INTRODUCCIÓN A LA INFORMÁTICA

PARTE II

ALGORITMOS DE PROGRAMACIÓN

ASIGNATURA	ESPECIALIDAD
FUNDAMENTOS DE INFORMÁTICA	**CIVIL**
	ELÉCTRICA
	MECÁNICA
	QUÍMICA
INFORMÁTICA I	**ELECTRÓNICA**
INFORMÁTICA I	**INDUSTRIAL**

AÑO 2022

www.utnfravirtual.org.ar
dslavutsky@fra.utn.edu.ar

ÍNDICE GENERAL

ALGORITMOS DE PROGRAMACIÓN

INTRODUCCIÓN

La idea general de este capítulo es introducir a los alumnos a la resolución de Algoritmos por medio del uso del Lenguaje C de Programación.

Cada alumno deberá estar capacitado a resolver problemas desde el punto vista de la lógica informática, que le permitirá entender de una manera más explícita la forma de comunicarse con una computadora mediante la creación de programas sencillos que no son ni más ni menos el inicio en esta etapa del conocimiento que le permitirá entender el modo de funcionamiento de un sistema de computación.

Por este motivo el desarrollo de algoritmos es un tema fundamental en el diseño de programas por lo cual el alumno debe tener buenas bases que le sirvan para poder desarrollar de manera fácil y rápida sus programas.

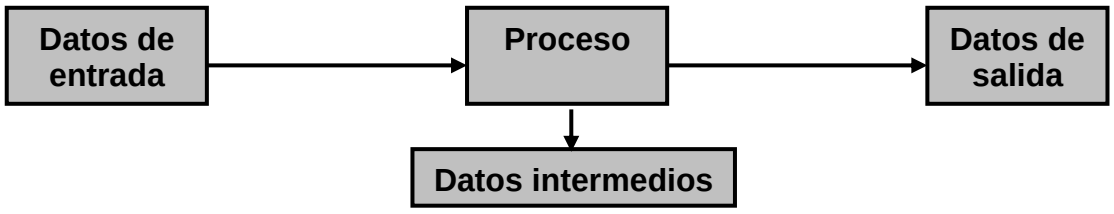
Una computadora no solamente es una máquina que puede realizar procesos para darnos resultados, sin que tengamos la noción exacta de las operaciones que realiza para llegar a esos resultados.

Con la computadora además de lo anterior también podemos diseñar soluciones a la medida, de problemas específicos que se nos presenten. Más aún, si estos involucran operaciones matemáticas complejas y/o repetitivas, o requieren del manejo de un volumen muy grande de datos.

El diseño de soluciones a la medida de nuestros problemas requiere como en otras disciplinas una metodología que nos enseñe de manera gradual, la forma de llegar a estas soluciones. A las soluciones creadas por computadora se les conoce como programas y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, con un grupo de datos específicos.

Lo anterior nos lleva al razonamiento de que un programa nos sirve para solucionar un problema específico. Para poder realizar programas, además de conocer la metodología mencionada, también debemos de conocer, de manera específica las funciones que pueden realizar las computadoras y las formas en que se pueden manejar los elementos que hay en la misma.

Proceso de información en la computadora:



DEFINICIONES IMPORTANTES

Programa: Es el conjunto de instrucciones escritas de algún lenguaje de programación y que, ejecutadas secuencialmente, resuelven un problema específico.

Código Fuente: Se llama al juego de instrucciones que formen parte del lenguaje de programación que estemos utilizando para la escritura del programa.

Lenguaje: Es una serie de símbolos que sirven para transmitir uno o más mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como **comunicación**.

Comunicación: Es un proceso complejo que requiere una serie de reglas simples, pero indispensables para poderse llevar a cabo. Las dos principales son las siguientes:

- Los mensajes deben correr en un sentido a la vez.
- Debe forzosamente existir 4 elementos: Emisor, Receptor, Medio de Comunicación y Mensaje.

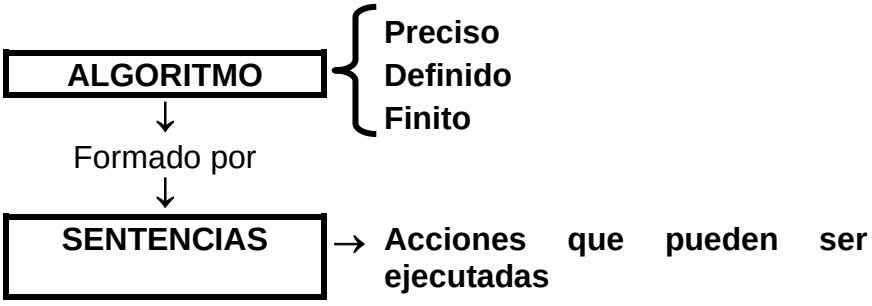
Algoritmo: La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.[1]

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

Es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

- **Preciso:** No se presta a interpretaciones ambiguas.
- **Definido:** Si se siguen 2 o más veces los pasos, se obtiene el mismo resultado.
- **Finito:** Tiene comienzo y fin; tiene un número determinado de pasos.

Los algoritmos se pueden expresar en forma de **diagramas de flujo**, por **fórmulas matemáticas** y en **pseudocódigo**. Esta última herramienta es la más usada en lenguajes estructurados como el **Lenguaje C**.



[1] https://es.wikipedia.org/wiki/Al-Juarismi

Tipos de algoritmos:

- **Cualitativos:** Son aquellos en los que se describen los pasos utilizando palabras.
- **Cuantitativos:** Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

Diseño de un algoritmo:

- Debe tener un punto particular de inicio.
- Debe ser definido, no debe permitir dobles interpretaciones.
- Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.
- Existen varias herramientas para hacerlo, entre ellas el pseudocódigo.

Pseudocódigo:

Es una mezcla de un lenguaje de programación y el idioma de un país, se utiliza en la programación estructurada para realizar el diseño de un algoritmo. En general, al pseudocódigo se lo puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir en un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un Pseudocódigo:

- Permite representar en forma fácil operaciones repetitivas complejas
- Es muy fácil pasar de pseudocódigo a un programa en algún lenguaje de programación.
- Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

Diagrama de flujo:

Es la representación gráfica de un algoritmo. Para resolverlo se utilizan distintos tipos de figuras juntos con flechas conectoras que permiten establecer la secuencia de un proceso.

LENGUAJES DE PROGRAMACIÓN[2]

Para que la parte física de un sistema de computación (hardware) funcione es necesario utilizar programas (software), los cuales le indican cuál es la tarea que se tiene que hacer. Un lenguaje de programación es el que se utiliza para escribir dichos programas. Posteriormente estos se introducirán en la memoria de la computadora y éste último ejecutará todas las operaciones que se incluyen.

Los lenguajes de programación constan de:

a) Un conjunto finito de símbolos, a partir del cual se define el **léxico** o vocabulario del lenguaje.
b) Un conjunto finito de reglas, la **gramática** del lenguaje, para la construcción de las sentencias "correctas" del lenguaje. (**Sintaxis**).
c) **Semántica**, que asocia un significado (la acción que debe llevarse a cabo) a cada posible construcción del lenguaje.

Así, podemos decir que un lenguaje de programación consta de un conjunto de símbolos y un conjunto de reglas válidas para componerlos, de forma que formen un mensaje con significado para la computadora.

CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

- Según el **sistema de computación**: El **Lenguaje Máquina** es específico para cada máquina. Este lenguaje utiliza como código al sistema binario de numeración que consta de los símbolos "0" y "1".
    - Las órdenes que se dan a un sistema de computación han de ir codificadas en instrucciones, y estas forman los programas. Las instrucciones tienen dos partes diferenciadas: código de operación y código(s) de operando(s).
    - En la primera, se codifica la operación que realiza la instrucción. Este código de operación siempre es único para cada instrucción. En la segunda se indica(n) la(s) dirección(es) de memoria en la que se encuentra él operando, hasta un máximo de tres, sobre el/(los) que se aplicará la operación.
    - Puesto que cada tipo de sistema de computación tiene su código máquina específico, para programar en este lenguaje el programador debe conocer la arquitectura física de la computadora con cierto detalle (registros de la CPU, palabras de memoria, ...).

[2] **SITIOS WEB UTILIZADOS PARA ESTE INFORME:**
https://qbitacora.wordpress.com/2007/09/21/clasificacion-de-lenguajes-de-programacion/
http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n
http://www.alegsaonline.com/art/13.php

- Según el **nivel de abstracción**, o sea, según el grado de cercanía a la máquina:
  - Lenguajes de **bajo nivel**: La programación se realiza teniendo muy en cuenta las características del procesador, por cada instrucción en este de lenguaje existe una instrucción en lenguaje de máquina asociada. Ejemplo: Lenguaje ensamblador (**Assembler**).
  - Lenguajes de **nivel medio**: Permiten un mayor grado de abstracción pero al mismo tiempo mantienen algunas cualidades de los lenguajes de bajo nivel. Ejemplo: El lenguaje C puede realizar operaciones lógicas y de desplazamiento con bits, tratar todos los tipos de datos como lo que son en realidad a bajo nivel (números). Otras de las características que marcan una diferencia con los lenguajes de bajo nivel, es que la mayoría de las instrucciones son funciones, y cada función es en sí un pequeño programa que consta de un juego de instrucciones generalmente de bajo nivel.
  - Lenguajes de **alto nivel**: Más parecidos al lenguaje humano. Manejan conceptos, tipos de datos, etc., de una manera cercana al pensamiento humano ignorando (abstrayéndose) del funcionamiento de la máquina. Ejemplos: Java, Ruby.

- Según el **propósito,** es decir, el tipo de problemas a tratar con ellos:
  - Lenguajes de propósito **general**: Aptos para todo tipo de tareas: Ejemplo: C.
  - Lenguajes de propósito **específico**: Hechos para un objetivo muy concreto. Ejemplo: Csound (para crear ficheros de audio).
  - Lenguajes de **programación de sistemas**: Diseñados para realizar sistemas operativos o drivers. Ejemplo: C.
  - Lenguajes de **script**: Para realizar tareas varias de control y auxiliares. Antiguamente eran los llamados lenguajes de procesamiento por lotes (Batch) o JCL (”Job Control Lenguages”). Se subdividen en varias clases (de shell, de GUI, de programación web, etc.). Ejemplos: bash (shell), Lingo (Macromedia Director), mIRC script, JavaScript (programación web).

- Según la **evolución histórica**: Se va incrementando el nivel de abstracción, pero en la práctica, los de una generación no terminan de sustituir a los de la anterior:
  - Lenguajes de **primera generación**: Código máquina.
  - Lenguajes de **segunda generación**: Lenguajes ensamblador.
  - Lenguajes de **tercera generación**: La mayoría de los lenguajes modernos, diseñados para facilitar la programación a los humanos. Ejemplos: C, Java.
  - Lenguajes de **cuarta generación**: Diseñados con un propósito concreto, o sea, para abordar un tipo concreto de problemas. Ejemplo: NATURAL, Mathematica.
  - Lenguajes de **quinta generación**: La intención es que el programador establezca el qué problema ha de ser resuelto y las condiciones a reunir, y la máquina lo resuelve. Se usan en inteligencia artificial. Ejemplo: Prolog.

- Según la **manera de ejecutarse**:
  - Lenguajes **compilados**: Un programa traductor traduce el código del programa (código fuente) en código máquina (código objeto). Otro programa, el enlazador, unirá los archivos de código objeto del programa principal con los de las bibliotecas para producir el programa ejecutable. Ejemplo: C.
  - Lenguajes **interpretados**: Un programa (intérprete), ejecuta las instrucciones del programa de manera directa. Ejemplo: Lisp.
- Según la **manera de abordar** la tarea a realizar:
  - Lenguajes **imperativos**: En ciencias de la computación se llama **lenguajes imperativos** a aquellos en los cuales se le ordena a la computadora cómo realizar una tarea siguiendo una serie de pasos o instrucciones, por ejemplo:
    - Paso 1, solicitar número.
      Paso 2, multiplicar número por dos.
      Paso 3, imprimir resultado de la operación.
      Paso 4, etc.
    - Algunos ejemplos de lenguajes imperativos son: BASIC, C, C++, Java, Clipper, Dbase, C# y Perl.
  - Lenguajes **declarativos**: Se les conoce como lenguajes declarativos en ciencias computacionales aquellos lenguajes de programación en los cuales se le indica a la computadora que es lo que se desea obtener o que es lo que se está buscando, por ejemplo: Obtener los nombres de todos los empleados que tengan más de 32 años. Algunos ejemplos de lenguajes declarativos son el Datatrieve, SQL y las expresiones regulares.
    - Siglas de **Structured Query Language** (**Lenguaje Estructurado de Consultas**). Es un lenguaje declarativo que aúna características del Álgebra y el Cálculo Relacionales que nos permite lanzar consultas contra una Base de Datos para recuperar información de nuestro interés, almacenada en ella.
    - Ejemplos de consultas SQL:
      - **SELECT** Nombre **FROM** Datos Personales **WHERE** Edad >=18**;**
      - Muestra el Campo "Nombre" de todos los individuos mayores de 18 años de la tabla "Datos Personales"
- Según el **paradigma de programación:** Un paradigma de programación provee (y determina) la visión y métodos de un programador en la construcción de un programa o subprograma. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas (con la solución de múltiples “problemas” se construye una aplicación). Los principales son:
  - Lenguajes de **programación procedural**: Divide el problema en partes más pequeñas, que serán realizadas por subprogramas (subrutinas, funciones, procedimientos), que se llaman unas a otras para ser ejecutadas. Ejemplos: C, Pascal.

- Lenguajes de **programación orientada a objetos**: Crean un sistema de clases y objetos siguiendo el ejemplo del mundo real, en el que unos objetos realizan acciones y se comunican con otros objetos. Ejemplos: C++, Java.
- Lenguajes de **programación funcional**: La tarea se realiza evaluando funciones, (como en Matemáticas), de manera recursiva. Ejemplo: Lisp.
- Lenguajes de **programación lógica**: La tarea a realizar se expresa empleando lógica formal matemática. Expresa qué computar (calcular). Ejemplo: Prolog.

- En sistemas distribuidos, según **dónde se ejecute**:
  - Lenguajes de **servidor**: Se ejecutan en el servidor. Ejemplo: PHP es el más utilizado en servidores web.
  - Lenguajes de **cliente**: Se ejecutan en el cliente. Ejemplo: JavaScript en navegadores web.

- Según **admitan o no concurrencia** de procesos, esto es, la ejecución simultánea de varios procesos lanzados por el programa:
  - Lenguajes **concurrentes**: Ejemplo: Ada.
  - Lenguajes **no concurrentes**. Ejemplo: C.

- Según la **interactividad** del programa con el usuario u otros programas:
  - Lenguajes **orientados a sucesos**: El flujo del programa es controlado por la interacción con el usuario o por mensajes de otros programas/sistema operativo, como editores de texto, interfaz gráfica de usuario (GUI) o kernels. Ejemplos: Visual Basic, lenguajes de programación declarativos.
  - Lenguajes **no orientados a sucesos**: El flujo del programa no depende de sucesos exteriores, sino que se conoce de antemano, siendo los procesos batch el ejemplo más claro (actualizaciones de bases de datos, colas de impresión de documentos, etc.). Ejemplo: Lenguajes de programación imperativos.

- Según la **realización visual** o no del programa:
  - Lenguajes de **programación visual**: El programa se realiza moviendo bloques de construcción de programas (objetos visuales) en una interfaz adecuado para ello. No confundir con entornos de programación visual, como Microsoft Visual Studio y sus lenguajes de programación textuales (como Visual C#). Ejemplo: Mindscript.
  - Lenguajes de **programación textual**: El código fuente del programa se realiza escribiéndolo. Ejemplos: C, Java, Lisp.

CREACIÓN Y DESARROLLO DEL LENGUAJE C[3]

La versión original de C fue desarrollada en los laboratorios de la Bell Telephone por Denis Ritchie, en los años 70. Fue implementada por Ritchie en una DEC PDP 11, con sistema operativo UNIX.

Ritchie creó el lenguaje basándose en otros dos existentes: BCPL, desarrollado por Martin Richards en 1967, y basado en el lenguaje CPL, que había sido creado en 1963 en la Universidad de Cambridge y el B, que había ideado Ken Thompson, también basado en el BCPL. Todos estos lenguajes - el BCPL, el B y, como dijimos, el C - fueron desarrollados en los laboratorios de la Bell Telephone. Se creaban para uso exclusivo de los laboratorios y no eran conocidos fuera de ellos. Estaban ligados, fundamentalmente, al sistema operativo UNIX, en cuyo entorno se trabajaba y que Denis Ritchie reescribe en C. Por todo ello se piensa en C como un lenguaje de programación de sistemas, aunque, al salir de los laboratorios Bell, comienza a ser utilizado para muchas otras aplicaciones.

Recién en 1978 el lenguaje C sale de los laboratorios Bell cuando Ritchie, en colaboración con Brian Kernighan, publica **"The C Programming Language"**, editado por Prentice Hall.

A partir de allí comienza a extenderse el uso del lenguaje. Se escriben varios compiladores para permitir el uso del C en distintos equipos y se desarrollan infinidad de programas para aplicaciones comerciales y científicas.

A pesar de la alta portabilidad del lenguaje, las distintas implementaciones de C tenían algunas incompatibilidades.

Esto lleva al Instituto Nacional Americano de Estándares (ANSI) a formar, en 1983, una comisión para lograr una versión normalizada del lenguaje. Esta versión se completa con 1989: se la denomina ANSI C.

Pero desde 1980 Bjorne Stroustrup, también de los laboratorios Bell, trabajaba en un lenguaje que puede ser considerado como un súper conjunto del lenguaje C aunque posee también componentes del Algol 68 y del Simula 67. Este lenguaje es el C++, que Stroustrup describe en su libro **"The C++ Programming Language"**.

En 1983 comienza a ser utilizado el lenguaje fuera del grupo de su creador. C++ mantiene la compatibilidad con C.

Lo más importante de C++ es su posibilidad de utilización en programación orientada a objetos.

[3] Maquieira, J. V., Viard, G. (1998); *Programación en C;* Argentina; Alfati S. A.; página 1.

