Apuntes de Asignatura: Programación (ELC-1022)

Impacta a Ingeniería Eléctrica (IELE-2010-209)

Elaboró: Saúl Almazán Cuéllar

Septiembre 2024

Contribución académica

La asignatura de programación aporta al perfil del egresado a la carrera de ingeniería eléctrica las competencias que le permiten comprender y aplicar los conocimientos teóricos en la modelación, análisis y síntesis de sistemas, adquiridos en las diversas materias cursadas durante su formación como ingeniero eléctrico para su aplicación en las áreas industrial, comercial y de servicios asociados con señales físicas, y sirviendo como antecedente principalmente a los conocimientos de las asignaturas de Electrónica Digital, Control I, Control II, Control Lógico Programable I y Control Lógico Programable II, entre otras.

La asignatura de programación proporciona las bases que permitirán a los alumnos que deseen profundizar sus conocimientos en una materia en particular o iniciar su propia empresa desarrollando proyectos de automatización de acuerdo con su experiencia.

Esta materia es la base para que en los siguientes semestres los estudiantes deben tener estos conocimientos para poder desarrollar los programas de las materias arriba mencionadas. Esto es la relevancia que tiene esta materia y a su vez la complejidad que tiene la enseñanza de los conocimientos elementales y transversales que se deben cubrir en el programa.

La práctica que se realiza en los laboratorios amplía el aprendizaje haciéndolo significativo y trascendental para avanzar de manera positiva en la retícula de la carrera.

Contenido

Tema 1 Fundamentos de programación

1.1. Importancia de la programación de computadoras.
1.2. Clasificación de los tipos de lenguajes de programación.
1.3. Diseño de algoritmos.
1.4. Diagramas de flujo
1.4.1. Uso de programas de simulación para diagramas de flujo.
1.5. Importancia del compilador en los lenguajes de alto nivel.

Tema 2 Elementos del lenguaje de programación

2.1. Introducción al entorno de programación.
2.2. Estructura básica de un programa
    2.2.1. Comentarios.
    2.2.2. Identificadores.
    2.2.3. Palabras reservadas.
    2.2.4. Tipos de datos:
        2.2.4.1. Simples.
        2.2.4.2. Compuestos.
    2.2.5. Variables y Constantes.
    2.2.6. Atributos.
    2.2.7. Operadores
        2.2.7.1. Aritméticos
        2.2.7.2. Lógicos
        2.2.7.3. Condicionales
        2.2.7.4. De desplazamiento
    2.2.8. Manejo de cadena de caracteres

Tema 3 Arreglos

3.1. Definición e importancia de los arreglos en la programación.
3.2. Declaración de arreglos unidimensionales y multidimensionales.
3.3. Lectura de arreglos unidimensionales y multidimensionales
3.4. Operaciones con arreglos.

Tema 4 Estructuras de control

4.1. Estructuras de selección.
    IF
    IF/ELSE
    IF/ELSEIF

4.2. Estructuras de repetición.
    WHILE
    DO WHILE
    FOR
4.3. Estructura de múltiple selección.
    SWITCH/CASE
4.4. Formulación y aplicación de algoritmos utilizando estructuras de control.

Tema 5 Funciones

5.1. Estructura de la función.
5.2. Llamado o invocación de una función.
5.3. Uso de funciones con parámetros
    5.3.1. De entrada
    5.3.2. De salida
5.4. Funciones externas
    5.4.1. Del usuario
    5.4.2. De bibliotecas

Tema 6 Uso de puertos de comunicación

6.1. Tipos de puertos comunicación.
6.2. Especificaciones de los puertos de comunicación.
6.3. Envió y recepción de datos a través de un puerto de comunicación.

Introducción

Con estos apuntes para la asignatura de Programación, se pretende que los estudiantes (principalmente) y docentes, tengan material de utilidad no solo durante el desarrollo de la asignatura, sino también les sirva de referencia para asignaturas posteriores.

En el caso de los estudiantes, les pueda ser de utilidad en su vida profesional. Esto se obtendrá por medio de ejemplos desarrollados paso a paso, empleo de evaluaciones numéricas, sugerencias, entre otras posibles herramientas dependiendo del tema abordado.

