;
            }
}
int main()
{
struct miestructura empleado [N];
generar(empleado);
ordenar(empleado);
mostrar(empleado);
return 0;
}
```

Ejercicios: Estructura de datos

27. Realice un algoritmo para llenar el siguiente registro de N países:
    Código internacional telefónico:
    Continente: ('a' → América, 'b'→ Asia, 'c'→ África, 'e' → Europa 'o'→ Oceanía)
    Población:
    Superficie:
    Luego mostrarlos.
28. Del anterior, agregue lo siguiente: halle la cantidad total de población y promedio total de superficie.
29. Del ejercicio 27 agregue lo siguiente: ordene de modo ascendente por Código internacional telefónico y muestre todos los datos.
30. Cree una estructura que almacene x y f(x).
    La función es la siguiente: $f(x) = \cos(x)^2-3x$.
    Con $x \in [-10, 10]$ con pasos de 0.5 automáticamente. Muestre los datos.

Nota: no olvidarse de agregar la librería *math.h* para reconocer la función trigonométrica.

31. Ídem al anterior, solo que los datos deben ingresarse al azar usando el mismo rango x ϵ [-10, 10] y un dígito decimal del 0 al 9.
    Ordenarlos de modo descendente por X y mostrar los datos.

Nota: son N registros.

32. Cree una estructura que almacene los lados a y b de un triángulo rectángulo (obtenidos al azar en un rango de [1,100] solo números enteros) y su hipotenusa.
    En total son 10 triángulos. Además debe agregar un campo llamado «código» de tipo char para identificar al triangulo ('A', 'B', 'C', ...,'J').
    Al ingresar el código, el usuario podrá mostrar y editar los datos del triángulo correspondiente.

33. Para almacenar tres datos: Dos números enteros N1 y N2 y un carácter TIPO que contenga (P- si son los dos números son pares, I- si son los dos números son impares y M- si uno es par y el otro impar) Debe hacerlo automáticamente el programa. La cantidad total de registros es 100. Luego hallar la sumatoria de los N1 (Todos) y el promedio de los N2 (solo los del TIPO M). Mostrar ambos resultados. El usuario solo debe cargar los dos números N1 y N2.

34. Para almacenar 100 triángulos cuyos datos deben ser: El código (tres dígitos), las longitudes en cm de sus tres lados A, B y C y además su tipo: (E - Escaleno, Q - Equilátero, I - Isósceles). El usuario solo debe ingresar el código y los lados de cada triangulo, mientras que el tipo lo debe determinar el programa (según sus lados).
    Mostrar los datos de la siguiente manera:

Código	Lado A	Lado B	Lado C	Tipo
102	4,10	5,00	4,10	I
117	7,15	1,00	3,80	E

Nota: Antes de mostrarlo deberá ordenarlos de modo ascendente según el código.

Capítulo 6: Planilla de cálculo-Calc

La planilla de cálculo Calc pertenece al paquete ofimático *Libre Office*.[1]

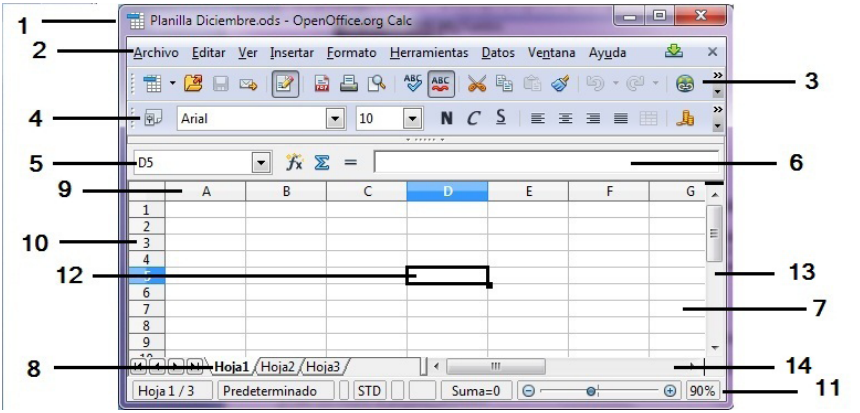
Permite realizar operaciones repetitivas de recálculo y obtener resultados tanto numéricos como gráficos, organizados en una cuadrícula o tabla. Es útil para realizar desde simples sumas, hasta cálculos complejos, por ejemplo la simulación de modo instantáneo de la evolución de un sistema físico.

En general, las planillas de cálculo dan un excelente apoyo en el diseño experimental, el cálculo y el análisis de los resultados en tareas de investigación. Pueden integrar fácilmente fórmulas, imágenes y gráficas.

Cuando en la planilla de cálculo se modifica cualquier parámetro de entrada del cómputo, directamente se obtiene el resultado actualizado con el cambio.

A modo de resumen, el presente capítulo tiene como objetivo introducir algunos conceptos para el tratamiento de datos con la planilla de cálculo Calc, y ofrecer un pantallazo de su poder por medio de algunas de sus herramientas más útiles.