CARACTERÍSTICAS DEL LENGUAJE C[4]

Vamos a encontrar a menudo que a C se lo califica como un lenguaje de nivel medio. Esto es debido a que C no sólo puede utilizarse como lenguaje de alto nivel sino que también permite trabajar a bajo nivel, a la par de un lenguaje ensamblador. Como lenguaje de alto nivel es un lenguaje estructurado, como el Algol, el Pascal y el ADA entre otros.

Recordemos que en programación estructurada una de las normas esenciales es la modularidad. El lenguaje C lleva a sus últimos extremos esta modularidad: todo programa es un conjunto de bloques totalmente independientes.

Estos bloques se denominan **funciones**. Una función es una subrutina que contiene una o más instrucciones y que realiza una o más acciones. Existe una biblioteca estándar de C que constituye una importante recopilación, que el programador puede utilizar, además de las funciones que él mismo cree.

Cada función está individualizada por un **identificador**, es decir, un nombre, que elige libremente el usuario respetando ciertas normas que daremos en el próximo apartado y, eventualmente, una lista de argumentos entre paréntesis. Si no necesita parámetros se coloca a continuación del identificador, paréntesis vacíos.

La función que inicia el programa (una especie de programa principal), es la única a la que no podemos asignarle libremente un identificador; ella recibe el nombre de **main**.

Dentro de cada bloque las estructuras de control permiten crear sub-bloques que pueden considerarse como una unidad. Estos sub-bloques se encierran entre llaves: una llave que abre "**{**"al comienzo y una que cierra "**}**" al final del bloque.

C tiene un conjunto de instrucciones pequeño con respecto a otros lenguajes de alto nivel. Las instrucciones básicas están apoyadas y mejoradas por las numerosas funciones de la biblioteca estándar.

Los compiladores son de pequeña dimensión, así como los programas objeto generados por ellos.

[4] Maquieira, J. V., Viard, G. (1998); *Programación en C;* Argentina; Alfati S. A.; página 2.

ENTORNO DE DESARROLLO INTEGRADOS (IDE[5])

Para comenzar a hablar de este grupo de programas, lo primero que debemos saber es que toda aplicación es creada a partir de un lenguaje de programación.

Estos paquetes de programas están dirigidos al desarrollo aplicaciones y se llaman integrados, porque están compuesto de varios programas que se juntan para una misma función, que en este caso es crear una aplicación.

Todos los lenguajes de programación tienen su propio IDE, dado que los programas que lo componen son indispensables para poder crear aplicaciones.

Estos paquetes de programas incluyen básicamente: Editores de Textos, Compiladores, Intérpretes, Linkeadores y Depuradores.

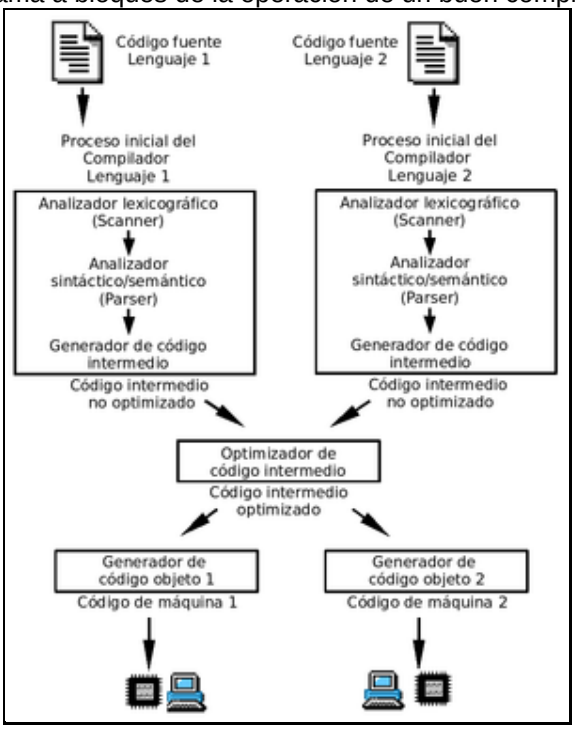
Funciones:

- **EDITOR DE TEXTOS:** Sirve para escribir el **código fuente** de los distintos lenguajes, existen distintos tipos de programas que cumplen esa función, esto quiere decir que ya están incorporados dentro de los utilitarios que forman parte de los distintos lenguajes. Cuando utilizamos los utilitarios propios del lenguaje estos tendrán la extensión correspondiente, si programamos con el lenguaje C, los archivos tendrán como extensión C o CPP, dependiendo del IDE que utilicemos. Estas extensiones identifican al tipo de lenguaje elegido para escribir el código fuente de la futura aplicación. Los editores de textos fueron creados para la escritura de programas, donde la apariencia del texto escrito no es importante, esto marca la diferencia con los Procesadores de Textos que apuntan exclusivamente a darle una mejor apariencia al texto escrito.

- **COMPILADOR E INTÉRPRETES**: Estos programas cumple dos funciones, la primera es la detectar los errores del tipo ortográfico (**análisis léxico**[6]) y gramatical (**análisis sintáctico**[7] **y semántico**[8]) que surgieran durante la escritura del código fuente. La segunda es la de traducir el código fuente al lenguaje de máquina, creando al objeto, que normalmente tiene como extensión OBJ. Cada lenguaje tiene su propio compilador o sea su traductor. Estos programas generalmente forman parte del sistema en el cual se va a realizar la compilación o interpretación, según el método que utilice cada sistema para realizar la traducción de estos, serán compilados o interpretados. Si un programa es creado en modo compilado, la aplicación se ejecuta a partir del objeto, en cambio sí se ejecuta en modo intérprete, las instrucciones del programa se ejecutan de una en una, siempre y cuando no tengan errores de sintaxis de forma directa no creándose el objeto correspondiente.

[5] Integrated Development Environment
[6] Analiza el código.
[7] Analiza la sintaxis.
[8] Analiza los tipos de datos.

Diagrama a bloques de la operación de un buen compilador[9]



Para un compilador existen dos ambientes:

- **Self-hosting o auto-hospedaje:** Son los compiladores nativos en los cuales el procesador que ejecuta el código usa un sistema operativo específico con manejo de archivos y diversos ambientes de ejecución.

- **Free Standing o compiladores cruzados:** es un compilador capaz de crear código ejecutable para otra plataforma distinta a aquella en la que el compilador se ejecuta, en los cuales el procesador que ejecuta es generalmente un microprocesador embebido o un microcontrolador sin servicios de algún sistema operativo (como en el caso de los sistemas embebidos).

Tipos de intérpretes:

- **Simple:** Es necesario cargar un intérprete para que "Traduzca" línea a línea la instrucción de un programa, suponiendo una menor performance, además de que necesariamente tener el intérprete. Supone una ventaja al tener que corregir el código no tener que compilarlo.

[9] https://es.wikipedia.org/wiki/Compilador#/media/Archivo:CompilationScheme-Spanish.png

- **Con Marco de trabajo:** También existen intérpretes que incluyen cierta "compilación" en el medio. Son aquellos que compilan a un código intermedio llamado bytecode[10], que es más eficiente de ejecutar que hacerlo directamente desde el código fuente. El bytecode es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código máquina.

- **VINCULADOR O LINKEADOR**: La función principal de estos programas es la de crear la aplicación, generando los archivos con extensión EXE. Para crearla lo que realiza es vincular a todos los objetos (bibliotecas o subprogramas) relacionados con la aplicación y crear el ejecutable. La vinculación del programa objeto para transformarlo en ejecutable es una función que puede formar parte del propio programa compilador, dependiendo esto del IDE que fue creado.

- **DEPURADOR O DEBUGGER**: El debugging o depuración es el proceso metodológico para encontrar y reducir **bugs** (errores) o defectos en un programa informático o en una pieza de hardware.

TIPOS DE DATOS

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, por ejemplo la letra ‘b’, le corresponde al número entero 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.

En la tabla se muestran los tipos de datos básicos para el Lenguaje C:

TIPOS	TAMAÑO EN BITS	RANGO
char	**8**	0 a 255
int	**16**	-32768 a 32767
float	**32**	3.4e-38 a 3.4e+38
double	**64**	1.7e-308 a 1.7e-308
void	**0**	Sin valor

Los valores del tipo **char** se usan normalmente para guardar valores definidos en el juego de caracteres ASCII, así como cualquier cantidad de 8 bits. Los valores de tipo **int** se usan para guardar cantidades enteras y los valores del tipo **float** o **double,** se usan para guardar números reales (Los números reales pueden tener un componente entero y uno fraccionario).

[10] http://www.alegsa.com.ar/Dic/bytecode.php

ARREGLOS (ARRAYS)[11]

Los arreglos son tipos de datos que reservan un espacio en la memoria RAM para almacenar temporalmente los datos ingresados. La cantidad de espacio que se reserve dependerá del tipo de dato y de la cantidad de elementos que lo compongan.

Los arreglos pueden almacenar tanto caracteres como cadena de caracteres y valores numéricos de distinto tipo (enteros y reales). Y pueden unidimensionales como los **VECTORES** o multidimensionales como las **MATRICES**.

Como se declaran los Arreglos:

VECTORES:

char nombre[15]; /*Creamos un vector de caracteres para almacenar una cadena de caracteres de 15 elementos. Reservando un espacio en memoria RAM de 15 bytes. (Cada carácter ocupa en memoria 1 byte).*/

float valores[10]; /*Creamos un vector numérico del tipo real para almacenar 10 valores. Reserva un espacio en memoria RAM de 40 bytes. (Cada número entero ocupa en memoria 4 bytes).*/

int valores[10]; /*Creamos un vector numérico del tipo entero para almacenar 10 valores. Reserva un espacio en memoria RAM de 20 bytes. (Cada número entero ocupa en memoria 2 bytes).*/

MATRICES:

char nombres[5][20]; /*Creamos una matriz para almacenar a 5 cadenas de caracteres de hasta 20 elementos cada una. Reserva un espacio en memoria RAM de 100 bytes. (Cada letra ocupa en memoria 1 byte).*/

float matriz[3][4]; /*Creamos una matriz de 3 filas por 4 columnas para almacenar valores numéricos reales. Reserva un espacio en memoria RAM de 48 bytes. (Cada número real ocupa en memoria 4 bytes).

int matriz[3][4]; /*Creamos una matriz de 3 filas por 4 columnas para almacenar valores numéricos enteros. Reserva un espacio en memoria RAM de 24 bytes. (Cada número entero ocupa en memoria 2 bytes).

[11] ANEXO VIII: LAS MATRICES

VARIABLES

Son objetos de un programa cuyo valor puede cambiar durante la ejecución del mismo. Datos que pueden sufrir modificaciones a lo largo de un programa. El cambio se produce mediante sentencias ejecutables como por ejemplo la asignación o el ingreso de datos. **Es** en realidad **una porción** (o posición) **de memoria con nombre** que permite almacenar temporalmente un dato durante la ejecución de un proceso. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo (a esto se lo denomina Declaración de variables). Al declararlas se reserva una posición de memoria para la misma, donde se almacenará el valor que toma cada variable en cada momento del programa. El nombre de la posición es el NOMBRE DE LA VARIABLE y el valor almacenado es el VALOR DE LA VARIABLE. Las variables pueden ser de todos los tipos de datos conocidos: entero, reales, carácter y cadena de caracteres.

DECLARACIÓN DE VARIABLES

Todas las variables han de ser declaradas antes de poder ser usadas. La forma general de declaración es la que se muestra a continuación: **tipo** (hace referencia al tipo de datos) lista **de variables** (son los nombres de las variables.

Aquí, tipo debe ser un tipo de datos válido de C y la lista de variables puede consistir en uno o más nombres de identificadores separados por comas. A continuación se muestran algunas declaraciones en Pseudocódigo y en C:

En Pseudocódigo	En C
entero cantidad;	int cantidad;
real valor;	float valor;
carácter letra;	char letra;
cadena palabra[cantidad];	char palabra[cantidad];

Recuerde que en C el nombre de una variable no tiene nada que ver con su tipo. Existen dos tipos básicos donde se pueden declarar variables: dentro de las funciones y fuera de todas las funciones. Estas variables son, respectivamente, las variables locales y las variables globales.

I. Variables locales.

Las variables que se declaran dentro de una función se denominan variables locales. Las variables locales pueden ser denominadas sólo por sentencias que estén dentro del bloque en el que han sido declaradas. O dicho de otra forma, las variables locales no son conocidas fuera de su propio bloque de código. Recuerde que un bloque de código comienza con una llave abierta y termina con una llave cerrada.

Una de las cosas más importantes que hay que comprender sobre las variables locales es que sólo existen mientras se está ejecutando el bloque de código en el que son declaradas, o sea, la variables local se crea al entrar en el bloque y se destruye al salir de él.

II. **Variables globales.**

A diferencia de las variables locales, las variables globales se conocen a lo largo de todo el programa y se pueden usar en cualquier parte del código. Además, mantienen todo su valor durante toda la ejecución del programa. Las variables globales se crean al declararlas en cualquier expresión independientemente de la función en la que se encuentre la expresión.

INICIALIZACIÓN DE VARIABLES

En Pseudocódigo	En C
entero cantidad ← 10;	int cantidad = 10;
real valor ← 8.2;	float valor = 8.2;
carácter letra ← ‘a’ ;	char letra = ’a’;
entero nro1 ← 3, nro2 ← 80;	int nro1 = 3, nro2 = 80;

CONSTANTES

Dato invariable a lo largo del programa. Es un valor que no puede cambiar durante la ejecución del programa; recibe un valor en el momento de la compilación del programa y este valor no puede ser modificado.

El lenguaje C soporta todo tipo de dato como constante, además de las de los tipos de datos predefinidos. Se trata de una cadena. Una constante de cadena siempre irá encerrada entre dobles comillas, como en "LA FACULTAD”. Una constante de un solo carácter va entre comillas simples, como 'a'.

Para declararla se utiliza la sentencia del preprocesador **define** precedida con el siguiente carácter # (numeral) de la siguiente forma:

En Pseudocódigo	En C
MAXIMO 100	#define MAXIMO 100
NOMBRE “LA FACULTAD”	#define NOMBRE “LA FACULTAD”
SIMBOLO ‘a’	#define SIMBOLO ‘a’
VALOR 7.50	#define VALOR 7.50

Reglas para formar el identificador (nombre) de una variable o constante:

- Debe comenzar con una letra (mayúscula o minúscula), nunca con un número o espacio, y no deben contener espacios en blanco.
- Letras, números y el carácter guion (-) están permitidos después del primer carácter.
- La longitud de identificadores. El nombre de un identificador debe ser descriptivo de aquello que representa, y considerando además la practicidad de su invocación durante el desarrollo del algoritmo. Si es muy extenso estaremos más expuestos a errores al nombrarlo.
- Normalmente los nombres de las variables se colocan en letras minúsculas y las constantes con letras mayúsculas.

BIBLIOTECAS[12]

Las bibliotecas cumplen la función de brindarle información a los programas, tanto para su creación como para su ejecución. Una biblioteca puede formar parte de un programa (**internas)**, o sea que se las incorpora dentro de él, como ocurre con los archivos de encabezado que se utilizan en el lenguaje C (***.h**), o pueden ser compartidas por varios programas que requieren de información adicional para poder continuar con su ejecución (**externas**), como ocurre por ejemplo con las bibliotecas dinámicas de Windows (***.DLL**).

Las bibliotecas poseen el código que van a utilizar las distintas funciones de un programa.

El carácter # (numeral) también nos sirve para que le indiquemos al C que vamos a utilizar una sentencia del preprocesador **include** que nos permite incluir las bibliotecas internas o archivos de encabezado que vamos a utilizar para la ejecución de un programa de la siguiente manera:

En Pseudocódigo	En C
No existe	#include <biblioteca>

OPERADORES Y OPERANDOS

- **Operadores:** Son elementos que relacionan de forma diferente, los valores de una o más variables y/o constantes. Es decir, los operadores nos permiten manipular valores.

- **Operandos:** Son los datos a ser procesados.

[12] En el ambiente informático también se las llama librería por la traducción de la palabra biblioteca al inglés que es Library.

- **Tipos de Operadores** { Asignación, Aritméticos, Relacionales, Lógicos

OPERADOR DE ASIGNACIÓN

La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor.

variable ← expresión
(Una expresión puede ser una variable, una constante, una expresión o fórmula para evaluar)

La asignación se puede clasificar de la siguiente forma:

- **Simples:** Consiste en pasar un valor constante a una variable:

En Pseudocódigo	Ejemplo	En C	Ejemplo
←	a ← 15	=	a = 15

- **Contador:** Es una variable que se incrementa, cuando se ejecuta, en una unidad o en una cantidad constante:

En Pseudocódigo	En C	Acción
contador ← contador + 1	**contador = contador + 1**	Contador
multiplo ← multiplo + 3	**multiplo = multiplo + 3**	Múltiplo

- **Acumulador:** Es una variable que se incrementa en una cantidad variable

En Pseudocódigo	En C	Acción
suma ← suma + numero	**suma = suma + numero**	Acumulador

(Donde “numero” es una variable que puede recibir distintos valores)

Considerar:

- Una variable del lado derecho debe tener valor antes de que la sentencia se ejecute. Si “numero” no tiene valor antes de: suma ← suma + numero Se produce un Error LÓGICO. Se dice que “numero” no se ha **inicializado.**
- A la izquierda de una sentencia de asignación sólo puede haber variables. No puede haber operaciones.

Nota: la operación de asignación es una **operación destructiva** debido a que el valor almacenado en una variable se pierde o destruye y se sustituye por el nuevo valor de asignación.

Ejemplos: numero ← 16
numero ← -23

"La variable **numero** conservará el último valor asignado, en este caso **-23"**

- **Entrada:**

La entrada de datos consiste en recibir desde un dispositivo de entrada (por ejemplo un teclado) un valor. Esta operación se representa en pseudocódigo como sigue:

Leer (a)
Leer (b)

Donde "**a**" y "**b**" son las variables que recibirán los valores

- **Salida:**

Consiste en mandar por un dispositivo de salida (por ejemplo: monitor o impresora) un resultado o mensaje. Este proceso se representa en pseudocódigo como sigue:

Mostrar ("El resultado es:", R)

Donde "**El resultado es:**" Es el mensaje que vamos a ver y **R** es una variable que contiene un valor.