Uno de los objetivos de estos apuntes es que el estudiante tenga una mejor preparación y los docentes tengan tiempo para preparar mejor sus exposiciones frente a grupo.

1. Fundamentos de programación

1.1. Importancia de la programación de computadoras.

De acuerdo con [1], “La importancia de la programación radica en que los seres humanos necesitamos acelerar y automatizar los trabajos, procesos y tareas que realizamos, así como optimizarlos y además no solo realizar trabajos sino a su vez llevar un registro de las actividades que se requieren realizar”.

Como complemento de lo expresado por [1], [2] hace notar que: “Por medio de la programación es posible la creación de herramientas que posibiliten el almacenamiento de datos facilitando su recuperación, procesar grandes cantidades de información, así como organizarla y obtener nuevos datos a partir de ella, establecer comunicaciones remotas y hasta simular inteligencia”. En otras palabras, [2] continúa “La programación es la base de la transformación digital, permitiendo a las organizaciones adoptar nuevas tecnologías como la inteligencia artificial, aprendizaje automático, internet de las cosas, etc.”.

1.2. Clasificación de los tipos de lenguajes de programación.

Existen diferentes clasificaciones para los lenguajes de programación, y depende del interés de lo que un determinado autor quiera mostrar será la clasificación que propondrá.

En nuestro caso, el interés radica principalmente en enfocar la atención en un lenguaje de programación que brinde la posibilidad de manipular bits, bytes y direcciones de un procesador que son los elementos básicos con los que una computadora funciona (esto es que soporte el lenguaje ensamblador por lo que tiene características de lenguaje de bajo nivel). Requerimos un lenguaje que soporte tipos de datos, esto es la definición de un conjunto de valores que una variable (por ejemplo enteros, flotantes y caracteres) y que tenga capacidad de almacenar un conjunto de operaciones que puedan ser ejecutadas en dicha variable. Los lenguajes de alto nivel son fuertemente tipeados y además realizan verificaciones para evitar errores en tiempo de ejecución.

Un **lenguaje de nivel medio** comparte características de los **lenguajes de alto nivel** y de los **lenguajes de bajo nivel**.

La Tabla 1.1 muestra la clasificación de los lenguajes de programación que propone [3, p. 5]:

Tabla 1.1 Clasificación de los Lenguajes de Programación

Alto nivel	Ada
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
Nivel medio	Java
	C++
	C
	FORTH
	Macroensamblador
Bajo nivel	Ensamblador

1.3. Diseño de algoritmos.

El diseño de algoritmos consta de una serie de sencillos pasos que deben ser cumplidos puntualmente. A continuación dichos pasos.

1) Definir el problema. Es necesario tener bien definida la idea de lo que se desea lograr.
2) La solución de un problema probablemente no sea la única, entonces dentro del universo de posibles soluciones se selecciona la más cercana y sencilla al problema al que nos enfrentamos.
3) Partiendo de la solución seleccionada de paso 2, diseñar una secuencia de pasos ordenados en los cuales aparecerá la necesidad del empleo de estructuras de selección (switch), ciclos de iteraciones fijas (for) y ciclos de iteraciones condicionadas (while y do while).

Existen aplicaciones que son de gran ayuda en el diseño de algoritmos, ya que son muy genéricas, esto es, no dependen de un lenguaje de programación en particular (emplean descripciones cotidianas por ejemplo Escribir, Leer, Para, Hasta, Hacer, etc. A esta descripción se le conoce como pseudocódigo). Son de muy fácil operación, permitiendo que el usuario realice pruebas de su algoritmo propuesto tomando en cuenta la serie de 3 pasos listados con anterioridad en esta sección.

Una aplicación en particular es Pseint, la cual es gratuita y puede ser descargada de http://pseint.sourceforgae.net.

A partir del algoritmo propuesto por el usuario, Pseint nos muestra un diagrama de flujo y analizar su comportamiento, permitiendo las modificaciones pertinentes al algoritmo en caso de ser necesario.
En la figura 1.1 se muestra la pantalla de Pseint para el algoritmo del cálculo del factorial de n (n!), mientras que la fig. 1.2 muestra su corrida de prueba.