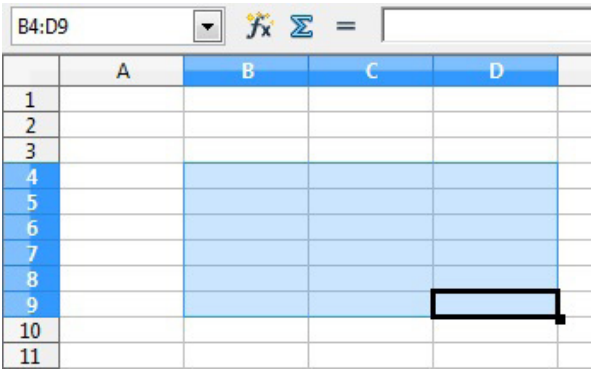
Primero mostraremos el entorno gráfico de Calc con algunas de las referencias:



1 LibreOffice es un paquete completo de productividad de calidad profesional libre y de código abierto que se puede descargar e instalar de forma gratuita. Está disponible en más de treinta idiomas y para todos los principales sistemas operativos. Además de tener la planilla de cálculo Calc, incluye el procesador de texto Writer, base de datos Base, editor de presentaciones multimedia Impress, editor de gráficos y diagramas Draw y un editor de ecuaciones matemáticas, Math.

Referencias

1. **Barra de título:** contiene el nombre del documento sobre el que se está trabajando en ese momento. Cuando creamos un libro nuevo se le asigna el nombre provisional «Sin Título 1», hasta que lo guardamos y le damos el nombre y la ubicación que queramos. En el extremo de la derecha están los botones para minimizar, (restaurar o maximizar) y cerrar.
2. **La barra de menús:** contiene todas las operaciones del Calc, agrupadas en menús desplegables. Al hacer clic en Insertar, por ejemplo, veremos las operaciones relacionadas con los diferentes elementos que pueden insertarse en la planilla.
3. **Barra de herramientas:** contiene botones en forma de íconos. Se trata de las operaciones más habituales y que se realizan más rápidamente.
4. **Barra de formato:** contiene las operaciones más comunes sobre formatos, cómo poner en negrita, cursiva, elegir tipo de fuente, tamaño de fuente, etcétera.
5. **Celda activa:** muestra la posición de la celda activa, en nuestro ejemplo la coordenada es D5, pero cuando seleccionamos un grupo de celdas como en el siguiente ejemplo:



La celda muestra los extremos (superior izquierdo : inferior derecho) →B4:D9, llamado rango de celdas.

6. **Barra de fórmulas**: muestra el contenido de la celda activa, es decir, la casilla donde estamos situados. Es muy útil para ver y editar fórmulas que no son imprimibles, pero sí su resultado.

7. **Planilla:** conformada por filas y columnas. La pantalla solo muestra una porción de esta.
8. **Barra de etiquetas:** permite moverse por las distintas hojas, también podemos agregar, eliminar y/o cambiar el nombre de la etiqueta. Por defecto aparece como «Hoja #», y # es el número de hoja a medida que se van insertando.
9. **Encabezado de columnas:** conformado por las letras en orden alfabético son los encabezados de las columnas de la planilla. Cuando llega a "Z" sigue con la doble a "AA", "AB","AC",...,"ZZ" y luego sigue con "AAA", "AAB",..., hasta llegar al límite "AMJ" conformando un total de 1024 columnas.
10. **Encabezado de filas:** conformado por los números en orden ascendente, son los encabezados de las filas de la planilla. La cantidad total es de 65 536 filas.
11. **Tamaño visual:** para visualizar en distintas escalas la planilla. Por defecto, siempre empieza en el 100 %.
12. **Celda:** es la intersección de la columna y la fila de la planilla. Puede contener números, caracteres alfanuméricos y fórmulas. Además tiene la posibilidad de cambiar el formato y la máscara.
13. **Barra de desplazamiento vertical:** simplemente sirve para desplazarse arriba y abajo en la planilla.
14. **Barra de desplazamiento horizontal:** sirve para el desplazamiento lateral de la planilla.

Funciones del Calc

En la planilla pueden hacerse las cuatro operaciones aritméticas básicas (suma +, resta –, división / y producto *), pero además dispone de un conjunto de funciones para cálculos más complejos y especializados, como:

- Funciones matemáticas: trigonometría, logaritmos, exponenciales, hiperbólicas, etcétera.
- Funciones lógicas: incluyen los operadores booleanos Y O NO, los de comparación, bifurcación, etcétera.
- Funciones financieras: para realizar cálculos de negocios como capitalización, amortizaciones, etcétera.
- Funciones de base de datos: utilizadas para extraer información.
- Funciones recursivas: para cálculo de iteraciones sucesivas.

- Funciones estadísticas: permiten el completo análisis estadístico de datos.
- Funciones matriz: muy útiles para métodos numéricos, permiten realizar operaciones entre matrices.
- Funciones de texto: para manipular textos y devuelven texto.

Una función es una fórmula predefinida por Calc que opera sobre uno o más valores y devuelve un resultado que aparecerá directamente en la celda introducida.

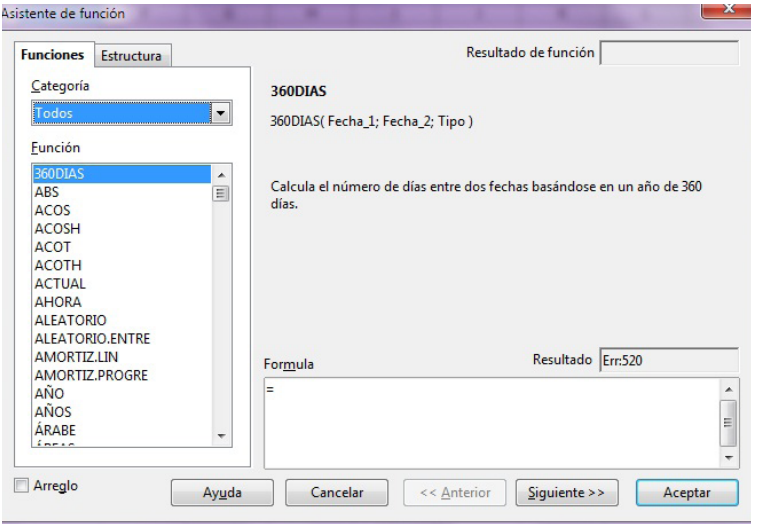
La sintaxis general de la función es:

```
=nombre_función(parámetro1;parámetro2;...;parámetroN)
```

Existen dos maneras diferentes para insertar una función: *a)* por medio de un asistente; *b)* manual.