OPERADORES ARITMÉTICOS

- Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).
- Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.

En Pseudocódigo	Ejemplo	En C	Ejemplo	Acción
-	a ← 6 – 3	**-**	a = 6 - 3	Resta
+	a ← 6 + 3	**+**	a = 6 + 3	Suma
***, × o el ·**	a ← 6 × 3	*****	a = 6 * 3	Multiplicación (Producto)
/, : o el ÷	a ← 6 ÷ 3	**/**	a = 6 / 3	División
Mod	a ← 6 mod 3	**%**	a = 6 % 3	Resto de la división
Fórmula	a ← a + 1	**++**	a = ++	Incremento
Fórmula	a ← a – 1	**--**	a = --	Decremento

OPERADORES RELACIONALES

- Se utilizan para establecer una relación entre dos valores.
- Compara estos valores entre sí y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).
- Los operadores relaciónales comparan valores del mismo tipo (numéricos o cadenas)
- Tienen el mismo nivel de prioridad en su evaluación.
- Los operadores relaciónales tiene menor prioridad que los aritméticos.

En Pseudocódico	En C	Acción
>	**>**	Mayor que
<	**<**	Menor que
>= o ≥	**>=**	Mayor o igual que
<= o ≤	**<=**	Menor o igual que
=	**==**	Igual que
<> o ≠	**!=**	Distinto o no igual

OPERADORES LÓGICOS

- Estos operadores se utilizan para establecer relaciones entre valores lógicos.
- Estos valores pueden ser resultado de una expresión relacional.
- En el término operador lógico la palabra lógico se refiere a las formas en que esas relaciones pueden conectarse entre sí siguiendo las reglas de la lógica formal.

En Pseudocódigo	En C	Acción
And / Y	**&&**	Producto Lógico
Or / O	**||**	Suma Lógica
Not / No	**!**	Negación

- La clave de los conceptos de operadores lógicos es la idea de verdadero y falso. En C, verdadero es cualquier valor distinto de 0, y falso es 0. Las expresiones que utilizan los operadores lógicos devuelven el valor 1 en caso de verdadero y 0 en caso de falso, siendo la tabla de verdad para los operadores lógicos:

TABLAS DE VERDAD

Operador AND = &&		
Operando1	Operando2	Resultado
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Operador OR = ||		
Operando1	Operando2	Resultado
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Operador NOT = !	
Operando	Resultado
0	**1**
1	**0**

ESTILO DE PROGRAMACIÓN

Antes de escribir los algoritmos de los ejercicios en Lenguaje C considerar:

Es **un programa legible y comprensible** **más fácil de:** { **- Entender**, **- Corregir**, **- Mantener**

Seguir las siguientes sugerencias:

1. **SANGRADO O INDENTACIÓN**. En cada estructura, y alineando las instrucciones (sentencias) dentro de cada una de ellas y dentro de todo el algoritmo (Comienzo, Fin)

2. **LÍNEAS EN BLANCO**. Dejarlas entre partes importantes o que estén lógicamente separadas. Recomendable luego de cada estructura (Repetitiva o Selectiva), luego de declaración de variables.

3. **COMENTARIOS**. Parte importante de la documentación de un programa que permite mayor comprensión de este, usaremos el mismo formato que en el lenguaje C:

   - **Párrafo: /***Esto es un comentario (puede ocupar varias líneas.***/**
   - **Línea: //**Esto va en la misma línea.

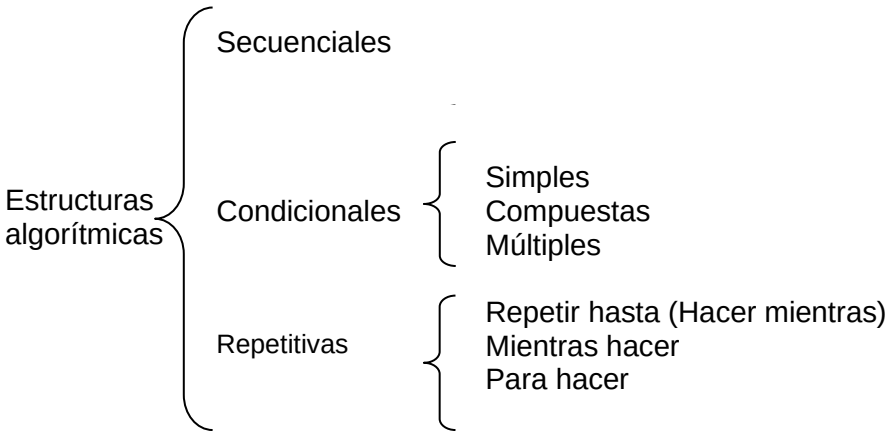
4. **NOMBRES SIGNIFICATIVOS DE IDENTIFICADORES**. que representen aquello que estamos tratando. Si son palabras compuestas usar guion común. Todos deben comenzar con una letra.

5. **CADA SENTENCIA EN UNA LÍNEA DISTINTA**. Al colocar una nueva sentencia comenzar una nueva línea, incluso las palabras claves de las estructuras en líneas separadas.

6. **ESPACIOS ENTRE ELEMENTOS DE UNA SENTENCIA**. lo hace más legible. Por ejemplo: **suma ← suma + numero**

ESTRUCTURAS DE CONTROL

En los lenguajes de programación, las estructuras de control son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución de problemas. Estas estructuras se clasifican de acuerdo con su complejidad en:



ESTRUCTURAS SECUENCIALES

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. Una estructura secuencial se representa de la siguiente forma:

CARACTERÍSTICAS

En Pseudocódigo	En C
COMIENZO Accion1 Accion2 . . AccionN FIN	**main()** /*Siempre se utiliza la función main() para definir la estructura principal del programa*/ **{** // Inicio del programa. **Sentencia1;** **Sentencia2;** **.** **.** **SentenciaN;** **}** // Fin del programa.

EJEMPLOS

1. Se deben ingresar dos números enteros cualquiera para realizar las cuatro operaciones matemáticas básicas. Mostrar los resultados correspondientes.

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
/*La función main()define la estructura principal del programa*/
main()
{  // Comienzo del programa.
   /*Declaro las variables, las variables siempre se escriben en minúsculas*/
   int nro1;
   int nro2;
   int suma;
   int resta;
   int producto;
   int division;
   /*Ingreso los datos*/
   scanf ("%i", &nro1);
   scanf ("%i", &nro2);

  /*Realizo los cálculos*/
   suma = nro1 + nro2;
   resta = nro1 – nro2;
   producto = nro1 * nro2;
   division = nro1 / nro2;
/*Muestro los resultados*/
   printf ("%i", suma);
   printf ("%i", resta);
   printf ("%i", producto);
   printf ("%i", division);
}  // Fin del programa.
```

2. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente.

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
/*La función main()define la estructura principal del programa*/
main()
{  // Comienzo del programa.
   /*Declaro las variables todas juntas y siempre en minúsculas*/
   float compra, descuento, pago;
```

```
   /*Ingreso el dato*/
   scanf("%f", &compra);
   /*Realizo los cálculos*/
   descuento = compra * 0.15;
   pago = compra -  descuento;
   /*Muestro el resultado*/
   printf("%f", pago);
}  // Fin del programa.
```

El carácter Ampersand "&" permite ingresar un dato a una variable declarada y las sintaxis "%f" y "%i" son modificadores de formato que permiten reconocer a los tipos de datos con las funciones scanf() y printf(). La 1º función corresponde a la entrada de datos y la 2º a la salida de datos.

Ver "Modificadores de Formato" de los Anexos

ESTRUCTURAS CONDICIONALES

Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que, sobre la base del resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen tres tipos básicos, las simples, las compuestas y las múltiples.

CARACTERÍSTICAS

Simples:

Las estructuras condicionales simples se les conoce como "Tomas de decisión". Estas tomas de decisión tienen la siguiente forma:

En Pseudocódigo	En C
SI (condición) ENTONCES Acciones FIN-SI	**if (condición)** **{** // Inicio del bloque. **Sentencias;** **}** // Fin del bloque.

Análisis del Pseudocódigo:	
Si……	Indica el comando de comparación
Condición……	Indica la condición a evaluar
Entonces……	Precede a las acciones a realizar cuando se cumple la condición
Acción(es)……	Son las acciones a realizar cuando se cumple o no la condición

Compuestas:

Las estructuras condicionales compuestas permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:

En Pseudocódigo	En C
SI (condición) ENTONCES Acciones SINO Acciones FIN-SI	**if (condición)** **{** // Inicio del bloque. **Sentencias;** **}** // Fin del bloque. **else** **{** // Inicio del bloque. **Sentencias;** **}** // Fin del bloque.

Análisis del Pseudocódigo:	
Si……	Indica el comando de comparación
Condición……	Indica la condición a evaluar
Entonces…….……	Precede a las acciones a realizar cuando se cumple la condición
Acción(es)……	Son las acciones a realizar cuando se cumple o no la condición
Sino……	Precede a las acciones a realizar cuando no se cumple la condición

Nota: Dependiendo del resultado de la comparación, que puede ser verdadera o falsa, se pueden realizar una o más acciones.

Múltiples:

Las estructuras de comparación múltiples, son tomas de decisiones especializadas que permiten comparar una variable, contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

En Pseudocódigo:	En C
LEER (variable) CON-SELECCIÓN (variable) HACER CASO constante 1: Acciones ROMPER CASO constante 2: Acciones ROMPER CASO constante N: Acciones ROMPER	**scanf(variable);** **switch (variable)** **{** // Inicio del bloque de la estructura. **case 1:** // El 1 es la constante. **sentencias;** **break;** /* Fuerza la salida de cada condición*/ **case 2:** **sentencias;** **break;** **case N:** **sentencias;** **break;**

OTROS CASOS: Acciones FIN-SELECCIÓN	**default:** /*Entra en esta condición si las demás no son verdaderas*/ **sentencias;** **}** // Fin del bloque de la estructura.

Análisis de una estructura múltiple:

- La estructura de selección múltiple sólo compara por igualdad el valor de la variable con cada una de las constantes de cada caso. Al encontrar una coincidencia comienza a ejecutar las sentencias en forma secuencial hasta encontrar el fin de la estructura o una instrucción que rompa la misma.
- Puede tener hasta 256 casos.
- No puede haber 2 casos con el mismo valor en la constante.
- Sólo se pueden utilizar variables de tipo carácter o enteras.
- Si la variable que se está seleccionando es de tipo carácter, las constantes de tipo carácter se colocan entre comillas simples o apóstrofes, para el caso de variables de tipo enteras, las constantes numéricas se colocan directamente.
- Puede contener casos vacíos.

EJEMPLOS

1. **Simples:** Ingresar un número positivo (debe ser mayor a 0) y mostrar el mensaje si la condición es verdadera, "**El número Nº es positivo**".

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
/*La función main() define la estructura principal del programa*/
main()
{  // Comienzo del programa.
   /*Declaro las variables, las variables siempre se escriben en minúsculas*/
   int nro;
   /*Ingreso el dato*/
   scanf("%i", &nro);
/*Utilizo la estructura condicional simple*/
   if (nro > 0)
      /*Muestro el mensaje*/
      printf("El número %i es positivo", nro);
}  // Fin del programa.
```

2. **Compuestas:** Hacer un algoritmo que calcule el total a pagar por la compra de camisas. Si se compran tres camisas o más se aplica un descuento del 20% sobre el total de la compra y si son menos de tres camisas un descuento del 10%.

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
/*La función main() define la estructura principal del programa*/
main()
{  // Comienzo del programa.
/*Declaro las variables, las variables siempre se escriben en minúsculas*/
   int total, cantidad, precio;
   /*Ingreso los datos*/
   scanf("%i", &cantidad);
   scanf("%i", &precio);
   /*Cálculo el total*/
   total = cantidad * precio
  /*Utilizo la estructura condicional compuesta*/
   if (cantidad >= 3 )
      /*Cálculo el total para la condición verdadera*/
      total = total - total * 0.20;
   else
      /*Cálculo el total para la condición falsa*/
      total = total - total * 0.10;
   /*Muestro el resultado*/
   printf("%i",  total);
}  // Fin del programa.
```

3. **Múltiples:** Ingresar 2 números y realizar un menú de opciones para que los sume, reste o los muestre.

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
/*La función main() define la estructura principal del programa*/
main()
{  // Comienzo del programa.
/*Declaro las variables, las variables siempre se escriben en minúsculas*/
   int nro1, nro2, opción;
   int suma, resta;
/*Ingreso los datos*/
   scanf("%i", &nro1);
   scanf("%i", &nro2);
```

```
/*Realizo el menú*/
  printf("\t\tMenú\n");
  printf("1.Suma – 2.Resta\n");
  /*Ingreso la opción para el menú*/
  scanf("%i", &opción);
  /*Utilizo la estructura condicional múltiple*/
  switch (opción)
  { // Inicio de la estructura múltiple.
      case 1 :
      /*Acciones para la opción 1*/
        suma = nro1 + nro2;
        printf("\nLa suma es:  %i", suma);
        break;
      case 2:
      /*Acciones para la opción 2*/
        resta = nro1 – nro2;
        printf("\nLa resta es:  %i", resta);
        break;
      default:
      /*Acciones para la opción mayor a 2*/
        printf("\nLos Nº ingresados son:\n");
        printf("%i\t-\t%i", nro1, nro2);
  } // Fin de la estructura múltiple.
} // Fin del programa.
```

En la estructura de condicional múltiple comenzamos a utilizar las secuencias de escape "\t" para la tabulación horizontal y "\n" para el salto de línea o ENTER.

Ver "S e c u e n c i a s d e E s c a p e" de los Anexos

ESTRUCTURAS REPETITIVAS

Se llaman estructuras repetitivas o cíclicas a aquellas en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa).

Estos tipos de estructuras generan un número de iteraciones que no se conocen con exactitud, ya que esta dado en función de un dato dentro del programa.

Este tipo de estructura se utiliza normalmente para los arreglos (**arrays**) unidimensionales (**vectores**) y bidimensionales o multidimensionales (**matrices**).

Las estructuras repetitivas se clasifican en: **MIENTRAS-HACER, HACER-MIENTRAS y PARA-HACER**.

CARACTERÍSTICAS

1. **MIENTRAS-HACER:** Ejecuta las sentencias que contiene mientras la condición sea VERDADERA. Primero evalúa la condición, y de ser VERDADERA ejecuta las sentencias. Esto supone que la variable que forma parte de la condición tenga un valor inicial.

En Pseudocódigo	En C
MIENTRAS (condición) HACER Acciones FIN-MIENTRAS	**while (condición)** **{** // Inicio de la estructura. **Sentencias;** **}** // Fin de la estructura.

2. **HACER-MIENTRAS:** Ejecuta las sentencias que contiene mientras la condición sea VERDADERA. A diferencia de la estructura anterior, primero ejecuta las sentencias y luego evalúa la condición.

En Pseudocódigo	En C
HACER Acciones MIENTRAS (condición)	**do** **{** // Inicio de la estructura. **Sentencias;** **}** // Fin de la estructura. **while (condición);**

3. **PARA-HACER:** Son aquellos en que el número de iteraciones se conoce antes de ejecutarse el ciclo. Esta estructura cuenta con tres campos: variable, condición y el contador.

Estructura del PARA-HACER:

En Pseudocódigo	En C
PARA (variable; condición; contador) HACER Acciones FIN-PARA	**for (variable; condición; contador)** **{** // Inicio de la estructura. **Sentencias;** **}** // Fin de la estructura.

Análisis de la estructura:
variable: Esta variable cumple la función de asignarle un valor inicial al contador, el valor puede ser cualquiera. A esta variable contador se la conoce en los arreglos como **índice**. **condición**: La condición le indica a la estructura PARA-HACER **desde** y **hasta** que número se va a repetir el ciclo. Desde, nos indica el límite inferir y hasta límite superior. **contador:** El contador permite incrementar o decrementar su valor inicial hasta que se cumpla la condición. Se puede incrementar o decrementar con los que sean necesarios para el programa (de 1 en 1, de 2 en 2, etc.)

Nota: En este ciclo la variable de control toma el valor inicial del ciclo y el ciclo se repite hasta que la variable de control llegue al límite superior. La cantidad de repeticiones que tenga depende del límite superior y del incremento de la variable.

EJEMPLOS

MIENTRAS-HACER: Leer 10 números, sumarlos y mostrar el resultado.

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
#include <stdlib.h>
/*La función main() define la estructura principal del programa*/
main()
{  // Comienzo del programa.
   /*Declaramos e inicializamos las variables para el contador y el
   acumulador*/
   int suma = 0;
   int contador = 0;
   int numero;
   /*Utilizo la estructura repetitiva mientras que el contador sea
   menor a 10*/
   while (contador<10)
   {  // Inicio la estructura.
      /*Ingreso el dato*/
      scanf("%i", &numero);
      /*Incremento el contador*/
      contador = contador + 1;
      /*Uso el acumulador suma*/
      suma = suma + numero;
   }  // Fin de la estructura.
   /*Mostramos el contenido de acumulador*/
   printf("\nLa suma es: %i", suma);
   /*Esta función espera que se pulse una tecla para continuar*/
   system(pause);
} // Fin del programa.
```

HACER-MIENTRAS: Ingresar **n** cantidad de números y averiguar cuál es el número mayor y el menor. Mostrarlos.