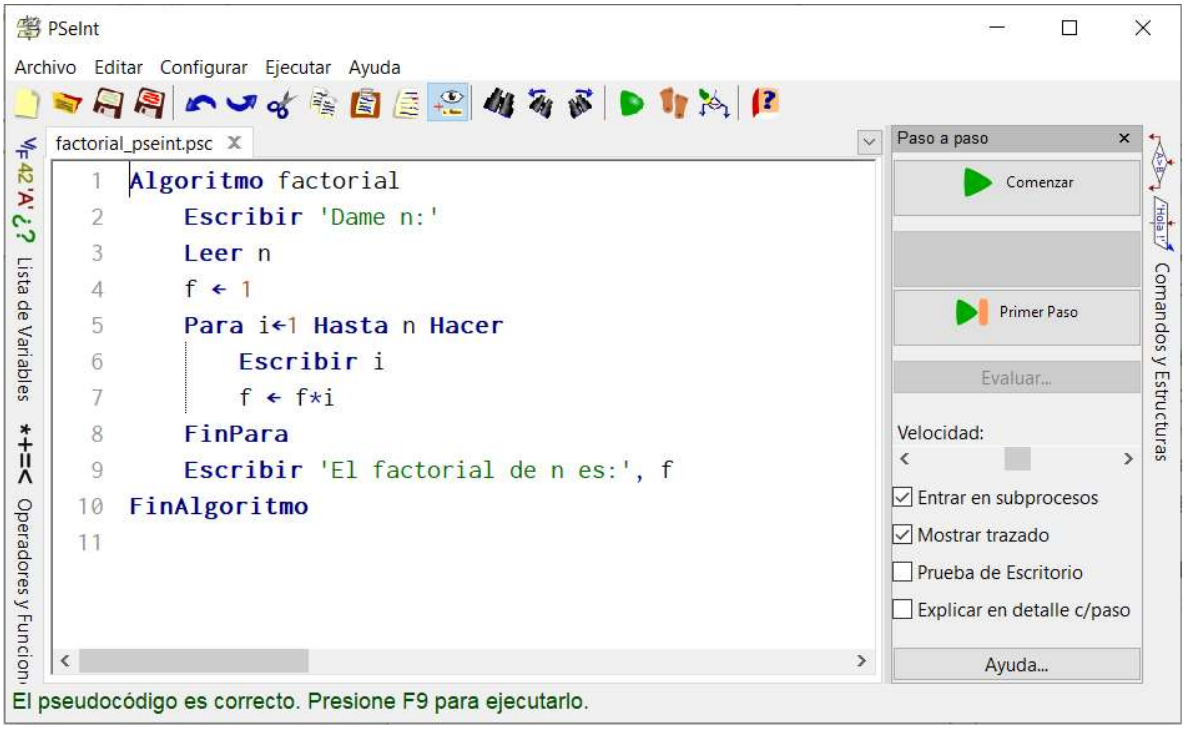


Fig. 1.1 Pantalla de Pseint para el algoritmo del cálculo del factorial de n (n!).