a. Asistente

Si nos ubicamos en la celda correspondiente y hacemos un clic en «Insertar» de la barra del menú, aparecerá la siguiente ventana:



Aquí se encuentran todas las funciones que posee el Calc, que depura incluso por categoría, ya sean funciones de matemática, finanzas, estadística, matriz, texto, lógico, etcétera.

Este asistente es útil cuando no recordamos el nombre de una función, o bien para buscar una que nunca hallamos utilizado.

Describe la utilidad de cada una de las funciones.

b. Manual

Mientras se recuerde la función este modo, es más directo y rápido. Simplemente se escribe en la celda donde queremos reproducir la función.

Reglas para las funciones cuando se inserta manualmente

1. Si la función va al comienzo de una fórmula debe empezar por el signo =.
2. Los parámetros o valores de entrada van siempre entre paréntesis. No deben dejarse espacios antes o después de cada paréntesis.
3. Los parámetros pueden ser valores constantes (número o texto), fórmulas o funciones.
4. Los parámetros deben estar separados por ";".
5. Los paréntesis deben ser equilibrados, es decir la misma cantidad de apertura "(" como de cierre ")".
6. Nunca hacer referencia a la propia celda. Por ejemplo, si estamos en F5 no podemos referirnos a dicha celda, ya que entraríamos en un problema cíclico.

Atención: la función puede escribirse en minúscula o mayúscula. Una función puede contener otra función. Para desarrollar operaciones aritméticas se usan +,-,/,* (suma, resta, división y multiplicación) y para la potencia, ^. Por ejemplo: =A3^2 (el contenido de la celda A3 se eleva al cuadrado), para la raíz cuadrada =A3^(1/2).

Algunas funciones predefinidas

Antes de definir las funciones recordemos que un rango de celdas está conformado por un grupo de celdas al que llamaremos «Área». Los parámetros que se encuentran dentro de las funciones se separan con ";".

SUMA

Sintaxis: =SUMA("Área")

Descripción: obtiene la sumatoria del contenido del rango de celdas.
Ejemplo: =SUMA(A1:A20) → Obtiene la sumatoria desde la celda A1 hasta la A20. Esto sería equivalente a sumar A1+A2+A3+...+A19+A20
Si entre el rango se encuentran celdas vacías se las toma como cero.

PRODUCTO

Sintaxis: =PRODUCTO("Área")
Descripción: obtiene el producto entre todos los valores involucrados en el rango.
Ejemplo: =PRODUCTO(A1:A20) → Obtiene el producto desde la celda A1 hasta la A20. Esto sería equivalente a (A1*A2*A3*...*A19*A20)
Si entre el rango se encuentran celdas vacías, se las toma como cero.

PROMEDIO

Sintaxis: =PROMEDIO("Área")
Descripción: obtiene el promedio o media aritmética del contenido del rango de celdas.
Ejemplo: =PROMEDIO(A1:A20) → Obtiene el promedio desde la celda A1 hasta la A20. Esto sería equivalente a (A1+A2+A3+...+A19+A20)/20
Si entre el rango se encuentran celdas vacías, se las toma como cero.

MÁX

Sintaxis: =MÁX("Área") →No olvidar la tilde en la "A" de Máximo
Descripción: devuelve el valor máximo de un conjunto de valores.
Ejemplo: =MÁX(A1:A20)

MÍN

Sintaxis: =MÍN("Área") →No olvidar la tilde en la "I" de Mínimo
Descripción: devuelve el valor mínimo de un conjunto de valores.
Ejemplo: =MÍN(A1:A20)

FACT

Sintaxis: =FACT("celda")
Descripción: obtiene el factorial de un número.
Ejemplo: =FACT(A7) → Obtiene el factorial del número que se encuentra en la celda A7, si en dicha celda hay un 4 (por ejemplo) entonces será equivalente a multiplicar 4*3*2*1 obteniendo 24.
Si la celda contiene un cero o esta vacía, FACT devuelve un 1.

CONTAR

Sintaxis: =CONTAR("Área")
Descripción: devuelve la cantidad de números del rango.
Ejemplo: =CONTAR(A1:A20) → Obtiene la cantidad de números del rango, obviando caracteres alfanuméricos y celdas vacías.

CONTARA

Sintaxis: =CONTARA("Área")
Descripción: devuelve la cantidad de datos no vacíos del rango.
Ejemplo: =CONTAR(A1:A20) → Obtiene la cantidad en el rango especificado de números, texto, fechas, formulas, etcétera (pero no incluye las celdas vacías).

CONTAR.BLANCO

Sintaxis: =CONTAR.BLANCO("Área")
Descripción: devuelve la cantidad de datos vacíos del rango.
Ejemplo: =CONTAR(A1:A20) → Obtiene la cantidad en el rango especificado de celdas vacías.

CONTAR.SI

Sintaxis: =CONTAR.SI("Área";"Criterios")
Descripción: cuenta las celdas dentro del rango que no están en blanco y cumplen con el criterio especificado.
Ejemplo: =CONTAR.SI(A1:A20;">30") →Se obtiene la cantidad de números mayores a 30 del rango especificado. El criterio debe estar entre comillas.