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
#include <stdlib.h>
```

```
/*La función main() define la estructura principal del programa*/
main()
{  // Comienzo del programa.
/*Declaramos e inicializamos las variables para averiguar el valor máximo y mínimo*/
   int mayor = 0;  /*Se debe inicializar con el valor mínimo posible para que  cualquier número ingresado sea  mayor a esta variable*/
   int menor = 9999; /*Se debe inicializar con el valor mayor posible para que cualquier número ingresado sea menor a esta variable*/
   int numero;
 /*Declaro la variable para poder salir  del programa*/
   char continuar = 's'; /*Las variables del tipo carácter se inicializan con el  carácter entre apóstrofes*/
   system("cls"); /*De esta forma ejecutamos al comando cls[13] del S.O. [14]para borrar la pantalla*/
   /*Utilizo  la  estructura  repetitiva  hacer-mientras  hasta  que decidamos lo contrario*/
   do
   { // Inicio la estructura.
      /*Ingreso el dato*/
      scanf("%i", &numero);
      /*Averiguo el valor máximo*/
      if (numero > mayor)
      /*Le asigno el valor de numero a la variable mayor*/
         mayor = numero;
      /*Averiguo el valor mínimo*/
      if (numero < menor)
      /*Le asigno el valor de numero a la variable menor*/
         menor = numero;
      /*Decido si quiero ingresar más números*/
      printf("\nContinuo S/N:\t");
   /*Para ingresar un solo carácter conviene hacerlos de esta forma*/
      fflush(stdin);  /*Vaciamos  el  buffer  del  teclado  y  siempre debemos  usarla  antes  de  leer  un  caracter  o  una  cadena  de caracteres*/
      scanf("%c", &continuar);
/*Sí pulsamos un carácter distinto de 's' continuamos dentro del programa, caso contrario salimos*/
   } // Fin de la estructura.
   while (continuar == 's');
```

[13] El comando cls del S.O. significa CleanScrin, esto significa que se borrara todo lo que estuviera escrito en la pantalla del monitor previamente a esta orden.
[14] S.O. significa Sistema Operativo.

```
   /*Mostramos los valores máximo y mínimo*/
   printf("\nMáximo:\t%i", mayor);
   printf("\nMínimo:\t%i", menor);
   /*Hago una pausa en espero que se pulse una tecla para continuar*/
   system("pause"); /*De esta forma ejecutamos al comando pause del S.O. para detener el programa, aparecerá en pantalla el mensaje de pulsar una tecla para continuar sin necesidad de escribir dicho mensaje con la función printf()*/
} // Fin del programa.
```

PARA-HACER: Ingresar 10 números y mostrarlos en forma de lista. Sacar el promedio y mostrar el resultado.

En C

```
/*Indico las bibliotecas que voy a utilizar*/
#include <stdio.h>
#include <stdlib.h>1
#include <string.h>
#define CANTIDAD 10 /*CANTIDAD es una constante y las constantes siempre se deben declarar en mayúsculas*/
main()
{ //Comienzo del programa.
   /*declaro las variables*/
   int vector[CANTIDAD]; /*Entre corchetes se ponen la cantidad de elementos del arreglo*/
   int contador; /*El contador cumple la función de un índice para los arreglos*/
   float promedio;
   int suma = 0;
/*Está variable la utilizamos para ingresar el título que queremos*/
   char titulo[30];
   system("cls"); /*De esta forma ejecutamos al comando cls del S.O. para borrar la pantalla*/
   for (contador = 0;
/*Inicializo el contador*/
            contador < 10;
/*La condición es desde el límite inferior 0 hasta el límite superior 9*/
            contador = contador + 1)
/*Incremento el contador de 1 en 1*/
    { //Inicio de la estructura
/*Ingreso un número en el arreglo en la posición en que se encuentre el contador (índice)*/
       scanf("%i", &vector[contador]);
/*Realizamos el cálculo con los valores ingresados en el  arreglo*/
       suma = suma+vector[contador];
   } // Fin de la estructura
```

```
/*Ingresamos el siguiente título "Estos son los Nº ingresados"*/
   printf("\nIngresamos el Título:\t");
   scanf("%s", titulo);
   system("cls"); /*De esta forma ejecutamos al comando cls del
DOS para borrar la pantalla*/
/*Los títulos se ponen siempre fuera de las estructuras repetitivas*/
   printf("\n%s\n", titulo)
  for /*Inicializo el contador*/
        (contador = 0;
/*La condición es desde el límite inferior 0 hasta el límite superior 9,
en este caso en vez de poner el número de elementos del arreglo
ponemos directamente la constante*/
        contador < CANTIDAD;
/*Incremento el contador de 1 en 1*/
        contador = contador + 1)
   { // Inicio de la estructura
/*Muestro el número que encuentra en la posición del contador
(índice) dentro del arreglo*/
      printf("%i\n", vector[contador]);
    } // Fin de la estructura
/*Realizo el cálculo del promedio usando el acumulador y el
contador*/
    promedio = suma / contador;
/*Muestro el promedio*/
    printf("Promedio:%f\n", promedio);
   system("pause"); /*De esta forma ejecutamos al comando pause
del S.O. para detener el programa, aparecerá en pantalla el mensaje
de pulsar una tecla para continuar sin necesidad de escribir dicho
mensaje con la función printf()*/
} // Fin del programa.
```

ANEXOS

COMENTARIOS A TENER EN CUENTA
1. Las constantes o variables del tipo carácter se colocan entre apóstrofes (comillas simples), porque se utilizan para un sólo carácter y las cadenas siempre entre comillas dobles, porque se utilizan para más de un caracter.
2. Para salir de una estructura repetitiva antes de que finalice (por su condición) o alcance el número de iteraciones indicados (en el **for()**) utilizamos la sentencia **break** (ROMPER O SALIR DE LA ESTRUCTURA).
3. Para salir de cada CASO de una estructura de selección múltiple **switch()**, se debe utilizar la sentencia **break** (PARA ROMPER O SALIR DE CADA CASO).
4. Los comentarios los hacemos como lo indica el lenguaje C de la siguiente manera: **/*Para un Párrafo o varias líneas*/** y **//Para una Línea**.
5. Cuando dentro de una estructura hay más de una sentencia las mismas se deben poner entre llaves **{Sentencias; }** que indican el comienzo o final de un bloque respectivamente.
6. Cada bloque de un programa en el lenguaje C comienza con una llave de apertura **"{"**, y debe terminar con una la llave de cierre **"}"**.
7. Los argumentos de una función son los distintos parámetros que incluimos entre paréntesis y que le permiten realizar la acción para la cual fueron diseñadas, por ejemplo: scanf **(**argumentos**)**; printf **(**argumentos**)**; etc.
8. La cantidad de elementos que va a contener un arreglo que ponen entre corchetes. Ejemplos: char vector **[**CANTIDAD DE ELEMENTOS**]**; int matriz **[**CANTIDAD DE FILAS**] [**CANTIDAD DE COLUMNAS**]**;
9. Después de cada sentencia en el lenguaje C se debe poner punto y coma.
10. Todo contador cumple la función de un índice.
11. Un acumulador cumple la función de un sumador.
12. Tanto los acumuladores como los contadores deben ser inicializados para que comiencen a sumar o contar a partir de un número determinado.
13. Todos los mensajes de salida se deben poner entre comillas dobles.
14. Las variables en general, se escriben preferentemente con letra en minúsculas.
15. Las constantes en el lenguaje C se escriben para diferenciarlas de las variables, siempre con letras en mayúsculas.
16. Las variables y constantes en el lenguaje C no deben empezar con un número ni deben tener espacios entre caracteres.
17. Los nombres de las constantes y variables deben ser claros e identificatorios del dato que va a almacenar.
18. La mayoría de las sentencias del lenguaje C son funciones y representan a cada una de las acciones que va a realizar un programa.
19. Toda función para poder ser utilizada debe tener como archivo de encabezamiento a la biblioteca a la cual pertenece.
20. Al lenguaje C se lo cataloga como un lenguaje de nivel medio, porque a diferencia del lenguaje assembler o ensamblador, en donde cada instrucción equivale a una instrucción en lenguaje de máquina y por consiguiente es de nivel bajo, el lenguaje C utiliza funciones que para poder crearlas se necesitan una indeterminada cantidad de instrucciones para poder realizar a cada una de las acciones que requiera un programa.
21. En el lenguaje C se pueden utilizar por defecto hasta 16 colores, tanto para los fondos como para la fuente. Si se quieren usar más se deben incorporar como archivos de encabezados a bibliotecas específicas.

22. El símbolo de numeral se utiliza para declarar las sentencias del preprocesador que nos sirven para declarar una constante con la sentencia **#define** y para declarar una biblioteca interna o estática, con la sentencia **#include**.
23. Con el lenguaje C fue creado el sistema operativo **Unix** y sus derivados, como los sistemas operativos **Minix** y **Linux**.
24. Un procedimiento está compuesto de una secuencia de instrucciones.
25. Los sistemas informáticos clásicos están compuestos por tres procedimientos básicos, que son las **Altas**, las **Bajas** y las **Modificaciones** que se puedan realizar con un programa, y se los conoce como **ABM**. Estos sistemas también pueden incluir **Listados**, **Búsquedas** y todo tipo de cálculos que sean indispensables para la resolución de un programa.
26. El procedimiento de **Altas**, significa que por medio de un programa se van a ingresar o cargar datos.
27. El procedimiento de **Bajas**, significa que por medio de un programa vamos a eliminar o borrar un dato.
28. El procedimiento de **Modificación**, significa que por medio de un programa podemos cambiar algunos de los datos ingresados por uno nuevo.
29. El procedimiento de **Listados**, significa que por medio de un programa podemos mostrar a los datos ingresados de uno por vez o en forma de lista, también se lo conoce con la acción de imprimir.
30. El procedimiento de **Búsqueda**, significa que por medio de un programa podemos encontrar a un dato o a un grupo de datos que el programa requiera.

MODIFICADORES DE FORMATO

Hemos visto por encima cómo mostrar datos en pantalla y cómo aceptar la introducción de datos por parte del usuario, mediante "printf()" y "scanf()", respectivamente. Veamos ahora su manejo y algunas de sus posibilidades con más detalle, junto con otras órdenes alternativas: El formato de printf() es printf (formato, lista de variables); Dentro del apartado de "formato" habíamos comentado que "%d" indicaba que se iba a escribir un número entero, y que "%s" indicaba una cadena de texto. Vamos a resumir en una tabla las demás posibilidades, que no habíamos tratado todavía:

MODIFICADOR	FORMATO POR TIPO DE DATO
%i o **%d**	Número entero con signo, en notación decimal.
%u	Número entero sin signo, en notación decimal.
%o	Número entero sin signo, en notación octal (base 8).
%x	Número entero sin signo, en hexadecimal (base 16), minúsculas.
%X	Número entero sin signo, en hexadecimal, mayúsculas.
%f	Número con decimales (coma flotante o punto flotante).
%e	Número real en notación científica.
%g	Usa el valor más corto entre %e y %f.
%c	Un único carácter.
%s	Cadena de caracteres.
%%	Signo de tanto por ciento: %
%p	Puntero (dirección de memoria)
%n	Se debe indicar la dirección de una variable entera (como en el scanf()), y en ella quedará guardado el número de caracteres impresos hasta ese momento.

EJEMPLO: Uso de los Modificadores de Formato.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*Declaro variables globales*/
int   entero = 1234, enteroNeg = -1234, contador;
float real = 234.567;
char  letra = 'E', mensaje[20] = "Un texto";
main() //Esta es la función principal del programa.
{
   system("cls");
   printf("El número entero vale %d en notación decimal,\n", entero);
   printf("  y %o en notación octal,\n", entero);
   printf("  y %x en notación hexadecimal,\n", entero);
   printf("  y %X en notación hexadecimal en mayúsculas,\n", entero);
   printf("  y %ld si le hacemos que crea que es entero largo,\n", entero);
   printf("  y %10d si obligamos a una cierta anchura,\n", entero);
   printf("  y %-10d si ajustamos a la izquierda.\n", entero);
   printf("El entero negativo vale %d\n", enteroNeg);
   printf("  y podemos hacer que crea que es positivo: %u (incorrecto).\n",
           enteroNeg);
   printf("El número real vale %f en notación normal\n", real);
   printf("  y %3.2f si limitamos a dos decimales,\n", real);
   printf("  y %e en notación científica (exponencial).\n", real);
   printf("La letra es %c y el texto %s.\n", letra, mensaje);
   printf("  Podemos poner \"tanto por ciento\": 50%%.\n");
   printf("Finalmente, podemos escribir direcciones. de memoria: %p.\n",
            letra);
   printf("  y contar lo escrito hasta aquí %n", &contador);
   printf(", que ha sido: %d letras.\n", contador);
   system("pause");
   return 0; /* Esta sentencia reemplaza al void en una función, pero a
diferencia retorna el valor que le indiquemos que puede ser una variable o una
constante*/
}
```

SECUENCIAS DE ESCAPE	
Las secuencias de escape cumplen cuatro funciones:	1. Cambiar de lugar al cursor. 2. Colocar algún tipo de carácter específico (comillas, apóstrofe o contrabarra). 3. Indicar el fin de una cadena de caracteres. 4. Hacer una llamada.
SECUENCIA	**FUNCIÓN**
\n	Nueva línea (Enter)
\r	Retorno de carro (Impresora).
\b	Retroceso (Backspace)
\f	Salto de página.
\t	Tabulación horizontal
\"	Comillas dobles (")
\'	Apóstrofe o comillas simples (')
****	Barra invertida o Contrabarra (\)
\a	Alerta (sonido).
\0	Carácter nulo o Fin de cadena de caracteres.
\ddd dígitos)	Constante octal (máximo tres)
\xddd	Constante hexadecimal (ídem)

Ejemplo: Uso de las Secuencias de Escape

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        printf("\"Uso de Comillas\"");
        printf("\tTabulación Horizontal");
        printf("\nSalto de Línea\n");
        printf("\¿Entre signos de interrogación\?\n");
        printf("\'Apóstrofo\'\n");
        printf("\\Contrabarra\b\b\b Retroceso");
        printf("\nFin de una cadena de\0 caracteres");
        printf("\nS\a o\a n\a i\a d\a o\a s\a\n");
        system(pause); /*De esta forma ejecutamos al comando pause del
S.O. para detener el programa, aparecerá en pantalla el mensaje de pulsar una
tecla para continuar sin necesidad de escribir dicho mensaje con la función
printf()*/
        return(0); /* Esta sentencia reemplaza al void en una función, pero a
diferencia retorna el valor que le indiquemos que puede ser una variable o una
constante*/
}
```

BIBLIOTECAS + FUNCIONES		
BIBLIOTECAS	**TRADUCCIÓN**	**FUNCIONES ASOCIADAS**
ctype.h	Operaciones Con caracteres	**tolower();** PASA UNA LETRA EN MAYÚSCULAS A MINÚSCULAS. **toupper();** PASA UNA LETRA EN MINÚSCULAS A MAYÚSCULAS.
stdio.h	Estándar de E/S	**scanf();** INGRESO DE DATOS DE DISTINTOS TIPOS, EXCEPTO LAS CADENAS DE CARACTERES COMPUESTAS, DADO QUE SOLO SOPORTA EL INGRESO DE UNA SOLA PALABRA. **printf();** MOSTRAR MENSAJES EN PANTALLA. **gets();** INGRESO DE CADENAS DE CARACTERES COMPUESTAS, O SEA, QUE PERMITE INGRESAR MAS DE UNA PALABRA. **puts();** MOSTRAR CADENAS DE CARACTERES. **fflush();** VACIA EL BUFFERS[15] DEL TECLADO.
stdlib.h	Biblioteca Estándar de Propósito General	**exit();** TERMINA LA EJECUCIÓN DEL PROGRAMA. **system();** EJECUTA UN COMANDO O PROCESO EXTERNO AL PROGRAMA QUE SE ESTA EJECUTANDO.
string.h	Cadenas de Caracteres	**strcat();** CONCATENA CADENAS. **strcmp();** COMPARA CADENAS. **strcpy();** COPIA CADENAS. **strlen();** CUENTA LOS CARACTERES DE UNA CADENA. **strlwr();** PASA UNA CADENA DE CARACTERES DE MAYUSCÚLAS A MINUSCÚLAS. **strupr();** PASA UNA CADENA DE CARACTERES DE MINÚSCULAS A MAYÚSCULAS.

[15] Un buffer es un registro de almacenamiento temporario y todos los dispositivos que pertenecen al Hardware lo tienen, y lo que hace la función **fflush()**, es la de vaciar o borrar toda la información que posea el buffer del teclado que no debemos utilizar, permitiendo ingresar solo él o los caracteres que han sido ingresados, solo se utiliza cuando trabajamos con caracteres o cadenas de caracteres, eliminando el residuo de información que haya quedado almacenado en el buffer del teclado, no es necesario utilizar esta función en el caso de que estemos trabajando con datos del tipo numéricos de cualquier tipo.

FUNCIONES MATEMÁTICAS EN LENGUAJE C[16]

El lenguaje C nos facilita una biblioteca de funciones matemáticas entre las que se incluyen las de uso más habitual como pueden ser: valor absoluto, potencia de un número elevado a otro, raíz cuadrada, funciones trigonométricas (seno, coseno, tangente, etc.), redondeo, exponenciación, logaritmo decimal, logaritmo neperiano y otras.

Para utilizar las funciones matemáticas indicadas a continuación es necesario incluir en la cabecera de nuestros programas la siguiente declaración:

#include <math.h>

El resultado de aplicar una función matemática es un valor numérico de tipo double (aunque el resultado puede ser un valor entero, internamente C lo considerará como un valor tipo decimal de doble precisión o tipo double)

Las funciones disponibles:

- **OPERACIONES BÁSICAS**

FUNCIÓN	PROPÓSITO
abs(valor); **labs(valor);**	RECUPERA EL VALOR ABSOLUTO DE UN VALOR INTEGRAL.
abs(valor); **fabs(valor);**	RECUPERA EL VALOR ABSOLUTO DE UN VALOR EN PUNTO FLOTANTE.
div(dividendo,divisor); **ldiv(dividendo,divisor);**	RECUPERA EL COCIENTE Y RESTO DE UNA DIVISIÓN.
fmod(dividendo,divisor);	RECUPERA EL RESTO DE UNA DIVISIÓN EN PUNTO FLOTANTE.

- **FUNCIONES EXPONENCIALES**

FUNCIÓN	PROPÓSITO
exp(valor);	CALCULA e ELEVADO A LA POTENCIA DADA.
exp2(valor);	CALCULA 2 ELEVADO A LA POTENCIA DADA.
log(valor);	CALCULA EL LOGARITMO NATURAL (BASE e).
log10(valor);	CALCULA EL LOGARITMO COMÚN (BASE 10).

[16] http://unn-lenguajec.blogspot.com.ar/2012/08/funciones-matematica-en-c.html

- **FUNCIONES POTENCIAS**

FUNCIÓN	PROPÓSITO
sqrt(valor);	CALCULA LA RAÍZ CUADRADA.
cbrt(valor);	CALCULA LA RAÍZ CÚBICA.
pow(base,exp);	CALCULA UN NÚMERO ELEVADO A LA POTENCIA DEL OTRO.

- **FUNCIONES TRIGONOMÉTRICAS**

FUNCIÓN	PROPÓSITO
sin(valor);	SENO
cos(valor);	COSENO
tan(valor);	TANGENTE
asin(valor);	ARCOSENO
acos(valor);	ARCOCOSENO
atan(valor);	ARCOTANGENTE
atan2(valor);	ARCOTANGENTE DE DOS ARGUMENTOS, USANDO EL SIGNO PARA DETERMINAR EL CUADRANTE

- **FUNCIONES HIPERBÓLICAS**

FUNCIÓN	PROPÓSITO
sinh(valor);	SENO HIPERBÓLICO
cosh(valor);	COSENO HIPERBÓLICO
tanh(valor);	TANGENTE HIPERBÓLICA

EJERCITACIÓN CON FUNCIONES DE CADENAS DE CARACTERES Y MATEMÁTICAS

1. **Vamos a utilizar algunas funciones menos habituales, como por ejemplo, las funciones ceil() y floor(), que cumplen la función de devolver el redondeo entero de un número. La diferencia entre ambas es que ceil() redondea al entero superior más próximo y floor() redondea al entero inferior más próximo. Tener en cuenta que el entero superior más próximo a un número negativo como – 5.75 es – 5 mientras que el entero superior más próximo a un número positivo como 5.75 es 6. Hay que recordar que, aunque estas funciones devuelven enteros, internamente el tipo de datos devuelto por estas funciones en C es double. Si se aplican estas funciones a un número entero, obtenemos ese mismo número, pero con formato double.**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main() {
   double M = -5.75;
   int N = 7;
   double Pi = 3.14159;
   printf ("El valor absoluto de M es %g y el de N es %g\n", fabs(M), fabs(N));
   printf ("El coseno de Pi es %g\n", cos(Pi));
   printf ("2 elevado al cubo vale %g\n", pow(2, 3));
   printf ("El numero pi redondeado con ceil vale %g\n", ceil(Pi));
   printf ("M redondeado con ceil vale %g\n", ceil(M));
   printf ("El numero pi redondeado con floor vale %g\n", floor(Pi));
   printf ("M redondeado con floor vale %g\n", floor(M));
   printf ("-M redondeado con floor vale %g\n", floor(-M));
   printf ("El numero e vale %g\n", exp(1));
   printf ("El logaritmo neperiano de e vale %g\n", log(exp(1)));
   printf ("El logaritmo decimal de 100 vale %g\n", log10(100));
   printf ("La raiz cuadrada de 81 vale %g\n", sqrt(81));
   return 0;
}
```