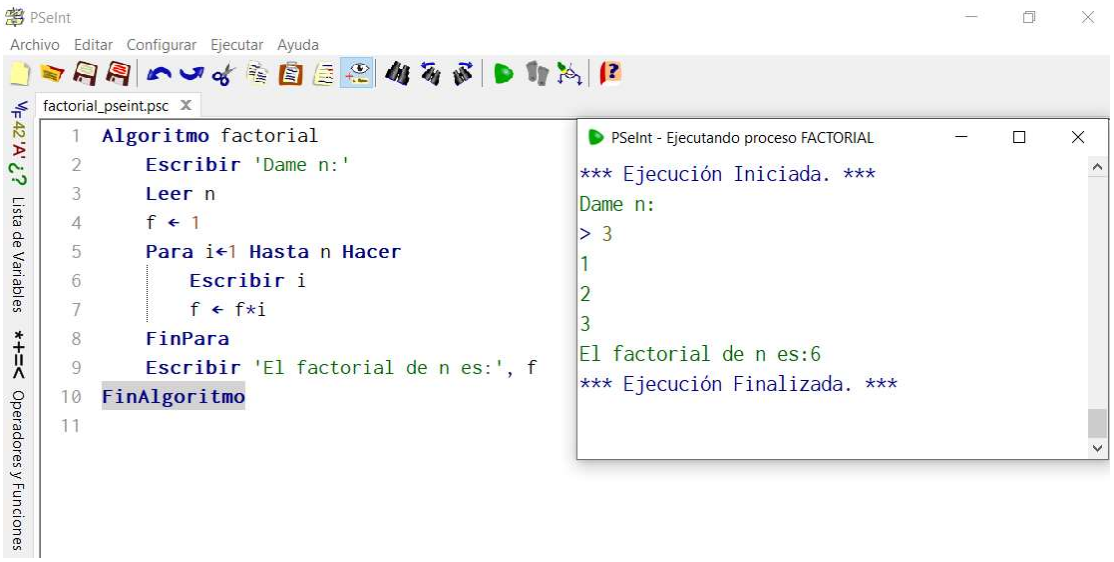


Fig. 1.2 Pantalla de Pseint para n! en corrida de prueba.

En la figura 1.1 observamos que en las orillas izquierda y derecha del marco de la pantalla principal (IDE), en los cuales por el lado izquierdo identificamos dos opciones de selección: “Lista de Variables” y “Operadores y Funciones”, mientras que por el lado derecho tenemos solo una opción que es “Comandos y Estructuras”, además también encontramos la sección del control de ejecución del algoritmo. En la figura 1.3 se muestra el despliegue de las 3 opciones de selección laterales y el control de ejecución del algoritmo (Paso a paso).