Otro Ejemplo: =CONTAR.SI(A1:A20;"=Argentina") →Se obtiene la cantidad de celdas que contiene la palabra «Argentina». La función no diferencia mayúsculas de minúsculas.

SUMAR.SI

Sintaxis: =SUMAR.SI("Área";"Criterios"; "Área de suma")
Descripción: realiza la sumatoria del "Área de suma", según el "criterio" que debe cumplir en el "Área".
Ejemplo: =SUMAR.SI(A1:A20;"<=10";B1:B20) →Se obtiene la sumatoria de los números que se encuentran en el rango B1:B20, filtrando solo los que cumplen el criterio «menores o iguales a 10», según el rango A1:A20.

HOY

Sintaxis: =HOY()

Descripción: con esta función sin parámetros se obtiene la fecha actual, dd/mm/aa.

AHORA

Sintaxis: =AHORA()
Descripción: devuelve la fecha y hora actual dd/mm/aa hh:mm.
Al igual que la función HOY no tiene parámetros.

DÍA

Sintaxis: =DÍA("celda")
Descripción: se obtiene el número del día de una celda con una fecha.
Ejemplo: =DÍA (D3) →Si en la celda D3 tenemos la fecha «11/07/14», entonces el resultado será 11.

MES

Sintaxis: =MES("celda")
Descripción: se obtiene el número del mes de una celda con una fecha.
Ejemplo: =MES(D3) →Si en la celda D3 tenemos la fecha «11/07/14», entonces el resultado será 7.

AÑO

Sintaxis: =AÑO("celda")
Descripción: se obtiene el número del año de una celda con una fecha.
Ejemplo: =AÑO(D3) →Si en la celda D3 tenemos la fecha «11/07/14», entonces el resultado será 2014.

DIASEM

Sintaxis: =DÍASEM("celda")
Descripción: se obtiene el número del día de la semana, siendo Domingo=1, Lunes=2,..,Sabado=7.
Ejemplo: =DÍASEM(D3)→ Si en la celda D3 tenemos la fecha «11/07/14», entonces el resultado será 6 porque se trata del día viernes.

HORA

Sintaxis: =HORA("celda")
Descripción: Se obtiene la hora hh siempre y cuando la celda contenga un horario.
Ejemplo: =HORA(D3) →Si la celda tiene «11/07/14 01:30» o «01:30», el resultado será 01.

PI

Sintaxis: =PI()→Sin paramentaros, devuelve el valor de π.
La precisión de PI va a depender del formato que le demos en cantidad de decimales.

COS, SENO, TAN

Sintaxis: =COS("celda")→En este caso la celda debe ser en radianes, por ejemplo =COS(PI()/2) o si queremos usar grados entonces podemos utilizar la función **RADIANES** de la siguiente manera: =COS(RADIANES(90)).
Lo mismo sucede con =SENO("celda") y =TAN("celda") para hallar el seno y tangente respectivamente.

MINÚSC

Sintaxis: =MINÚSC("celda") → Convierte el texto de la celda a minúscula.
Ejemplo: =MINÚSC(A6) →Si el contenido de A6 es «PErrO», la función devuelve «perro».

MAYÚSC

Sintaxis: =MAYÚSC("celda") → Convierte el texto de la celda a mayúscula.
Ejemplo: =MAYÚSC(A6) →Si el contenido de A6 es «peRRo», la función devuelve «PERRO».

LARGO

Sintaxis: =LARGO("celda") → Devuelve la cantidad de caracteres que contiene la celda (incluidos los espacios en blanco).
Ejemplo: =LARGO(B9)→ Si B9 contiene el siguiente texto: «El perro ladra», la función devuelve 14 (tambien cuenta la cantidad de espacios que aparecen a la derecha del texto y/o a la izquierda). Por ejemplo: si el texto de la celda es " El perro ladra " el resultado será 16.

IZQUIERDA

Sintaxis: =IZQUIERDA("celda";número) → Devuelve el primero o primeros caracteres del texto que se encuentra en la celda, según lo indique el número.
Ejemplo: =IZQUIERDA(F3;2)→Si en la celda F3 se encuentra el texto «Montaña» el resultado será «Mo».

DERECHA

Sintaxis: =DERECHA("celda";número) → Devuelve el último o últimos caracteres del texto que se encuentra en la celda, según lo indique el número.
Ejemplo: =DERECHA(F3;2)→Si en la celda F3 se encuentra el texto «Montaña» el resultado será «ña».

CONCATENAR

Sintaxis:=CONCATENAR("argumento1";"argumento2";..;"argumentoN") →
Ejemplo: =CONCATENAR(A1;A2;A3) → A1="El" A2="perro" A3="ladra" entonces la función devuelve: "Elperroladra". Para mejorar el texto deberíamos agregar más argumentos con espacios, de la siguiente manera: = CONCATENAR(A1;" ";A2;" ";A3) → Obteniendo «El perro ladra».

SI

Sintaxis: =SI("Prueba lógica"; "Valor verdadero"; "Valor falso")
Descripción: esta función devuelve el valor verdadero si se cumple una determinada condición. Caso contrario, devuelve el valor falso. El tipo de dato devuelto por la función depende de los argumentos asignados. El argumento «Prueba lógica» contiene una expresión que devuelve un valor lógico.
La prueba lógica es cualquier valor o expresión que pueda evaluarse como Verdadero o Falso.