2. Averiguar el seno, coseno y tangente.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Pi 3.1416
main ()
{
  float numero, seno, coseno, tangente;
  numero=2*Pi;
  seno=sin(numero);
  coseno=cos(numero);
  tangente=tan(numero);
  printf("Los valores son:\nSeno:%f\nCoseno:%f\nTangente:%f",seno,
  coseno, tangente);
  system(“pause”);
}
```

3. Se deben ingresar 5 nombres en una matriz para su listado en pantalla.

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

```
   /*Creamos una matriz de caracteres para almacenar 5 nombres de hasta 19
   caracteres, porque al final de la cadena se agrega el caracter \0 que indica
   su fin*/
   char nombre[5][20] ;
   for(i=0; i<5; i++)
   {   fflush(stdin);
       scanf("%s", nombre[i]);
   }
   for(i=0; i<5; i++)
   {
   /*Mostramos los datos del vector uno por uno, con un sonido al final*/
       printf("\tNombre: %s\n\a", nombre[i]);
       system("pause");
  }
}
```

4. Utilización de las funciones de cadenas strcpy, strlen, strcat//

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main()
{
  char palabra1[80]; //Tamaño de la cadena.
  char palabra2[30]; //Tamaño de la cadena.
  system("cls");
  /*Copio la cadena de la derecha a la de la izquierda*/
  strcpy(palabra1,"buenos días");
  strcpy(palabra2,"señor");
  printf("\n palabra1 tiene una longitud de %2d y contiene
            <%s>\n", strlen(palabra1), palabra1);
  system("pause");
/*Agrego (concateno) un espacio al final de la cadena*/
  strcat(palabra1," "); //
  printf("\n palabra1 tiene una longitud de %2d y contiene
            <%s>\n",strlen(palabra1),palabra1);
  system("pause");
/*Concateno las cadenas*/
  strcat(palabra1, palabra2);
  printf("\npalabra1 tiene una longitud de %d y contiene
            <%s>\n", strlen(palabra1), palabra1);
  system("pause");
/*Concateno la palabra de la derecha a la cadena de la izquierda*/
  strcat(palabra1," y señora");
  printf("\npalabra1 tiene una longitud de %d y contiene
            <%s>\n", strlen(palabra1), palabra1);
  system("pause");  }
```

5. Este programa imprime las primeras diez potencias de 10 y las funciones trigonométricas seno, coseno, tangente en el dominio -1 y 1.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
main()
{
  double x=3.0 , y=0.0, raiz=16.0;
  double val=-1.0;
  int i=0; //Usamos esta variable para desplazarnos por las filas
  system("cls");
  do
  { /* devuelve base elevada a la potencia exp (base^exp) */
     printf("%f \n", pow(x,y));
     y++;
     i++;
  } while (y<11);
  printf("\nLa raíz cuadrada de %f es %f\n",  raiz, sqrt(raiz));
  system("pause");
  system("cls");
  printf("Calcular el seno, coseno y tangente del  dominio -1 y 1 \n\n\n");
  do
  {
       printf("\n");
       printf("Para: %f ", val);
       printf("Seno: %f ", sin(val));
       printf("Coseno: %f ",cos(val));
       printf("Tangente: %f\n", val, tan(val));
       printf("\n");
       val=val+0.1;
   } while(val<=1.0);
  system("pause");
  return (0);
}
```

6. Se tienen 3 usuarios: Juan tiene la clave 1, Pedro tiene la clave 2 y José la clave 3. Se pide un programa que salude al usuario luego de identificarlo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{       /*Variable para almacenar el nombre del Usuario*/
       char usuario[10];
       int clave=0;
```

```
	/*Vector de 3 cadenas*/
	 char puesto[3][15]={"Sr. Gerente", "Sr. Jefe", "Sr. Empleado"};
		char linea[26];
	 do
	 {
		printf("\nIngrese la clave del usuario:\t");
		scanf("%d", &clave);
		strcpy(linea, puesto[clave-1]);
		switch (clave)
		{
			case 1:
				strcat(linea," Juan");
				break;
			case 2:
				strcat(linea," Pedro");
				break;
			case 3:
				strcat(linea," José");
				break;
			default:
			{	printf("\nClave Incorrecta ..... FIN\n");
				system(“pause”);
			}
		}
		if ((clave>0) && (clave<4))
		{
			printf("\nBuen día %s\n", linea);
			system(“pause”);
		}
		}while (clave<4);
		return(0);
}
```

EJERCICIO TIPO

Hacer un programa que mediante un menú de opciones realice las siguientes acciones:

a. Ingresar los siguientes datos de un stock de materiales compuesto de 100 artículos: Código (Según el índice), Descripción (30 caracteres), Cantidad (Pueden ser negativa) y Precio Unitario.
b. Calcular el monto de cada artículo y mostrar el capital actual del Stock.
c. Buscar los datos de un artículo determinado por su nombre (Descripción), y mostrar a todos los datos incluyendo el monto.
d. Hacer los listados de los artículos existentes y faltantes, por separado, mostrar a todos los datos.
x. Salir del programa.

FUNCIONES UTILIZADAS PARA LA RESOLUCIÓN DEL PROBLEMA

fflush(stdin):	**Para vaciar el buffer del teclado.**
scanf():	**Para el ingreso de todo tipo de datos.**
gets():	**Para el ingreso de las cadenas de caracteres.**
printf():	**Para mostrar los distintos tipos de mensajes o para la ejecución de las secuencias de escape.**
strcpy():	**Para asignarle un valor a una cadena de caracteres.**
strcmp():	**Para comparar el contenido de dos cadenas de caracteres.**
strlen():	**Para calcular el tamaño de una cadena de caracteres.**
system():	**Nos permite ejecutar programas externos al programa que estamos utilizando.**
exit():	**Forzamos la salida de un programa.**
break:	**Rompemos la ejecución de un caso de la estructura switch().**
toupper():	**Convertimos un carácter en mayúsculas.**

RESOLUCIÓN DEL PROGRAMA:

Declaramos las bibliotecas internas del programa:

#include <stdio.h> //Biblioteca estándar de E/S.
#include <string.h> //Biblioteca para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Biblioteca estándar.
#include <ctype.h> //Biblioteca para la conversión de caracteres.

Declaramos las constantes:

#define M 100 //Defino la constante.

Inicio de la estructura principal del programa:

```
main()
{    //Primero declaramos todas las variables.
     int i; //Índice.
     int control=0; //Variable para controlar que opción del menú debemos hacer.
     char opcion='Z'; //Para seleccionar la opción del menú
     char auxop; //Para cargar en forma temporal la opción del menú seleccionada.
     int codigo[M]; //Para almacenar el número del código de artículo.
     char auxdes[30]; //Para cargar el forma temporal la cadena de caracteres a validar.
     char descripcion[M][30]; //Para almacenar el nombre del artículo.
     float pu[M]; //Para almacenar el precio unitario del artículo.
     int cantidad[M]; //Para almacenar la cantidad de artículos del stock.
     float auxprecio; //Para cargar en forma temporal el precio unitario.
     float monto[M]; //Para el almacenar el precio unitario.
     double capital=0; //Sirve como acumulador de los montos calculados.
```

Diseñamos el menú de opciones:

```
     while(opción!='X')
     {    system("cls"); //El proceso ejecutado por esta función borra la pantalla.
          //Menú de opciones.
          printf("\n\tMENU");
          printf("\nA. ALTAS");
          printf("\nB. CALCULOS");
          printf("\nC. BUSQUEDA");
          printf("\nD. LISTADO");
          printf("\nX. SALIR");
          printf("\nElegir Opcion:\t");
          fflush(stdin); //Vaciamos el buffer del teclado.
          scanf("%c", &auxop); //Seleccionamos la opción.
          opción=toupper(auxop); /*Pasamos la opción seleccionada a mayúsculas.*
```

Resolvemos cada opción del menú:

ALTAS: Ingreso de datos.

```
          switch(opción)
          {   case 'A':
                    if(control==0) /*Controlamos si esta hecho la opción A del menú.*/
                    {   control=1; /*A la variable control le asignamos el valor 1 para saber
                                       que vamos a realizar el ingreso de los datos.*/
```

```
        for(i=0; i<M; i=i+1)
        {    codigo[i]=i+1; //Ingresamos el código del artículo.
             printf("\nArticulo Nro:\t%i", codigo[i]);
             printf("\nNombre:\t");
             fflush(stdin);
             gets(auxdes); //Se puede usar: scanf("%s", auxdes);
             while(strlen(auxdes)>30)
             {    printf("Ingresar una nueva descripcion:\t");
                  fflush(stdin);
                  gets(auxdes);
             }
             strcpy(descripcion[i], auxdes);
             printf("\nCantidad:\t");
             scanf("%i", &cantidad[i]);
             printf("\nPrecio unitario:\t");
             scanf("%f", &auxprecio);
             pu[i]=auxprecio;
        }
   }
   break;
```

CÁLCULOS: Realizamos en este caso todos los cálculos pedidos en el programa.

```
case 'B':
   if(control==1) /*Controlamos si ya está hecha la opción A del menú.*/
   {    control=2; /*A la variable control le asignamos el valor 2 para saber
                     que vamos a realizar los cálculos del programa.*/
        for(i=0; i<M; i=i+1)
        {    monto[i]=pu[i]*cantidad[i]; //Cálculo el monto.
             capital=capital+monto[i]; //Cálculo el capital.
        }
        printf("\nEl capital existente es %8.2f", capital);
   }
   if(control==2)
        printf("El capital ya fue calculado!!!");
    else
        printf("Primero ingresamos los datos");
   break;
```

BÚSQUEDA: Por el nombre del artículo.

```
case 'C':
    if(control!=2)  /*Antes de hacer cada caso controlamos si los
                        casos A y B fueron realizados*/
    {   printf("Debemos hacer primero las opciones A y B del
        menu");
    }
    else
    {   printf("Buscar al siguiente articulo\n");
        printf("Nombre:\t");
        fflush(stdin);
        gets(auxdes); //se puede usar: scanf("%s", auxdes);
        while(strlen(auxdes)>30)
        {    printf("Ingresar una nueva descripcion:\t");
             fflush(stdin);
             gets(auxdes);
        }
        for(i=0; i<M; i=i+1)
        {    if(strcmp(descripcion[i], auxdes)==0)
             {    printf("\nCodigo:\t%i", codigo[i]);
                  printf("\nCantidad:\t%i", cantidad[i]);
                  printf("\tPrecio Unitario:\t%8.2f", pu[i]);
                  printf("\tMonto:\t%8.2f", monto[i]);
                  break;
             }
        }
        if(strcmp(descripcion[i], auxdes)!=0)
        {    printf("\nArticulo inexistente!!!");
        }
    }
    break;
```

LISTADOS: Bajo distintas condiciones.

```
case 'D':
    if(control!=2) /*Antes de hacer cada caso controlamos si los casos A y
                        B fueron realizados*/
    {   printf("Debemos hacer primero las opciones A y B del
                menu\n");
    }
    else
    {   printf("\nListado Articulos existentes");
        printf("\nCod  Descripcion        Cant   Precio  Total  \n");
```

```
                    for(i=0; i<M; i=i+1)
                    {    if(cantidad[i]>0)
                         {    printf("\n%3i", codigo[i]);
                              printf("%20s", descripcion[i]);
                              printf("%8i", cantidad[i]);
                              printf("%8.2f", pu[i]);
                              printf("%8.2f", monto[i]);
                         }
                    }
                    printf("\nListado Articulos faltantes");
                    printf("\nCod  Descripcion        Cant  Precio  Total  \n");
                    for(i=0; i<M; i=i+1)
                    {    if(cantidad[i]<=0)
                         {    printf("\n%3i", codigo[i]);
                              printf("%20s", descripcion[i]);
                              printf("%8i", cantidad[i]);
                              printf("%8.2f", pu[i]);
                              printf("%8.2f", monto[i]);
                         }
                    }
               }
               break;
          //SALIR: Opción obligatoria para salir del programa.
          case 'X':
               printf("Este es el fin del Programa...");
               system("pause");
               exit(0);
          /*Esta es la condición de falso de la estructura SWITCH(), todas las
            demás para que se cumplan tienen que ser verdaderas*/
          default:
               printf("OPCION INCORRECTA!!!");
     } //Fin del switch(opción).
     /*Con este procedimiento aparece el mensaje para todas las opciones,
       menos para la opción "A" de pulsar una tecla para continuar.*/
     if(opción!='A')
     {   printf("\n"); //Realiza un salto de línea.
         system("pause"); /*El proceso ejecutado por esta función hace una pausa
                            dentro del programa.*/
     }
} //Fin del while(opción!='Z').
return (0);
} //Fin del main()
```

PROCEDIMIENTOS FUNDAMENTALES

ABM

El diseño de programas cuenta con varios procedimientos que son un estándar dentro de la programación y se los conoce con el nombre de **ABM**.

Un **ABM** es la suma de varios procedimientos indispensables para el diseño de un algoritmo de programación, el cual nos permita realizar de forma adecuada a un programa.

Estos procedimientos están vinculados con cada uno de los caracteres que significa la sigla **ABM** y son los que se detallan a continuación:

- La letra **A** hace referencia a las **Altas** que nos indican que por medio de un programa vamos a ingresar o cargar datos.
- La letra **B**, nos indica que vamos a dar de **Baja** al o los datos, o sea eliminarlos, parcialmente o definitivamente, dependiendo esto del lenguaje de programación que estemos utilizando y del lugar en donde esos datos estén almacenados (un archivo o un espacio en la memoria RAM).
- La letra **M**, nos indicara que podremos **Modificar** o cambiar, al o los datos que fueron almacenados.

Existen otros dos procedimientos que aunque sus siglas no estén incorporadas en la sigla, también son un estándar dentro del ámbito de la programación, estos son:

- La **Búsqueda** del o los datos
- La realización de **Listados**, parciales o totales, de los datos ingresados.

Todo dato[17] para ser procesado por un programa va a estar contenido dentro de una estructura denominada registro, estos registros estarán formados por campos y cada uno de los campos pueden contener solo letras, solo números o bien cualquier tipo de caracteres (letras, números y símbolos).

Los datos pueden ser nombres, valores, números enteros o reales, caracteres, palabras, textos, fechas, etc.

Como ejemplo podríamos poner los datos de cualquiera de los alumnos que cursen en la facultad, en donde cada dato estaría contenido dentro de un campo y la suma de todos los datos que lo identifiquen, formarían un registro, como por ejemplo, nuestro nombre y apellido, DNI, edad, especialidad que cursemos, lugar donde vivimos, etc. Cada uno de esos datos es independiente el uno del otro, y alguno de ellos nos podría identificar de una manera unívoca a cada uno de nosotros, siendo alguno de ellos únicos e irrepetibles, como lo es nuestro DNI o el número de legajo que nos asignen en la institución educativa que estudiemos.

Obviamente que para la realización de un programa se necesitan datos, uno o varios, dado que si no existieran, no tendría ninguna utilidad.