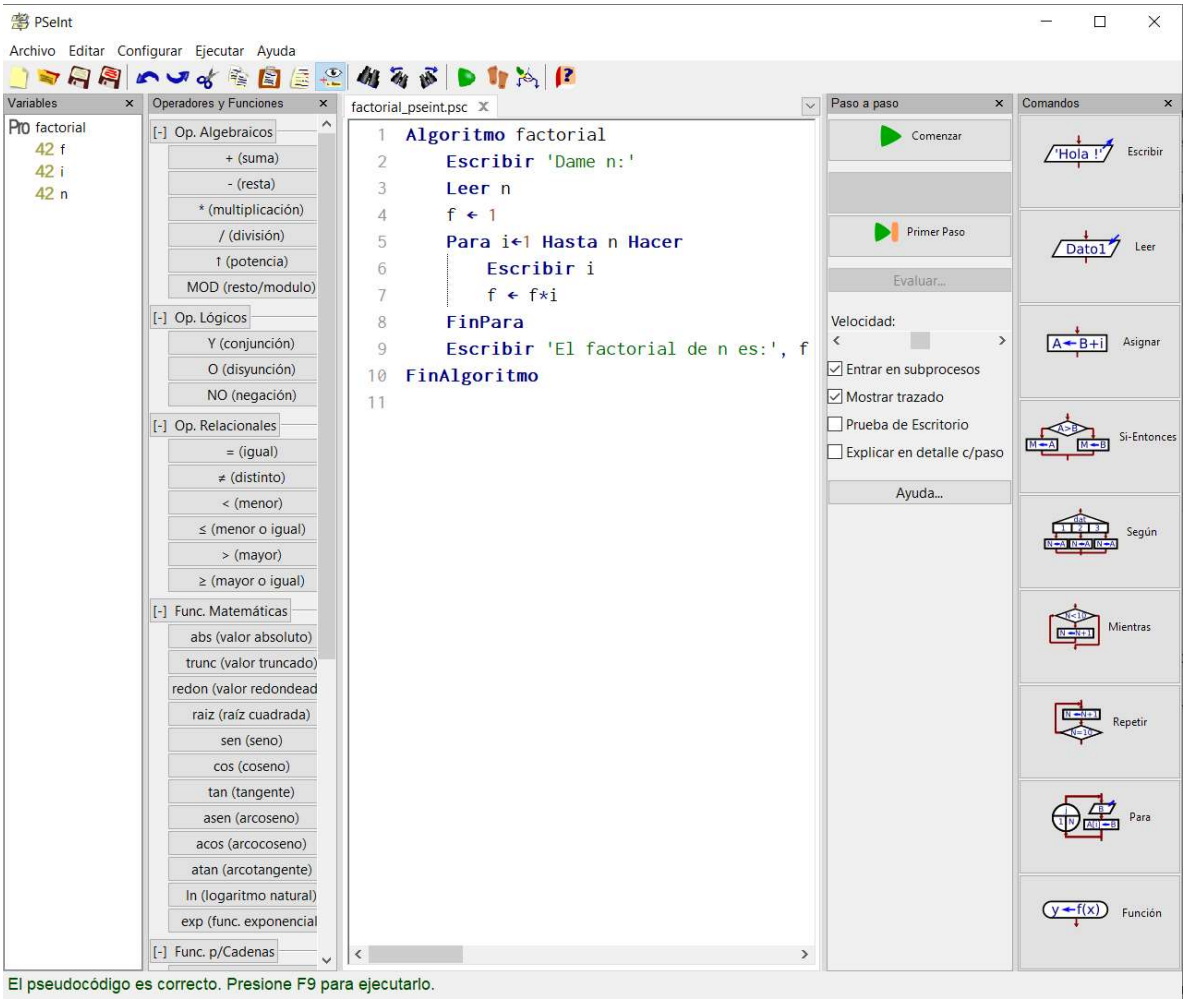


Fig. 1.3 Despliegue de las 3 opciones de selección laterales de la pantalla principal de Pseint y el control de ejecución del algoritmo (Paso a paso).

En la opción de selección a la izquierda (Variables) leemos el encabezado Pro que significa Proceso y para Pseint, Proceso es sinónimo de Algoritmo. Y lo que muestra son las variables empleadas en el algoritmo actual. En este caso se

muestran las variables f (factorial), i (iterador) y n (número del cual se desea el factorial).

Un punto importante es que los identificadores, o nombres de variables, deben constar sólo de letras, números y/o guion bajo (_), comenzando siempre con una letra.

En la columna se observa que cada variable empleada en el algoritmo tiene como antecedente un número, en este caso 42, que significa los potenciales tipos de datos que determina el intérprete en caso de que el tipo de variable pueda deducirse antes de ejecutar el algoritmo. Puede seleccionar una para resaltarla en el pseudocódigo.

En la segunda columna a la izquierda (la más interna, cuyo título es Operadores y Funciones), es un menú de selección que consta de 7 grupos. Estos grupos son: Operadores Algebraicos, Lógicos y Relacionales, Funciones Matemáticas, Operaciones con cadenas, Funciones misceláneas y Constantes comunes. En total se tienen 39 opciones de selección, lo que facilita enormemente el diseño de casi cualquier algoritmo típico en ingeniería.

1.4. Diagramas de flujo.

Un diagrama de flujo representa la esquematización gráfica de un algoritmo, mostrando los pasos o procesos a seguir para lograr la solución del problema. Estos diagramas representan ideas, soluciones, mecanismos y fenómenos para facilitar su comprensión. Los diagramas de flujo son importantes porque no dejan lugar a dudas, ya que muestran la relación de las entradas, los procesos y las salidas.

Es importante hacer notar que la simbología empleada en la realización de un diagrama de flujo es convencional, y esta es la razón por la que encontramos tablas de símbolos indicando que representan y/o significado.

1.4.1. Uso de los programas de simulación para diagramas de flujo.

En nuestro caso emplearemos PSDraw que es parte de Pseint. PSDraw es un software que convierte el algoritmo escrito empleando pseudocódigo a diagrama de flujo. Seleccionando “Editar Diagrama de Flujo” (desde el menú de Archivo) o por medio del icono “Dibujar Diagrama de Flujo...” (que es el penúltimo icono de la barra de herramientas que se encuentra en la parte superior del IDE), PSDraw inmediatamente mostrará el diagrama de flujo correspondiente al algoritmo activo en Pseint.

La figura 1.4 muestra el diagrama de flujo correspondiente al algoritmo Factorial. En esta figura observamos al lado derecho de PSDraw una opción de selección y esta es “Comandos y Estructuras”, en Pseint también aparece dicha opción en su lado derecho.