Ejemplo: =SI(C9<0;"NEGATIVO";"POSITIVO") → Si C9 contiene un valor menor a cero entonces se cumple lo del valor verdadero que expresa el cartel «NEGATIVO», sino se cumple (quiere decir que en C9 tiene un valor mayor o igual a cero, entonces se cumple el valor falso que expresa el cartel «POSITIVO».
Otro ejemplo: =SI(B5= "V";"color verde";"otro color") → Acá también se devuelve un dato de tipo texto. Si la celda B5 contiene el texto «V», la función devuelve el texto «color verde». En caso contrario, la función devuelve el texto «otro color».
Otro ejemplo: =SI(M10=1;G5*1,40;G5) → Si la celda M10 contiene el valor 1 entonces se aumenta un 40 % el valor de G5, caso contrario, queda G5 sin modificar.

Nota: se pueden anidar funciones (una dentro de otra), por ejemplo para el caso de saber si un número en positivo, negativo o neutro (cero) debemos utilizar tres posibles respuestas, entonces anidamos 2 funciones SI. La función sería así:
=SI(C9<0; "NEGATIVO";SI(C9=0;"NEUTRO";"POSITIVO"))

Para tres respuestas posibles hace falta usar dos preguntas "SI" porque la última sale por descarte.
Es decir que para N respuestas posibles, se van a utilizar N-1 preguntas «SI».

Funciones lógicas Y, O

Sintaxis: **Y**("condición1";"condición2";...;"condiciónN").
Sintaxis: **O**("condición1";"condición2";...;"condiciónN").

Estas dos funciones se usan cuando tenemos más de una condición en una función «Si». Hasta el momento, solo habíamos especificado una condición dentro del parámetro en la prueba lógica, pero existen situaciones donde necesitamos utilizar más de una.
Ejemplo de Y: =SI(Y(C7>=14;C7<=15);"Cadete";"Otra categoría")→ Si la edad de un muchacho que se encuentra en la celda C7 cumple con las dos condiciones (entre 14 y 15 incluidos) entonces el texto será «Cadete» sino «Otra categoría». Dentro del Y se puede tener hasta 30 condiciones.
Ejemplo de O: =SI(O(C7=14;C7=15);"Cadete";"Otra categoría")→ Si la edad de un muchacho que se encuentra en la celda C7 cumple con una de las condiciones (tiene 14 o 15 años) entonces el texto será «Cadete» sino «Otra categoría». Dentro del Y se pueden tener hasta 30 condiciones.
En estos casos ambos ejemplos cumplen con la misma consigna.

Atención: al utilizar cualquier función de tipo fecha, el formato de la celda cambia automáticamente a formato fecha, con la posibilidad de modificar la máscara.

Referencias relativas y absolutas

Una referencia indica dónde buscar los valores que desean utilizarse en la fórmula o la función.

Existen dos clases de referencias: las relativas y las absolutas.

Las referencias relativas

Por lo general, cuando se crea una fórmula, las referencias a las celdas se basan en su posición relativa a la celda que la contiene. Esta característica es

muy útil cuando se copia una fórmula a otras celdas, ya que las direcciones de las celdas se actualizan de acuerdo a su nueva posición.

Por ejemplo:

Tenemos la siguiente planilla:

	A	B	C	D	E	F
1						
2						
3		Codigo de producto	Precio unitario	Cantidad vendida	Importe	
4		A001	$12,50	12		
5		A003	$78,00	1		
6		B010	$91,50	10		
7		B012	$33,00	19		
8		C107	$49,75	4		
9		A088	$0,50	7		
10		B099	$14,00	2		
11		C111	$30,00	1		
12		B049	$22,00	90		
13						
14						

Y queremos hallar el importe de cada venta según el precio de cada producto y la cantidad vendida, entonces debemos empezar por resolver en la celda E4 con la siguiente fórmula:

=C4*D4. Luego copiamos hacia abajo hasta E12 usando el arrastre del mouse donde indica la flecha de la siguiente imagen:

Entonces obtendremos todos los importes necesarios gracias a la referencia relativa. En la celda E5 tendremos → =C5*D5 en E6→ =C6*D6 y así hasta E12 → =C12*D12.

Observe que al copiar la fórmula original hacia abajo, las direcciones de las celdas se actualizan automáticamente.

Referencias absolutas

No siempre vamos a necesitar que las direcciones de las celdas cambien, entonces es necesario fijar las direcciones.

Por ejemplo, si queremos hallar el valor en dólares en la columna F de cada importe de venta sabiendo que en la celda C1 tenemos el valor actualizado, entonces:

	A	B	C	D	E	F
1		Valor del dólar	$8,50			
2						
3		**Codigo de producto**	**Precio unitario**	**Cantidad vendida**	**Importe**	**U$s**
4		A001	$12,50	12	$150,00	
5		A003	$78,00	1	$78,00	
6		B010	$91,50	10	$915,00	
7		B012	$33,00	19	$627,00	
8		C107	$49,75	4	$199,00	
9		A088	$0,50	7	$3,50	
10		B099	$14,00	2	$28,00	
11		C111	$30,00	1	$30,00	
12		B049	$22,00	90	$1.980,00	
13						

Comenzamos por hacer la primera fórmula ubicada en la celda F4 con el importe del primer producto dividido el valor del dólar: =E4/C1.

Pero si a este lo copiamos hacia abajo como en el ejemplo anterior, obtendremos valores absurdos porque en estos casos la celda C1 de la fórmula deberá mantenerse fija, es decir, absoluta.

Para fijar una dirección debemos situarnos entre la letra y el número de la celda a fijar, en nuestro caso «C1» y luego presionamos del teclado «SHIFT+F4», aparecerá unos signos $ de la siguiente manera: =E4/$C$1 y luego sí podremos copiar la fórmula.