[17] Ver Tipos de Datos, Pág.14

PROGRAMAS BASADOS EN ABM

HERRAMIENTAS DE PROGRAMACIÓN A UTILIZAR

BIBLIOTECAS:

stdio.h	**Biblioteca estándar de E/S.**
stdlib.h	**Biblioteca estándar.**
string.h	**Biblioteca para el manejo de las cadenas de caracteres.**

FUNCIONES:

break:	**Rompemos la ejecución de un caso de la estructura de control switch().**
exit():	**Forzamos la salida de un programa.**
fflush(stdin)	**Para vaciar el buffer del teclado. Linux setbuf(stdin, NULL)**
gets()	**Para el ingreso de las cadenas de caracteres compuesta.**
printf()	**Para mostrar los distintos tipos de mensajes o para la ejecución de las secuencias de escape.**
scanf()	**Para el ingreso de todo tipo de datos.**
strcmp()	**Para comparar el contenido de dos cadenas de caracteres.**
strcpy()	**Para asignarle un valor a una cadena de caracteres.**
system()	**Nos permite ejecutar programas externos al programa que estamos utilizando.**

ESTRUCTURAS DE PROGRAMACION

do-while()	**Estructura de control repetitiva: HACER-MIENTRAS**
for()	**Estructura de control repetitiva: PARA-HACER**
while()	**Estructura de control repetitiva: MIENTRAS-HACER**
if()	**Estructura de control condicional simple.**
if() - else	**Estructura de control condicional compuesta.**
switch()	**Estructura de control condicional múltiple.**

PROCEDIMIENTO DE ALTAS:

Los procedimientos de **Altas** se realizan en todos los programas, dado que siempre se deben ingresar o cargar datos, de lo contrario no se podrían realizar los demás procedimientos.

Enunciado 1: Ingresar el legajo, apellido y edad de los 50 alumnos de este curso de Informática. En este ejercicios utilizaremos la estructura repetitiva MIENTRAS-HACER.

1. Declaramos las bibliotecas internas del programa:

```
#include <stdio.h> //Biblioteca estándar de E/S.
#include <string.h> //Biblioteca para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Biblioteca estándar.
```

2. Declaramos las constantes del programa:

```
#define M 50 /*Definimos la constante (recordemos que el nombre de las constantes por norma general, siempre se declaran con letras mayúsculas para marcar una diferencia con respecto a las variables, que siempre se deberán declarar con letras minúsculas).*/
```

3. Iniciamos la estructura principal del programa:

```
main()
{
```

4. Declaramos las variables:

```
    float legajo[M]; /*Declaramos un vector de números reales para almacenar los legajos.*/
    char apellido[M][30]; /*Declaramos una matriz de caracteres para almacenar los apellidos.*/
    int edad[M]; //Declaramos un vector entero para guardar las edades.
    int c=0; //Declaramos el contador.
```

5. Resolvemos el ejercicio (en este caso solo ingresamos los datos):

```
    while(c<M) //Mientras que el contador sea menor al valor de M hacer.
    {
        system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
        printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso de los legajos.*/
        scanf("%f", &legajo[c]); //Ingresamos los números de legajo.
```

```
            printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el
            ingreso de los apellidos.*/
            fflush(stdin); //Vaciamos el buffer del teclado.
            gets(apellido[c]); //Ingresamos los apellidos.
            printf("Ingresar Edad:\t"); /*Mostramos el mensaje para el
            ingreso de las edades.*/
            scanf("%i", &edad[c]); //Ingresamos la edad.
            c=c+1; //Se incrementa el contador de uno en uno.
            if(c==M) //Si el contador es igual a M entonces.
            {
                  printf("\nFin de la carga de datos"); /*Mostramos el mensaje
                  correspondiente*/
                  printf("\n"); //Realizamos un salto de línea.
                  system("pause"); /*Realizamos una pausa mostrando el
                  siguiente mensaje "Pulsar una tecla para continuar".*/
            } //Fin del if()
      } //Fin del while()
} //Fin del main()
```

Comentario: Como verán en este ejercicio solo hicimos el procedimiento para el ingreso de los datos, lo lógico sería poder mostrar los datos ingresados.

PROCEDIMIENTO DE LISTADOS:

Los procedimientos para la realización de los Listados o el de mostrar los datos, son fundamentales para todo proceso, dado que si no los haríamos no podríamos ver los resultados que nos brinda un proceso.

Enunciado 2: Ingresar el legajo, apellido y edad de los 50 alumnos de este curso de Informática. Mostrarlos por alumno y uno debajo del otro. En este ejercicio usaremos la estructura repetitiva HACER-MIENTRAS.

1. Declaramos las bibliotecas internas del programa:

```
#include <stdio.h> //Biblioteca estándar de E/S.
#include <string.h> //Biblioteca para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Biblioteca estándar.
```

2. Declaramos las constantes del programa:

```
#define M 50 /*Definimos la constante (recordemos que el nombre de las constantes
por convención, siempre se declaran con letras mayúsculas para marcar una diferencia
con respecto a las variables, que se deberán declarar con letras minúsculas).*/
```

3. Iniciamos la estructura principal del programa:

```
main()
{
```

4. Declaramos las variables:

```
float legajo[M]; /*Declaramos un vector de números reales para almacenar los legajos.*/
char apellido[M][30]; /*Declaramos una matriz de caracteres para almacenar los apellidos.*/
int edad[M]; //Declaramos un vector entero para guardar las edades.
int c=0; //Declaramos el contador.
```

5. Resolvemos el ejercicio (en este caso solo ingresamos los datos):

```
do //Hacer lo siguiente.
{
    system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso de los legajos.*/
    scanf("%f", &legajo[c]); //Ingresamos los números de legajo.
    printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el ingreso de los apellidos.*/
    fflush(stdin); //Vaciamos el buffer del teclado.
    gets(apellido[c]); //Ingresamos los apellidos.
    printf("Ingresar Edad:\t"); /*Mostramos el mensaje para el ingreso de las edades.*/
    scanf("%i", &edad[c]); //Ingresamos la edad.
    c=c+1; //Se incrementa el contador de uno en uno.
    if(c==M) //Si el contador es igual a M entonces.
    {
        printf("\nFin de la carga de datos"); /*Mostramos el mensaje correspondiente*/
        printf("\n"); //Realizamos un salto de línea.
        system("pause"); /*Realizamos una pausa mostrando el siguiente mensaje "Pulsar una tecla para continuar".*/
    } //Fin del if()
}while(c<M); //Mientras que el contador sea menor al valor de M hacer.
```

6. Mostramos los resultados (en este caso mostramos los datos ingresados en forma de lista o de forma individual):

```
c=0; //Reiniciamos el contador.
system("cls"); /*Borramos todo lo que estuviera previamente escrito
en la pantalla por el programa.*/
printf("Listado de los alumnos ingresados");
do //Hacer lo siguiente.
{
        printf("\n%.0f", legajo[c]); //Mostramos el legajo.
        printf("\t%s", apellido[c]); //Mostramos el apellido.
        printf("\t%i", edad[c]); //Mostramos la edad.
        c=c+1; //Se incrementa el contador de uno en uno.
 }while(c<M); //Mientras que el contador sea menor al valor de M hacer.
printf("\n"); //Realizamos un salto de línea.
system("pause"); /*Realizamos una pausa mostrando el siguiente
mensaje "Pulsar una tecla para continuar".
}//Fin del main()
```

PROCEDIMIENTO DE BUSQUEDA:

El procedimiento para realizar una Búsqueda nos permite buscar los datos de un elemento en particular o de un grupo de elementos.

Enunciado 3: Ingresar el legajo, apellido y edad de los 50 alumnos de este curso de Informática. Mostrar los datos de un alumno en particular. En este ejercicio usaremos la estructura repetitiva PARA-HACER.

1. Declaramos las bibliotecas internas del programa:

```
#include <stdio.h> //Biblioteca estándar de E/S.
#include <string.h> //Biblioteca para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Biblioteca estándar.
```

2. Declaramos las constantes del programa:

#define M 50 /*Definimos la constante (recordemos que el nombre de las constantes por convención, siempre se declaran con letras mayúsculas para marcar una diferencia con respecto a las variables, que se deberán declarar con letras minúsculas).*/

3. Iniciamos la estructura principal del programa:

```
main()
{
```

4. Declaramos las variables:

```
float legajo[M]; /*Declaramos un vector de números reales para almacenar los legajos.*/
char apellido[M][30]; /*Declaramos una matriz de caracteres para almacenar los apellidos.*/
int edad[M]; //Declaramos un vector entero para guardar las edades.
int c=0; //Declaramos el contador.
int control=0; /*Esta variable la usaremos para distintas acciones del programa.*/
float auxleg; //Variable para la búsqueda del dato por el numero de legajo.
char auxape[30]; /*Vector de caracteres para la búsqueda del dato por el apellido*/
```

5. Resolvemos el ejercicio (en este caso solo ingresamos los datos):

```
for(c=0; c<M; c=c+1) /*Para-Hacer desde el contador en cero; mientras el contador sea menor al valor de M; incrementando el contador de uno en uno.*/
{
    system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso de los legajos.*/
    scanf("%f", &legajo[c]); //Ingresamos los números de legajo.
    printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el ingreso de los apellidos.*/
    fflush(stdin); //Vaciamos el buffer del teclado.
    gets(apellido[c]); //Ingresamos los apellidos.
    printf("Ingresar Edad:\t"); /*Mostramos el mensaje para el ingreso de las edades.*/
    scanf("%i", &edad[c]); //Ingresamos la edad.
    c=c+1; //Se incrementa el contador de uno en uno.
    if(c==M) //Si el contador es igual a M entonces.
    {
        printf("\nFin de la carga de datos"); /*Mostramos el mensaje correspondiente*/
        printf("\n"); //Realizamos un salto de línea.
        system("pause"); /*Realizamos una pausa mostrando el siguiente mensaje "Pulsar una tecla para continuar".*/
    } //Fin del if()
} //Fin del for()
```

6. Buscamos un dato (en este caso mostramos los datos buscados con la utilización de dos métodos diferentes):

```
system("cls"); /*Borramos todo lo que estuviera previamente escrito
en la pantalla por el programa.*/
```

Primer método: realizaremos la búsqueda por el número de legajo:

```
printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el
ingreso de los legajos.*/
scanf("%f", &auxleg); //Ingresamos el números de legajo a buscar.
for(c=0; c<M; c=c+1) /*Para-Hacer desde el contador en cero; mientras el
contador sea menor al valor de M; incrementando el contador de uno en uno.*/
{
        if(auxleg==legajo[c]) /*Si el legajo del alumno es igual a la
        variable auxleg entonces nos muestra el apellido y la edad.*/
        {
                printf("\n%s", apellido[c]); //Mostramos el apellido.
                printf("\t%i", edad[c]); //Mostramos la edad.

                control=1; /*Si se encontró el legajo ponemos la variable
                control con un valor distinto a cero, para indicarle al
                programa que el dato fue encontrado*/
        } //Fin del if()
} //Fin del for()
```

Segundo método: realizaremos la búsqueda por el apellido:

```
        printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el
        ingreso del apellido*/
        fflush(stdin); //Vaciamos el buffer del teclado.
        gets(auxape); //Ingresamos el apellido a buscar.
        for(c=0; c<M; c=c+1) /*Para-Hacer desde el contador en cero; mientras
        el contador sea menor al valor de M; incrementando el contador de uno
        en uno.*/
        {
        if(strcmp(auxape, apellido[c])==0) /*Si el apellido del alumno es
        igual a la variable auxiliar entonces nos muestra el legajo y la edad.*/
        {
                printf("\n%.0f", legajo[c]); //Mostramos el legajo.
                printf("\t%i", edad[c]); //Mostramos la edad.
                control=1; /*Si se encontró al apellido ponemos la variable
                control con un valor distinto a cero, para indicarle al
                programa que el dato fue encontrado
        } //Fin del if()
 } //Fin del for()
```

Independientemente del método de búsqueda que utilicemos, controlamos si el dato fue encontrado o no:

```
        if(control==1) /*Controlamos que la variable control tenga el valor
        de uno para dar entonces por terminada la búsqueda.*/
        {
                printf("\nFin de la busqueda…"); /*Mostramos el mensaje
                correspondiente*/
                exit(); /*Si los datos fueron encontrados, salimos del
                programa.*/
        }
        else
        {
                if(c==M) /*Si el contador es igual a M quiere decir el los
                datos no fueron encontrados y entonces damos aviso del fin
                de la búsqueda*/
                {
                        printf("\nDatos no encontrados…"); /*Mostramos
                        el mensaje correspondiente*/
                }
        } //Fin del if()

    printf("\n"); //Realizamos un salto de línea.
    system("pause"); /*Realizamos una pausa mostrando el siguiente mensaje
    que "Pulsar una tecla para continuar".*/
}//Fin del main()
```

PROCEDIMIENTO DE MODIFICACIÓN:

El procedimiento de la modificación de los datos, nos permite cambiar todos o algunos de los datos ingresados de acuerdo a los elementos seleccionados.

Enunciado 4: Ingresar el legajo, apellido y edad de los 50 alumnos de este curso de Informática. Modificar todos los datos de un alumno en particular. En este ejercicio usaremos las estructuras repetitivas PARA-HACER y MIENTRAS-HACER.

1. Declaramos las bibliotecas internas del programa:

```
#include <stdio.h> //Biblioteca estándar de E/S.
#include <string.h> //Biblioteca para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Biblioteca estándar.
```

2. Declaramos las constantes del programa:

```
#define M 50 /*Definimos la constante (recordemos que el nombre de las constantes
por convención, siempre se declaran con letras mayúsculas para marcar una diferencia
con respecto a las variables, que se deberán declarar con letras minúsculas).*/
```

3. Iniciamos la estructura principal del programa:

```
main()
{
```

4. Declaramos las variables:

```
float legajo[M]; /*Declaramos un vector de números reales para almacenar los legajos.*/
char apellido[M][30]; /*Declaramos una matriz de caracteres para almacenar los apellidos.*/
int edad[M]; //Declaramos un vector entero para guardar las edades.
int c=0; //Declaramos el contador.
int control=0; /*Esta variable la usaremos para distintas acciones del programa.*/
float auxleg; //Variable para la búsqueda del dato por el numero de legajo.
char auxape[30]; /*Vector de caracteres para la búsqueda del dato por el apellido*/
```

5. Resolvemos el ejercicio (en este caso solo ingresamos los datos):

```
for(c=0; c<M; c=c+1) /*Para hacer desde el contador en cero, mientras el contador sea menor al valor de M e incrementando el contador de uno en uno.*/
{
    system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso de los legajos.*/
    scanf("%f", &legajo[c]); //Ingresamos los números de legajo.
    printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el ingreso de los apellidos.*/
    fflush(stdin); //Vaciamos el buffer del teclado.
    gets(apellido[c]); //Ingresamos los apellidos.
    printf("Ingresar Edad:\t"); /*Mostramos el mensaje para el ingreso de las edades.*/
    scanf("%i", &edad[c]); //Ingresamos la edad.
    c=c+1; //Se incrementa el contador de uno en uno.
    if(c==M) //Si el contador es igual a M entonces.
    {
        printf("\nFin de la carga de datos"); /*Mostramos el mensaje correspondiente*/
        printf("\n"); //Realizamos un salto de línea.
        system("pause"); /*Realizamos una pausa mostrando el siguiente mensaje que "Pulsar una tecla para continuar".*/
    } //Fin del if()
} //Fin del for()
```

6. Modificación del o los datos utilizando los distintos métodos de búsqueda aprendidos y modificamos los datos que sean necesarios cambiar):

```
system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
while(c<M) /*Mientras-hacer, desde el contador en cero (antes de esta línea int c=0;), mientras el contador sea menor al valor de M hacer (en ésta línea while(c<M)), e incrementando el contador de uno en uno (después de ésta línea c=c+1;).*/
{
```

Primer método: realizaremos la búsqueda por el número de legajo:

```
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso de los legajos.*/
    scanf("%f", &auxleg); //Ingresamos el número de legajo.
    if(auxleg==legajo[c]) /*Si el legajo del alumno es igual a la variable auxleg entonces nos muestra el apellido y la edad.*/
    {
        printf("Modificamos el Legajo:\t"); /*Mostramos el mensaje para modificar el legajo.*/
        scanf("%f", &legajo[c]); //Ingresamos el nuevo legajo.
        printf("Modificamos el Apellido:\t"); /*Mostramos el mensaje para modificar el apellido.*/
        fflush(stdin); //Vaciamos el buffer del teclado.
        gets(auxape); //Ingresamos el nuevo apellido.
        strcpy(apellido[c], auxape); /*Le asignamos (copiamos) el nuevo apellido a la matriz de caracteres correspondiente.*/
        printf("Modificamos la Edad:\t"); /*Mostramos el mensaje para modificar la edad.*/
        scanf("%i", &edad[c]); //Ingresamos la nueva edad.
        control=1; /*Si se encontró al apellido ponemos la variable control con un valor distinto a cero, para indicarle al programa que el dato fue encontrado*/
    } //Fin del if()
```

Segundo método: realizaremos la búsqueda por el apellido:

```
    printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el ingreso del apellido*/
    fflush(stdin); //Vaciamos el buffer del teclado.
    gets(auxape); //Ingresamos el apellido.
    if(strcmp(auxape, apellido[c])==0) /*Si el apellido del alumno es igual a la variable auxape entonces nos muestra el legajo y la edad.*/
    {
        printf("Modificamos el Legajo:\t"); /*Mostramos el mensaje para modificar el legajo.*/
        scanf("%f", &legajo[c]); //Ingresamos el nuevo legajo.
```

```
                printf("Modificamos el Apellido:\t"); /*Mostramos el
                mensaje para modificar el apellido.*/
                fflush(stdin); //Vaciamos el buffer del teclado.
                gets(auxape); //Ingresamos el nuevo apellido.
                strcpy(apellido[c], auxape); /*Le asignamos (copiamos)
                el nuevo apellido a la matriz de caracteres correspondiente.*/
                printf("Modificamos la Edad:\t"); /*Mostramos el mensaje
                para modificar la edad.*/
                scanf("%i", &edad[c]); //Ingresamos la nueva edad.
                control=1; /*Si se encontró al apellido ponemos la variable
                control con un valor distinto a cero, para indicarle al
                programa que el dato fue encontrado*/
        } //Fin del if()
```

Independientemente del método de búsqueda que utilicemos, controlamos si el dato fue encontrado:

```
        if(control==1) /*Controlamos que la variable control tenga el valor
        1 para dar por terminada la búsqueda.*/
        {
                printf("\nFin de la modificación"); /*Mostramos el
                mensaje correspondiente*/
                exit(); /*Si los datos fueron encontrados, salimos del
                programa.*/
        }
        else
        {
                if(c==M) /*Si el contador es igual a M quiere decir el los
                datos no fueron encontrados y entonces damos aviso del fin
                de la búsqueda*/
                {
                        printf("\nDatos no encontrados…"); /*Mostramos
                        el mensaje correspondiente*/
                }
        } //Fin del if()
        c=c+1;
    } //Fin del while()
    printf("\n"); //Realizamos un salto de línea.
    system("pause"); /*Realizamos una pausa mostrando el siguiente mensaje
    que "Pulsar una tecla para continuar".*/
}//Fin del main()
```

PROCEDIMIENTO DE BAJAS:

El procedimiento de dar de baja uno o varios datos, consiste en eliminar o borrar los datos del elemento buscado, dado que el lugar que estaba ocupando el dato, sigue existiendo físicamente dentro de un archivo o de un espacio en memoria, como ocurre con los arreglos. Dar de Baja significara entonces, que si los datos son numéricos se los pondrá en cero y si es una cadena de caracteres, la dejaremos vacía, justamente de caracteres.

Enunciado 5: Ingresar el legajo, apellido y edad de los 50 alumnos de este curso de Informática. Eliminar los datos de un alumno en particular. En este ejercicio usaremos las estructuras repetitivas PARA-HACER y HACER-MIENTRAS.

1. Declaramos las bibliotecas internas del programa:

#include <stdio.h> //Biblioteca estándar de E/S.
#include <string.h> //Biblioteca para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Biblioteca estándar.

2. Declaramos las constantes del programa:

#define M 50 /*Definimos la constante (recordemos que el nombre de las constantes por convención, siempre se declaran con letras mayúsculas para marcar una diferencia con respecto a las variables, que se deberán declarar con letras minúsculas).*/

3. Iniciamos la estructura principal del programa:

```
main()
{
```

4. Declaramos las variables:

```
float legajo[M]; /*Declaramos un vector de números reales para
almacenar los legajos.*/
char apellido[M][30]; /*Declaramos una matriz de caracteres para
almacenar los apellidos.*/
int edad[M]; //Declaramos un vector entero para guardar las edades.
int c=0; //Declaramos el contador.
int control=0; /*Esta variable la usaremos para distintas acciones del
programa.*/
float auxleg; //Variable para la búsqueda del dato por el número de legajo.
char auxape[30]; /*Vector de caracteres para la búsqueda del dato por el
apellido*/
```

5. Resolvemos el ejercicio (en este caso solo ingresamos los datos):

```
for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero, mientras el contador sea menor al valor de M e incrementando el contador de uno en uno.*/
{
    system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso de los legajos.*/
    scanf("%f", &legajo[c]); //Ingresamos los números de legajo.
    printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el ingreso de los apellidos.*/
    fflush(stdin); //Vaciamos el buffer del teclado.
    gets(apellido[c]); //Ingresamos los apellidos.
    printf("Ingresar Edad:\t"); /*Mostramos el mensaje para el ingreso de las edades.*/
    scanf("%i", &edad[c]); //Ingresamos la edad.
    c=c+1; //Se incrementa el contador de uno en uno.
    if(c==M) //Si el contador es igual a M entonces.
    {
        printf("\nFin de la carga de datos"); //Mostramos el mensaje correspondiente*/
        printf("\n"); //Realizamos un salto de línea.
        system("pause"); /*Realizamos una pausa mostrando el siguiente mensaje que "Pulsar una tecla para continuar".*/
    } //Fin del if()
} //Fin del for()
```

6. Daremos de baja los datos ingresados utilizando los distintos métodos de búsqueda aprendidos y los eliminaremos:

```
system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
do //Hacer lo siguiente.
{
```

Primer método: realizaremos la búsqueda por el número de legajo:

```
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso de los legajos.*/
    scanf("%f", &auxleg); //Ingresamos el número de legajo.
    if(auxleg==legajo[c]) /*Si el legajo del alumno es igual a la variable auxleg, limpiamos el legajo, el apellido y la edad.*/
    {
        legajo[c]=0; //Ponemos en cero al número de legajo.
        strcpy(apellido[c], " "); /*La doble comilla le asigna un espacio en blanco a la cadena de caracteres.*/
        edad[c]=0; //Ponemos en cero la edad.
```

```
                control=1; /*Si se encontró al legajo ponemos la variable
                control con un valor distinto a cero, para indicarle al
                programa que el dato fue encontrado*/
        } //Fin del if()
```

Segundo método: realizaremos la búsqueda por el apellido:

```
        printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el
        ingreso del apellido*/
        fflush(stdin); //Vaciamos el buffer del teclado.
        gets(auxape); //Ingresamos el apellido.
        if(strcmp(auxape, apellido[c])==0) /*Si el apellido del alumno es
        igual a la variable auxape, limpiamos el legajo, el apellido y la edad.*/
        {
                legajo[c]=0; //Ponemos en cero al número de legajo.
                strcpy(apellido[c], " "); /*La doble comilla le asigna un
                espacio en blanco a la cadena de caracteres.*/
                edad[c]=0; //Ponemos en cero la edad.
                control=1; /*Si se encontró al legajo ponemos la variable
                control con un valor distinto a cero, para indicarle al
                programa que el dato fue encontrado*/
        } //Fin del if()
```

Independientemente del método de búsqueda que utilicemos, controlamos si el dato fue encontrado:

```
        if(control==1) /*Controlamos que la variable control tenga el valor
        uno para dar por terminada la búsqueda.*/
        {
                printf("\nAlumno dado de Baja"); /*Mostramos el
                mensaje correspondiente*/
                exit(); /*Si los datos fueron encontrados, salimos del
                programa.*/
        }
        else
        {
                if(c==M) /*Si el contador es igual a M significa que los
                datos no fueron encontrados y damos aviso del fin de la
                búsqueda*/
                {
                        printf("\nDatos no encontrados…"); /*Mostramos
                        el mensaje correspondiente*/
                }
        } //Fin del if()
        c=c+1;
    } while(c<M); //Mientras que el contador sea menor al valor de M hacer.
    printf("\n"); //Realizamos un salto de línea.
    system("pause"); /*Realizamos una pausa mostrando el siguiente mensaje
    que "Pulsar una tecla para continuar".*/
}//Fin del main()
```

Los procedimientos que pertenecen a un **ABM**, dependen de que se haya realizado el procedimiento de **Altas**, dado que si necesitamos darlos de **Baja**, **Modificarlo**, **Buscarlo** o **Listarlo**, siempre se deberá realizar las **Altas** primero. Por ese motivo siempre es conveniente para hacer un procedimiento completo de un **ABM**, utilizar la estructura condicional múltiple, la cual nos permite realizar un menú de opciones.

Veamos entonces, como quedarían todos estos los procedimientos si los utilizamos en un mismo programa, de acuerdo al ejercicio del siguiente enunciado:

Enunciado 6: Se debe hacer un menú de opciones para realizar las siguientes acciones:

1) Ingresar el legajo, apellido y edad de los 50 alumnos de este curso de Informática.
2) Mostrarlos por alumno y uno debajo del otro.
3) Mostrar los datos de un alumno en particular.
4) Modificar a todos los datos de un alumno en particular.
5) Eliminar los datos de un alumno en particular.
6) Mostrar la cantidad existente de alumnos y la cantidad que fueron eliminados (este procedimiento lo agregaremos para ver el uso de los contadores).
7) Salir del programa (todo menú de opciones debe tener una opción para salir del programa).

1. Declaramos las bibliotecas internas del programa:

```
#include <stdio.h> //Biblioteca estándar de E/S.
#include <string.h> //Biblioteca para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Biblioteca estándar.
```

2. Declaramos las constantes del programa:

```
#define M 50 /*Definimos la constante (recordemos que el nombre de las constantes por convención, siempre se declaran con letras mayúsculas para marcar una diferencia con respecto a las variables, que se deberán declarar con letras minúsculas).*/
```

3. Iniciamos la estructura principal del programa:

```
main()
{
```

4. Declaramos las variables:

```
    float legajo[M]; /*Declaramos un vector de números reales para
    almacenar los legajos.*/
    char apellido[M][30]; /*Declaramos una matriz de caracteres para
    almacenar los apellidos.*/
```

int edad[M]; //Declaramos un vector entero para guardar las edades.
int c=0; //Declaramos el contador.
int control=0; /*Esta variable la usaremos distintas acciones del programa.*/
float auxleg; //Variable para la búsqueda del dato por el número de legajo.
char auxape[30]; /*Vector de caracteres para la búsqueda del dato por el apellido*/
int contador=0; /*Este contador lo utilizaremos para contar la cantidad total de alumnos.*/
int bajas=0; /*Este contador lo utilizaremos para contar la cantidad de alumnos que fueron eliminados.*/
int opción=0; /*Cuando trabajamos con un menú de opciones, será necesario utilizar una variable que nos permita seleccionar una de las opciones del mismo. Se inicializa con un valor distinto de siete, ya que con ese valor en este ejercicio se sale del programa.*/

5. **Creamos el Menú de Opciones utilizando la estructura repetitiva MIENTRAS-HACER, aunque también la podríamos crear con la estructura HACER-MIENTRAS, pero nunca la podríamos hacer con la estructura PARA-HACER que es especial para cuando trabajamos con arreglos:**

while(opcion!=7) /*Mientras que la opción elegida sea distinta de 7 ingresamos al menú de opciones*/
{
system("cls"); /*Borramos todo lo que estuviera previamente escrito en la pantalla por el programa.*/
printf("MENU DE OPCIONES"); //Mostramos el encabezado del menú.
printf("\n1. ALTAS");
printf("\n2. LISTADO");
printf("\n3. BUSQUEDA");
printf("\n4. MODIFICACION");
printf("\n5. BAJAS");
printf("\n6. CONTADORES");
printf("\n7. SALIR");
/*Leemos las opciones del menú para determinar la acción a realizar.*/
printf("\nElegir opcion:\t");
scanf("%i", &opción);

6. **Resolvemos cada una de las opciones del menú utilizando la estructura condicional múltiple:**

switch(opción) /*Iniciamos la estructura condicional múltiple teniendo en cuenta el valor de la variable opcion.*/
{
case 1: //Realizamos el procedimiento de Altas.

```
for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero,
mientras el contador sea menor al valor de M e incrementando el contador
de uno en uno.*/
{
        printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el
        ingreso de los legajos.*/
        scanf("%f", &legajo[c]); /*Ingresamos los números de
        legajo.*/
        printf("Ingresar Apellido:\t"); /*Mostramos el mensaje para el
        ingreso del apellido.*/
        fflush(stdin); //Vaciamos el buffer del teclado.
        gets(apellido[c]); //Ingresamos los apellidos.
        printf("Ingresar Edad:\t"); /*Mostramos el mensaje para el
        ingreso de las edades.*/
        scanf("%i", &edad[c]); //Ingresamos la edad.
} //Fin del for()
break; /*Rompemos el caso o damos por finalizado el procedimiento.*/

case 2: //Realizamos el procedimiento de Listado
    printf("Listado de los alumnos ingresados");
    for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero,
    mientras el contador sea menor al valor de M e incrementando el
    contador de uno en uno.*/
    {
        printf("\n%.0f", legajo[c]); //Mostramos el legajo.
        printf("\t%s", apellido[c]); //Mostramos el apellido.
        printf("\t%i", edad[c]); //Mostramos la legajo.
    } //Fin del for()
    break; /*Rompemos el caso o damos por finalizado el
    procedimiento.*/

 case 3: /*Realizamos el procedimiento de Búsqueda, siempre es
 conveniente hacerla por uno de los datos que identifique de manera
 única al elemento buscado*/
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso del
    legajo a buscar.*/
    scanf("%f", &auxleg); /*Ingresamos el número de legajo.*/
    for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero,
    mientras el contador sea menor al valor de M e incrementando
    el contador de uno en uno.*/
    {
```

```
            if(auxleg==legajo[c]) /*Si el legajo del alumno es igual a la
            variable auxleg entonces se muestra el apellido y la edad.*/
            {
                  printf("\n%s", apellido[c]); //Mostramos el apellido.
                  printf("\t%i", edad[c]); //Mostramos la edad.
                  control=1; /*Si se encontró al legajo ponemos la
                  variable control con un valor distinto a cero, para
                  indicarle al programa que el dato fue encontrado*/
            } //Fin del if()
      } //Fin del for()
      break; /*Rompemos el caso o damos por finalizado el procedimiento.*/

case 4: /*Realizamos el procedimiento de Modificación. Siempre es
conveniente hacer la búsqueda del dato a modificar por el dato que
identifique de manera única al elemento buscado.*/
      printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso del
      legajo a buscar.*/
      scanf("%f", &auxleg); /*Ingresamos el número de legajo.*/
      for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero,
      mientras el contador sea menor al valor de M e incrementando
      el contador de uno en uno.*/
      {
            if(auxleg==legajo[c]) /*Si el legajo del alumno es igual a la
            variable auxleg entonces se muestran los mensajes para
            modificar el legajo, el apellido y la edad.*/
            {
                  printf("Modificamos el Legajo:\t"); /*Mostramos el
                  mensaje para modificar el legajo.*/
                  scanf("%f", &legajo[c]); /*Ingresamos el nuevo
                  legajo.*/
                  printf("Modificamos el Apellido:\t"); /*Mostramos el
                  mensaje para modificar el apellido.*/
                  fflush(stdin); //Vaciamos el buffer del teclado.
                  gets(auxape); //Ingresamos el nuevo apellido.
                  strcpy(apellido[c], auxape); /*Le asignamos (copiamos) el
                  nuevo apellido a la matriz de caracteres correspondiente.*/
                  printf("Modificamos la Edad:\t"); /*Mostramos el mensaje
                  para modificar la edad.*/
                  scanf("%i", &edad[c]); //Ingresamos la nueva edad.
                  control=1; /*Como se encontró al apellido ponemos la
                  variable control con un valor distinto a cero, para
                  indicarle al programa que el dato fue encontrado. */
            } //Fin del if()
      } //Fin del for()
      break; /*Rompemos el caso o damos por finalizado el procedimiento.*/
```

```
case 5: /*Realizamos el procedimiento de Bajas. Siempre es
conveniente hacer la búsqueda del dato a dar de baja por el dato que
identifique de manera única al elemento buscado.*/
    printf("Ingresar Legajo:\t"); /*Mostramos el mensaje para el ingreso del
    legajo a buscar.*/
    scanf("%f", &auxleg); /*Ingresamos el número de legajo.*/
    for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero,
    mientras el contador sea menor al valor de M e incrementando el
    contador de uno en uno.*/
    {
        if(auxleg==legajo[c]) /*Si el legajo del alumno es igual a la
        variable auxleg, limpiamos el legajo, el apellido y la edad.*/
        {
            legajo[c]=0; //Ponemos en cero al número de legajo.
            strcpy(apellido[c], " "); /*La doble comilla le asigna
            un espacio en blanco a la cadena de caracteres.*/
            edad[c]=0; //Ponemos en cero la edad.
            control=1; /*Como se encontró al legajo ponemos la
            variable control con un valor distinto a cero, para
            indicarle al programa que el dato fue encontrado.*/
        } //Fin del if()
    } //Fin del for()
    break; /*Rompemos el caso o damos por finalizado el procedimiento.*/

case 6: /*Realizamos los procedimientos para el uso de los contadores.*/
    printf("Primero contamos a todos los alumnos");
    for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero,
    mientras el contador sea menor al valor de M e incrementando el
    contador de uno en uno.*/
    {
        if(legajo[c]!=0) /*Si el número de legajo es distinto de cero
        entonces.*/
        {
            contador=contador+1; //Contamos
        } //Fin del if()
    } //Fin del for()
    printf("\nCantidad existente de alumnos:\t%i", contador);
    //Mostramos el mensaje correspondiente.
    printf("Ahora contamos a los alumnos dados de Baja…");
    for(c=0; c<M; c=c+1) /*Para-hacer desde el contador en cero,
    mientras el contador sea menor al valor de M e incrementando el
    contador de uno en uno.*/
    {
```

```
                    if(legajo[c]==0) /*Si el número de legajo es igual de cero
                                     entonces.*/
                    {
                        bajas=bajas+1; //Contamos
                    } //Fin del if()
                } //Fin del for()
                printf("\nCantidad de alumnos dados de baja:\t%i", bajas);
                //Mostramos el mensaje correspondiente.
                break; /*Rompemos el caso o damos por finalizado el procedimiento.*/

            case 7: //Realizamos el procedimiento para el Salir del programa.
                printf("Salir del programa…"); /*Mostramos el mensaje
                correspondiente.*/
                break; /*Rompemos el caso o damos por finalizado el procedimiento.*/
            default: /*Esta opción de la estructura condicional múltiple nos
                permite dar un mensaje de error si seleccionamos una opción del
                menú incorrecta.*/
                printf("Opcion incorrecta…"); /*Mostramos el mensaje
                correspondiente.*/
        } //Fin del switch()

        printf("\n"); //Realizamos un salto de línea.
        system("pause"); /*Realizamos una pausa mostrando el siguiente
        mensaje que "Pulsar una tecla para continuar". */
    } //Fin del while()
} //Fin del main()
```

Como vimos el procedimiento del uso de un menú de opciones, facilita la realización de un **ABM**, dado que se pueden realizar todos los procedimientos en un mismo programa.