Los signos $ en una dirección de celda significa que se va a mantener fija en el copiado.

Finalmente, obtendremos en la columna F las siguientes fórmulas:

En F4→=E4/$C$1, en F5→=E5/$C$1, y así hasta F12→=E12/$C$1.

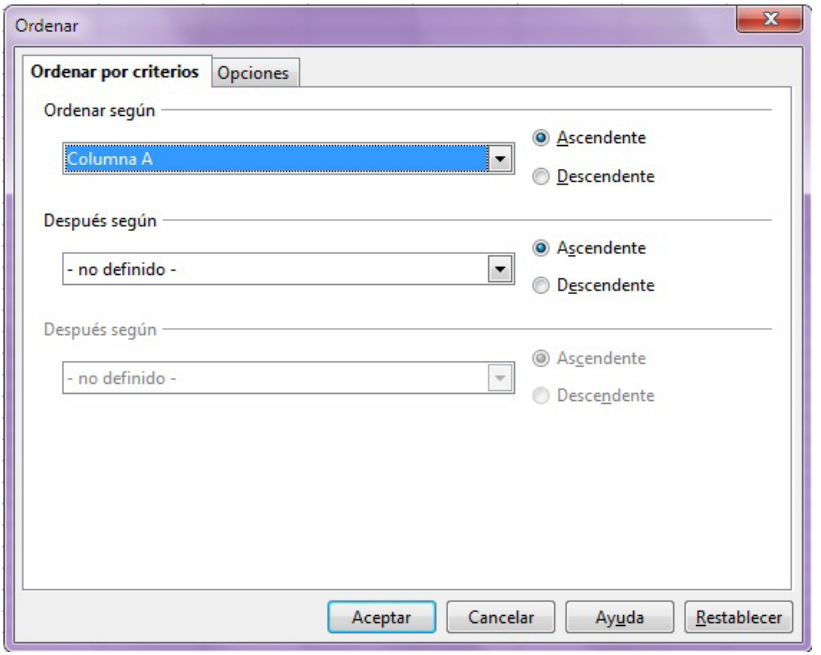
Ordenar datos en una planilla

Para ordenar valores en Calc, simplemente luego de seleccionar los datos a ordenar, vamos al menú Datos y cliqueamos Ordenar. En el cuadro de diálogo que aparece a continuación, seleccionamos las opciones a nuestro gusto y luego cliqueamos Aceptar.

Para ordenar rápidamente en forma ascendente o descendente varios datos, los seleccionamos y luego cliqueamos los botones Orden ascendente u Orden descendente. Éstos se encuentran en la barra de herramientas Estándar.

En la selección previa a la ventana Ordenar, incluimos los títulos de cabecera.

Solo se pueden tener hasta tres campos en el criterio de ordenamiento, es decir que podemos ordenar una lista, por ejemplo, por Apellido, Nombre y DNI.



BUSCARV

Una vez visto el ordenamiento y las direcciones absolutas, podemos explicar de qué se trata esta poderosa función muy utilizada.

Buscar un valor específico en la columna clave de una matriz y devolver el valor de otra tabla llamada «matriz de referencia».

Sintaxis: =BUSCARV(valor buscado; matriz de referencia; indicador de columnas).

A continuación se detallan los tres parámetros de la función:

- **Valor buscado** es el valor que se busca en la columna clave de la matriz, puede ser un valor numérico, una referencia o una cadena de texto.
- **Matriz de referencia** es el conjunto de información donde se buscan los datos. La matriz debe estar ubicada en otro lugar (fuera de la tabla principal) y debe tener en común un dato que se encuentra en

ambas tablas. Además, la tabla de referencia debe estar ordenada de modo ascendente según la clave.

- **Indicador de columnas** es el número de columna de la matriz de referencia desde la cual debe devolverse el valor coincidente. Si el argumento es igual a 1, la función devuelve el valor de la primera columna del argumento (no tiene sentido porque se repetiría el valor de la clave); si el argumento es igual a 2, devuelve el valor de la segunda columna de la matriz de referencia.

Observaciones: si BUSCARV no puede encontrar el «valor buscado», utiliza el valor más grande que sea menor o igual a dicho valor.
También existe la función BUSCARH que procede de la misma forma pero la diferencia está en que sirve para matrices cuyas claves y datos se encuentran por fila y no por columnas.

Ejemplo 1:
Supongamos que nuestra matriz de referencia se encuentra en el rango B20:D26, es decir que tiene tres columnas (la primera contiene la clave y las otras dos los datos relacionados):

Legajo Alumno	Carrera	Plan de estudio
1001	Ingeniería Ambiental	2009
1023	Ingeniería Electrónica	2010
1100	Ingeniería Ambiental	2010
1207	Licenciatura en Informática	2012
1208	Ingeniería en Telecomunicaciones	2009
1210	Ingeniería Ambiental	2009

De la planilla principal necesitamos obtener la carrera según el número de legajo del estudiante, entonces utilizamos: =BUSCARV(B2; $B$21:$B$26; 2) →la función nos devuelve la carrera y si escribíamos un 3 en lugar del 2 (indicador de columnas) →nos devuelve el plan de estudio.

Nota: se supone que en B2 existe el número de legajo del alumno/a que es el dato en común con la matriz de referencia.

El segundo parámetro del BUSCARV, la matriz de referencia debe mantenerse fija con las direcciones absolutas $B$21:$B$26.

> **Atención:** no debe involucrar en el segundo parámetro de la función la cabecera de la tabla de referencia, es decir obviar los títulos (Legajo Alumno-Carrera-Plan de estudio).