EJERCITACIÓN SOBRE ALGORITMOS

GUIA DE EJERCICIOS Nº 1: Estructuras Secuenciales

1. Se debe calcular la superficie de un rectángulo, debiéndose ingresar los valores de los lados y calcular el resultado.
2. Ingresar 3 números y mostrarlos, uno al lado del otro y uno debajo del otro. Se deben utilizar las secuencias de escape.
3. Escribir un algoritmo que realice las siguientes acciones:
   a. Ingrese valores para 2 variables NUMERO1 y NUMERO2.
   b. Efectúe el producto de dichas variables.
   c. Muestre el resultado por pantalla.
   d. Obtenga el cuadrado del NUMERO1 y mostrarlo por pantalla.
4. Diseñar un algoritmo que calcule la longitud de la circunferencia y el área del círculo de radio dado.
5. De un alumno del curso de ingreso a la UTN se ingresan las notas obtenidas en los exámenes de las 3 materias que lo forman. Calcular el promedio del alumno y mostrarlo por pantalla
6. Calcular el número de pulsaciones que una persona debe tener por cada 10 segundos de ejercicio, si la fórmula es: número de pulsaciones = (220 - edad) /10. Para realizar el algoritmo se deberá informar la edad de la persona y luego de aplicar la fórmula correspondiente, mostrar el resultado.
7. En un hospital existen tres áreas: Cardiología, Pediatría y Traumatología. El presupuesto anual del hospital se reparte conforme a la siguiente tabla:

Área	Porcentaje del presupuesto
a. Cardiología	40%
b. Traumatología	25%
c. Pediatría	35%

Obtener la cantidad de dinero que recibirá cada área, para cualquier monto presupuestal asignado.
8. Diseñar el algoritmo necesario para que, habiéndose leído el valor de 2 variables NUM1 y NUM2 se intercambien los valores de estas, es decir que el valor que tenía NUM1 ahora lo contenga NUM2 y viceversa.
9. Se debe ingresar: nombre, sexo, edad y porcentaje de tiempo transcurrido de este año «(mes actual / total meses) * 100». Mostrar a todos los datos antes de salir del programa.

GUIA DE EJERCICIOS Nº 2: Estructuras Condicionales

10. Se debe ingresar un número entero, debiéndose indicar si dicho número es par o impar.
11. Se leen tres números, A, B, y C. Se pide escribir el mayor de ellos.
12. Se deben ingresar 3 números enteros, sumarlos y controlar si el resultado es mayor a 100. Mostrar un mensaje por si es igual, mayor o menor.
13. Ingresar la marca y el kilometraje de un vehículo, mostrar el mensaje “Realizar el control técnico”, si el kilometraje es mayor 10000.

14. Un operario de una empresa puede trabajar en dos turnos. El diurno, cuyo código es menor que 10, y el nocturno de código mayor o igual a 10. Indicar mediante un mensaje en que turno trabaja.
15. Se desea calcular el jornal para un operario sabiendo que: para el turno nocturno es de $ 500.- la hora y para el turno diurno es de $ 300.- la hora, pero en este último caso, si el día es domingo, se paga un adicional de $ 100.- la hora.
16. Hacer un algoritmo que calcule el total a pagar por la compra de camisas. Si se compran tres camisas o más, se aplica un descuento del 20% sobre el total de la compra y si son menos de tres camisas un descuento del 10%.
17. Se leen tres datos que representan: el nombre, sueldo básico y antigüedad de un empleado. Se solicita imprimir el nombre y el sueldo a cobrar. Este sueldo por cobrar se calcula adicionando al básico el 35% del mismo, si la antigüedad supera los 10 años.
18. Ingresar 2 números y realizar un menú de opciones para que los sume, reste o los muestre.
19. Se debe hacer un programa que nos permita mediante un menú de opciones ingresar dos números y realizar a cada una de las cuatro operaciones aritméticas básicas, para cada acción debe existir una opción del menú. En todos los casos mostrar el resultado correspondiente.
20. En una fábrica de computadoras se planea ofrecer a los clientes un descuento que dependerá del número de computadoras que compre. Si las computadoras son menos de cinco se les dará un 10% de descuento sobre el total de la compra; si el número de computadoras es mayor o igual a cinco, pero menos de diez, se le otorga un 20% de descuento; y si son 10 o más se les da un 30% de descuento. El precio de cada computadora es de $50000.
21. Se debe realizar un menú de opciones para: Ingresar dos cadenas, mostrarlas con funciones diferentes, por tamaño, indicar si son iguales o diferentes, y salir del programa.
22. Se leen dos cadenas de menos de 30 letras, se debe indicar si fueron ingresadas en orden alfabético.

GUIA DE EJERCICIOS Nº 3: Estructuras Repetitivas

23. Leer 5 números, sumarlos y mostrar el resultado.
24. Pedir al operador que ingrese 5 números y mostrar por pantalla el mayor.
25. Ingresar el stock de n cantidad de artículos, debiéndose averiguar cuál es el stock mayor y cual el menor. Mostrar el stock correspondiente.
26. Pedir al operador que ingrese 10 números enteros positivos y mostrar por pantalla un resultado de la suma y el promedio, uno al lado del otro.
27. Pedir al operador que ingrese un número entero positivo mayor a 1000. Determinar si es par o impar (no usar el operador módulo) y mostrar por pantalla el mensaje “Número Par” o “Número Impar”.
28. Pedir al operador que ingrese 2 números enteros positivos distintos entre sí. Determinar cuál es el mayor y el menor. Dividir al mayor por el menor, sin usar división. Mostrar por pantalla el dividendo, el divisor, el cociente y el resto.

29. Pedir al operador que ingrese 20 números enteros mayores a 999 y menores a 10000, ir mostrando por pantalla el subtotal de la suma por cada número y al final del programa, el total, el promedio, la cantidad de números pares y la cantidad de números impares. Cada resultado uno debajo del otro.

30. Ingresar el apellido y el sueldo de 5 empleados. Sacar el promedio de los sueldos, mostrar el resultado y a continuación a los datos ingresados.

31. Pedir al operador que ingrese 30 nombres. Mostrar por pantalla al nombre que tenga mayor cantidad de caracteres y el número de orden en el cual ingresó, y el nombre que tenga menor cantidad de caracteres y el número en el cual ingresó. Al finalizar hacer un listado con el número de orden, los nombres y la cantidad de caracteres de cada uno de ellos.

32. Se quiere calcular la cantidad de caracteres que va a ocupar un archivo de texto que contiene la descripción del stock de materiales de una empresa, no se sabe la cantidad de existente. Para ello se debe contar la cantidad de caracteres de la descripción de cada artículo y mostrarse al final del programa “El archivo de texto ocupa N cantidad de caracteres”, donde en N se debe mostrar la suma total de los caracteres.

33. Hacer un programa para ingresar el apellido, el nombre, carrera de ingeniería que cursa y la edad de cada uno de los 500 alumnos que cursan primer año en la facultad. Mostrar todos los datos. Se deben validar todas las cadenas de caracteres, que no pueden tener ninguna de ellas, más de 30 caracteres y no pueden estar vacías, y para la edad, controlar que sean mayores de edad.

34. Pedir al operador que ingrese N cantidad de números enteros mayores a 100 y que sean pares hasta que la suma total de los mismos sea mayor o igual a 25000 Mostrar por pantalla la suma, la cantidad de números ingresados, el menor número ingresado con su número de orden y el mayor número ingresado, también con su número de orden.

35. Pedir al operador que ingrese N cantidad números reales distintos de 0, el programa debe terminar con la palabra Final o bien cuando la cantidad de los números ingresados sea igual a 20. Mostrar por pantalla al final del programa a la lista de los números que fueron ingresados, cuantos fueron, la suma y el promedio.

36. Ingresar una serie de números enteros debiéndose averiguar la cantidad de enteros positivos y enteros negativos. Realizar la sumatoria, y si el resultado es positivo, mostrar la cantidad de números positivos y si es negativo, mostrar el resultado de la suma y a la cantidad de números negativos.

37. Ingresar 20 números, sumarlos, mostrar al final del programa los números ingresados en forma de lista y al resultado de la suma.

38. Ingresar 15 números, averiguar al mayor y al menor, mostrar a todos los datos ingresados en forma de lista y al número mayor y al menor.

39. Hacer un programa para ingresar tu nombre y apellido y mostrarlos por separado y juntos, debiéndose mostrar la cantidad de caracteres tiene el nombre, el apellido y el nombre y apellido todo junto, el nombre del apellido debe estar separado por un espacio.

40. Hacer un programa para ingresar el nombre, la carrera que cursan y la edad de N cantidad de estudiantes. Se deben validar todos los datos ingresados y mostrarlos en forma de lista en columnas (Nombre Carrera Edad).

41. Del registro de partes meteorológico, de un mes de 30 días, se registra la fecha, temperatura máxima y temperatura mínima. Diseñar un algoritmo con el uso de matrices que permita informar:
   a. el día más frío y cuál fue su temperatura.
   b. el día más cálido y cuál fue su temperatura.

42. Se debe hacer un programa que nos permita cargar las notas de las 5 materias de los 5000 estudiantes que cursan en la facultad, debiéndose cargar su legajo (debe tener 3 dígitos), las notas de las materias sin decimales y calcular el promedio por estudiante con 2 decimales. Se debe hacer un listado que nos muestre lo siguiente: LEGAJO – M1 – M2 – M3 – M4 – M5 – PROMEDIO. Para la resolución de este ejercicio se deben utilizar matrices.

43. Por teclado se ingresa el valor hora de N cantidad de empleados (la cantidad se ingresa por teclado). Posteriormente se ingresa el nombre del empleado, la antigüedad y la cantidad de horas trabajadas en el mes. Se pide calcular el importe a cobrar teniendo en cuenta que, al total que resulta de multiplicar el valor hora por la cantidad de horas trabajadas, hay que sumarle la cantidad de años trabajados multiplicados por $300 y al total de todas esas operaciones, restarle el 13% en concepto de descuentos. Imprimir el recibo correspondiente con el nombre, la antigüedad, el valor hora, el total a cobrar en bruto, el total de descuentos y el valor neto a cobrar.

GUIA DE EJERCICIOS Nº 4: Ejercicios Combinados

44. Se debe hacer un programa que mediante un menú de opciones nos permita realizar las siguientes acciones:
   1. Ingresar dos números y mostrar la suma de los números ingresados.
   2. Calcular el cuadrado de la suma del punto “a” del menú.
   3. Calcular el promedio de los números ingresados.
   4. Mostrar a todos los resultados.
   5. Salir del programa.

45. Un profesor de matemática de un establecimiento educativo registra de un total de 30 alumnos, su Nº de legajo, nombre y promedio de notas. Según el promedio, mediante un menú de opciones, desea conocer el Nº de legajo y nombre de los alumnos que:
   a. Lograron la aprobación directa (promedio mayor o igual a 6).
   b. Deben dar el examen final (promedio menor a 6 y mayor o igual a 4)
   c. Deben recursar la materia (promedio menor a 4).
   d. Indicar cuantos alumnos obtuvieron la aprobación directa, dan final o recursan.
   e. Salir del programa.

46. Crear un programa con un menú de opciones para realizar las siguientes acciones: En la opción 1 se deben cargar los legajos, los nombres y las horas trabajadas en un mes, por cada uno de ellos, de una N cantidad de empleados de una empresa. La opción 2 calcular el sueldo de cada uno multiplicando la cantidad horas por el valor hora (el valor hora es a elección). La opción 3 mostrará el listado de todos los datos ingresados, debiéndose indicar cuantos empleados son en total. La opción 4 al salir del programa, mostrará el total y el promedio de los sueldos pagados.

47. Hacer un programa que mediante un menú de opciones nos permita realizar las siguientes acciones:

A. Ingresar los nombres y las notas de 50 alumnos, debiéndose controlar que la nota sea positiva y menor o igual a 10.
B. Averiguar la nota mayor y la menor.
C. Hacer dos listados, el primero con todos los alumnos que tengan la mayor nota, y el segundo con todos los alumnos que tengan la menor nota.
D. Salir del programa.

48. Hacer un programa que por medio de un menú de opciones nos permita realizar las siguientes acciones:

a. Ingresar el nombre y el precio unitario de 100 artículos. El precio unitario debe ser mayor que 0 (cero).
b. Calcular la ganancia del 25 % para cada artículo y mostrar todos los datos de cada uno, después de realizado el cálculo.
c. Indicar cuantos artículos tienen la menor ganancia y mostrar cuales son.
d. Hacer un listado de los artículos que superen el promedio de la ganancia. Debe aparecer el orden de ingreso.
e. Salir del programa.

49. Hacer un programa que por medio de un menú de opciones nos permita realizar las siguientes acciones:

1. Ingresar los siguientes datos de un stock de materiales de una cantidad indeterminada de artículos: Código (Según el índice), Descripción (30 caracteres), Cantidad y Precio de Compra (que no pueden ser negativos).
2. Calcular el Precio de Venta de cada artículo que se obtiene calculando un 30% de incremento del Precio de Compra.
3. Borrar los datos de un artículo determinado, buscarlo por su nombre (Descripción) y mostrar el resultado.
4. Hacer los listados de los artículos existentes y de los que fueron eliminados, por separado, mostrar a todos los datos.
5. Salir del programa.

50. Se debe hacer un programa que mediante un menú de opciones, nos permita realizar las siguientes acciones:

a) Ingresar la clave y el nombre y apellido de los cinco mejores alumnos.
b) Mostrarlos en el orden inverso del que fueron ingresados.
c) Fin del programa.

NOTA: Las opciones del menú deben ser en minúsculas. La clave y el nombre y apellido deben ser validados: La clave debe tener 4 dígitos alfanuméricos y el nombre y apellido, no más 30 caracteres.

51. Se debe realizar un programa que administre los 1000 empleados de una fábrica y que mediante un menú de opciones nos permita realizar las siguientes acciones:

A. Ingresar los siguientes datos (durante el ingreso de los datos debe aparecer el número de orden):
   .1. Legajo debe tener 4 dígitos.
   .2. Nombre no debe superar los 30 caracteres.
   .3. Salario debe ser mayor a cero y debe ser con decimales.
   .4. Sectores 'A', 'B' y 'C'. Todos en mayúsculas.
B. Calcular el salario máximo y mínimo.
C. Contar cuántos empleados trabajan en cada sector.
D. Mostrar el salario máximo y la lista de empleados que perciben ese salario.
E. Mostrar el salario mínimo y la lista de empleados que perciben ese salario.
F. Hacer el listado de los empleados por sector, debiéndose mostrar todos los datos incluyendo el número de orden.
G. Salir del programa.

52. Se debe hacer un programa que mediante un menú de opciones nos permita realizar las siguientes acciones:

A. Ingresar el código (debe ser de 8 caracteres), la descripción, el stock inicial y el stock vendido de un total de 100 artículos.
B. Calcular el stock actual (sacar la diferencia entre el stock inicial y el vendido), hacer un listado de cada uno de los stocks.
C. Calcular el total y el promedio de cada stock, y al promedio general entre los stocks. Mostrarlos.
D. Hacer un submenú para trabajar con los datos del stock actual:
   1. Buscar un artículo.
   2. Borrar a un artículo.
   3. Modificar los datos de un artículo.
   4. Listar a los artículos que tengan stock.

   Para encontrar los datos de un artículo siempre lo hacemos por su código.
E. Salir del programa.

53. Se debe hacer un programa que nos permita averiguar el valor de X (que va a ir del 1 al 30 inclusive y de 3 en 3), con las funciones trigonométricas que nos permitan calcular el coseno, el seno y la tangente. Para ello se debe realizar un menú de opciones que nos permita mostrar en columnas:

(a) El valor de X y el coseno.
(b) El valor de X y el seno.
(c) El valor de X y la tangente.
(d) El valor de X y el resultado de la suma del seno más el coseno y más la tangente.
(e) Salir del programa.

CUESTIONARIO GENERAL TEÓRICO

1. ¿Cómo podríamos definir a los lenguajes de programación y de qué manera los podríamos clasificar?
2. ¿El código fuente de un programa a que le hace referencia?
3. ¿Cuál es el lenguaje que se utiliza internamente el hardware y cuáles son los símbolos que lo componen?
4. ¿Teniendo en cuenta la clasificación de los lenguajes de programación en qué nivel de abstracción podemos encontrar al lenguaje C y por qué?
5. ¿El lenguaje C se ejecuta de forma compilada o a modo interprete?
6. ¿Para qué se utilizan las llaves, los corchetes, los paréntesis, las comillas dobles y las comillas simples dentro de un programa?
7. ¿Para qué sirven las secuencias de escape, con cuales podemos cambiar al cursor de lugar y con qué función o funciones se utilizan?
8. ¿Para qué se utilizan los modificadores de formato y cuales les corresponden a los diferentes sistemas de numeración[18]?
9. ¿Qué es un programa y de que elementos están compuestos?
10. ¿Qué son las constantes y como se declaran dentro de un programa?
11. ¿Qué son las bibliotecas, que funciones cumplen, de qué tipo existen y de qué manera se declaran dentro de un programa?
12. ¿Qué bibliotecas utilizamos en el lenguaje C y que funciones que están incluidas en cada una de ellas?
13. ¿Qué son los argumentos de una función?
14. ¿Qué son las variables, de qué tipo existen, que diferencia existen en declarar una variable con inicializarla y como realizaríamos ambas acciones con los distintos tipos de datos dentro de un programa?
15. ¿Qué son los algoritmos y cuáles son las características que deben cumplir?
16. ¿De qué distintas maneras se puede resolver un algoritmo?
17. ¿A qué se llama pseudocódigo y para que se utilizan los diagramas de flujo?
18. ¿Qué son los arreglos, de qué tipo existen y que tipos de datos pueden contener?
19. ¿Qué tipos de comentarios podemos incorporar dentro de un programa y para que se utilizan?
20. ¿Qué tipos de datos podemos utilizar para realizar los programas y que modificadores de formato están asociados a cada uno de ellos?
21. ¿Qué son las estructuras de control en programación, de qué tipo existen y que funciones cumplen cada una?
22. ¿A qué se llama IDE (Entorno de Desarrollo Integrado), de qué grupo de programas está compuesto y que funciones cumplen cada uno de ellos?
23. ¿Qué tipo de análisis realizan los compiladores y que tipos de compiladores existen?
24. ¿Cuáles son los pasos para la creación de un programa?
25. ¿Cómo se ejecutan los programas en modo intérprete y de qué tipo existen?
26. ¿Qué son los operandos y que son los operadores?
27. Indicar que tipos de operadores existen, de que símbolos están compuestos y funciones cumplen cada uno de ellos.
28. ¿A qué se llama procedimiento, cuáles son los más importantes que se pueden realizar dentro de un programa y que acciones realizan cada uno de ellos?
29. ¿Qué sistemas operativos son creados con el Lenguaje C?
30. ¿Quién fue el creador del Lenguaje de C de programación?

[18] Los sistemas de numeración son: BINARIO, DECIMAL, HEXADECIMAL y OCTAL.