Crear gráficos

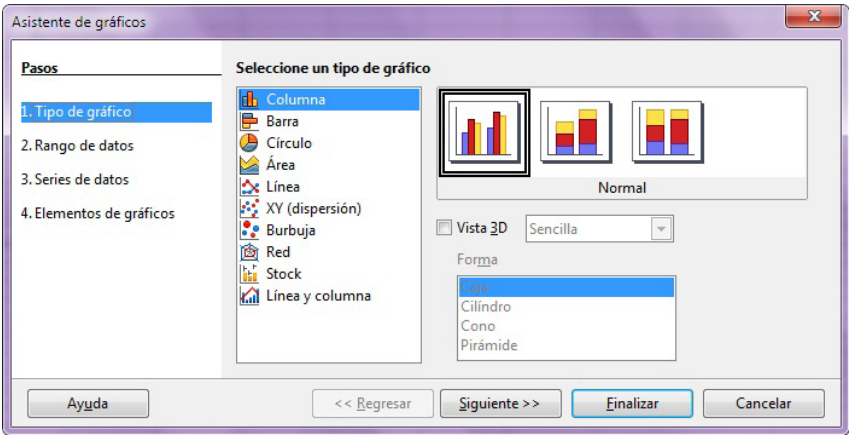
Veremos cómo crear gráficos a partir de los datos introducidos en una hoja de cálculo. Así resultará más sencilla la interpretación de los datos.

Un gráfico es la representación de los datos de una hoja de cálculo que facilita su interpretación.

A la hora de crear un gráfico, Calc dispone de un asistente que nos guiará en su creación, de forma que resulte más fácil.

Los pasos a seguir para crear un gráfico son los siguientes:

1. Seleccione los datos a representar en el gráfico.
2. Seleccione el menú Insertar.
3. Elija la opción Gráfico, entonces aparecerá el siguiente asistente



4. Elija un tipo de gráfico.
5. Una vez elegido el tipo de gráfico, en el recuadro de la derecha, elija un subtipo.
6. Si pulsa sobre el botón Presionar para ver muestra y lo mantiene pulsado, aparece en lugar de.

7. En todos los pasos del asistente se dispone de los botones Regresar, Siguiente, Finalizar y Cancelar, en la parte inferior del cuadro de diálogo.
8. Luego de varios pasos Siguiente, llegaremos a Finalizar para terminar con la gráfica.

Ejercicios: Planilla de Cálculo - Calc

1. La siguiente tabla muestra el movimiento de los clientes de un Spa:

	A	B	C	D	E	F	G	H	I	J	K	L
1	SPA - Rejuvenecer											
2	Ingresos Abril								Real	3,68		
3					Actividades							
4	Socio Nro	Localidad	Deuda anterior	Edad	Masaje	Sauna	Jacuzzi	Cuota base	Total a pagar	Total Real	Indicador	Nuevo Codigo
5	1213	Bariloche	$ 350,00	34	s	s						
6	6214	Viedma	$ 615,00	39	s		s					
7	2217	El Bolson		43		s	s					
8	1219	Bariloche		54			s					
9	3221	Pilcaniyeu	$ 325,00	23	s	s	s					
10	6222	Bariloche	$ 300,00	19	s	s						
11	8234	Villa Angostura		32			s					
12	1235	Bariloche		26	s	s	s					
13	1237	Esquel	$ 195,00	28		s						
14	7243	Bariloche		32	s	s						
15	5254	Jacobacci	$ 220,00	42		s	s					
16							Total recaudado:					
17												
18							Cantidad de socios:					
19							Cantidad de socios que realizan las tres actividades:					
20							Cantidad de socios sin deuda:					
21							Porcentaje de socios que eligen Sauna:					
22							Porcentaje de socios menores de 35 años:					
23							Edad promedio de los clientes:					
24							Importe total de los socios que eligieron Jacuzzi:					
25							Importe total de los socios de Bariloche:					
26							Menor Total a pagar:					
27							Mayor Deuda:					

Realizar las siguientes consignas:

1. Vuelque exactamente los datos de la figura anterior a una planilla (respete formatos y ubicación) y guarde el archivo con el nombre «SPA Río Negro».
2. Cambie la etiqueta «Hoja 1» por «Ingresos Abril».
3. La «cuota base» se calculará teniendo en cuenta lo siguiente:
   a. si la edad del cliente es menor a 22 años, se cobrarán $ 135;
   b. si la edad del cliente es menor a 35 años y mayor o igual a 22, se cobrarán $ 240;
   c. si la edad del cliente es mayor o igual a 35 años, se cobrarán $ 300;
   d. si elige sauna se sumarán $ 58 (sin importar la edad);
   e. si elige jacuzzi se sumarán $ 90 (sin importar la edad);
   f. si elige masaje se sumarán $ 70 (sin importar la edad).

4. El «total a pagar» se calculará realizando un 10% de descuento a la cuota base si realiza las tres actividades y un 5% si realiza dos actividades, pero si tiene deuda no se le realiza ningún descuento.
5. En «total real» se debe agregar el valor total en moneda brasilera Real, cuyo valor se encuentra en la celda J2.
6. En el indicador deberá aparecer la palabra «CON DEUDA» si posee deuda.
7. En Nuevo Código, cree para cada socio la nueva identificación que consiste en las dos primeras letras de la localidad en mayúscula y el número de socio separado por un guion. Por ejemplo: BA-1213. Complete los datos de H18 a H27 (usar las fórmulas).
8. Ordene la planilla según la localidad y luego por el n.º de socio.
9. Realice el gráfico de barras de los socios según el total a pagar.

2. La siguiente tabla muestra la clasificación de los países más contaminantes del mundo, medidos en toneladas de CO_2 per cápita:

País	Población	2008	2009	2010
Arabia	27 345 986	16,6	16,1	17
Aruba	110 663	21,7	21,5	22,8
Australia	22 507 617	18,6	18,2	16,9
Bahrein	1 314 089	21,4	20,7	19,3
Brunei	422 675	27,5	23,7	22,9
Canadá	34 834 841	16,3	15,2	14,6
Emiratos Unidos	5 628 805	25	22,6	19,9
Kazajistán	17 948 816	15,2	14	15,2
Kuwait	2 742 711	30,1	30,3	31,3
Luxemburgo	520 672	21,5	20,4	21,4
Omán	3 219 775	17,3	15,2	20,4
Qatar	2 123 160	49,1	44	40,3
Trinidad y Tobago	1 223 916	37,4	35,8	38,2
USA	318 892 103	17,9	17,3	17,6

Realizar las siguientes consignas:

1. Vuelque exactamente los datos de la tabla anterior a una planilla a partir de la celda B4 y guarde el archivo con el nombre «Contaminacion Mundial».
2. Agregue el título «Los países más contaminantes del mundo 2008 al 2010» en B1.
3. Ordénela de modo descendente según la población.
4. Al pie de la tabla (en B19) agregue la etiqueta «Total» y halle solo la sumatoria de la población en C19.
5. Cambie el formato de los índices de contaminación de modo que queden todos con dos dígitos decimales.
6. En K4 y K5 agregue las etiquetas «Máximo» y «Mínimo», respetivamente, para mostrar los resultados a su derecha, es decir L4 y L5 (dos resultados para todos los años).
7. En B22 agregue la siguiente tabla de continentes y sus respectivos centros de convenciones:

África	Casablanca
América	Caracas
Asia	Tokio
Europa	Oslo
Oceanía	Camberra

8. En G4 agregue la cabecera «Continente» y en H4 «Ciudad». Debajo del primero agregue uno por uno el continente correspondiente, respetando la sintaxis «acentos, mayúsculas, etc.».
9. En la columna H agregue cada ciudad según el continente usando la función BUSCARV.
10. En la columna I agregue el porcentaje (%) de cada país según la población. Tome en cuenta el total que se ubica en la celda C19.
11. Realice un gráfico circular para cada año de contaminación y edítelo cambiando el aspecto a gusto.

Bibliografía

Augenstein, M. J., Langsam, Y. y Tenenbaum, A. (1993). Estructuras de datos en C. México: Prentice-Hall Hispanoamericana.

Cataldi, Z., Lage, F., y Salgueiro, F. (2001). Fundamentos de Algoritmos y Programación I. Buenos Aires: Editorial Nueva Librería.

Cormen, T. H., Leiserson, C. E., Rivest, R. y Stein, C. (1990). Introducción a los Algoritmos. Cambridge: Mit Press.

De Giusti, A. (2001). Algoritmos, Datos y Programas. Buenos Aires: Pearson Education.

Joyanes Aguilar, L. y Zahonero Martínez, I. (2005). Programación en C. Metodología, Algoritmos y Estructuras de Datos. Madrid: McGraw-Hill.

Kernighan, B. W. y Ritchie, D. M. (1985). The C Programming Language. Editorial: Prentice Hall.

Caminando junto al lenguaje C
Martín Goin
Segunda edición. Viedma : Universidad Nacional de Río Negro, 2022.
Libro digital, PDF - (Lecturas de cátedra)
Archivo digital: descarga y online

Lecturas de Cátedra

ISBN 978-987-4960-75-7

1. Lenguaje de programación. I. Título
CDD 005.133

www.editorial.unrn.edu.ar
Belgrano 526, Viedma, Río Negro, Argentina.

Coordinación editorial: Ignacio Artola
Edición de textos: Natalia Barrio
Corrección: Cecilia Soto
Diagramación y diseño: Sergio Campozano
Imagen de tapa: Editorial UNRN

Caminando junto al lenguaje C
fue compuesto con la familia tipográfica Alegreya ht Pro,
en sus múltiples variables, Liberation Sans Narrow y Source Code Pro.

Se editó en marzo de 2022 en la Dirección de Publicaciones-Editorial
de la Universidad Nacional de Río Negro.

Caminando junto al lenguaje C

Con esta publicación se busca que los estudiantes cuenten con una herramienta sencilla para aprender y manejar, de modo progresivo, un lenguaje de programación. El objetivo principal es dar a conocer el lenguaje C, muy utilizado en las cátedras universitarias.
Se pone atención a los aspectos prácticos del uso de algoritmos, sin descuidar los conceptos teóricos de cada tema.

El libro, que expone numerosos ejemplos e intenta ayudar y acompañar al lector a resolver los problemas planteados, es ideal para aquellos estudiantes que incursionan por primera vez en el mundo de la programación. De hecho, está orientado a los primeros cursos de grado de carreras como ingeniería, profesorados, licenciaturas, tecnicaturas y otras que incluyan materias de programación.

El material funciona como un tutorial que explica paso a paso las bases de la programación. Se divide en seis capítulos, cada uno –salvo el primero– contiene problemas para resolver. En total se plantean 186 ejercicios y 106 ejemplos prácticos. El último capítulo es un obsequio, donde se presenta una guía rápida del programa Calc (planilla de cálculo), muy útil para la formación universitaria.

L U A
EDITORIAL UNRN
Universidad Nacional de Río Negro
9 789874 960757