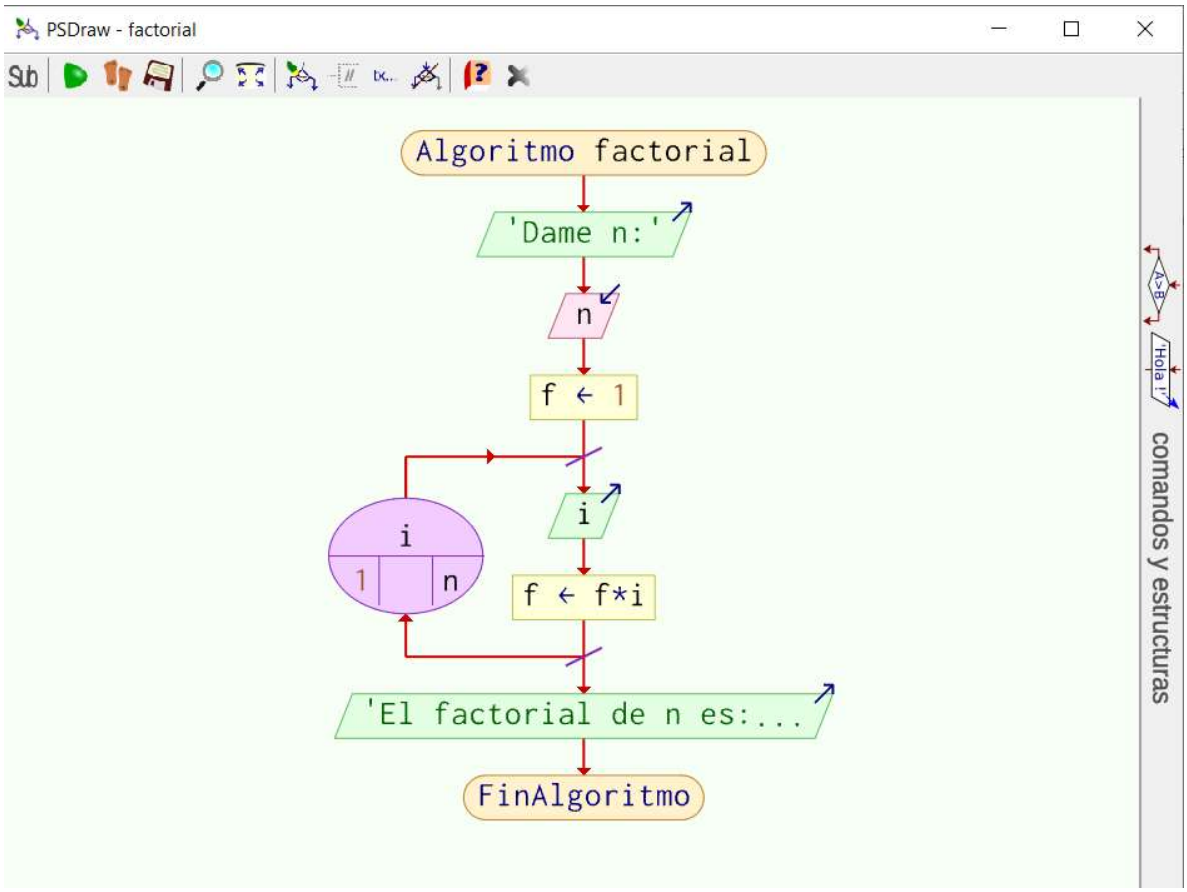


Fig. 1.4 Diagrama de flujo correspondiente al algoritmo Factorial.

En figura 1.5 se muestra el diagrama de flujo del algoritmo factorial en tiempo de ejecución. Donde una flecha de color verde va apuntando la figura equivalente al pseudocódigo en Pseint.

Debemos notar en el diagrama de flujo que:

- El principio y final del algoritmo son óvalos alargados y el texto que contienen es azul y negro en fondo rosa.
- La petición de información y en los resultados, el texto mostrado es de color verde al igual que el fondo pero de tonalidad diferente.
- En la asignación (en f← f*i o en f←1) el texto es negro sobre fondo rosa, todo en un rectángulo.

- La estructura **Para** es la equivalente a la estructura **for** de algún lenguaje de programación estructurada, por ejemplo **C**. Su color es morado con texto en negro.
- El iterador i está en un rombo, cuyo texto es de color negro y su fondo es verde pálido.

Habrá usuarios que se sientan cómodos con la combinaciones de color de texto y fondo comentadas recientemente, sin embargo es posible que muchos otros no, y seguramente quisieran modificar a su preferencia.

Existe un archivo conocido como **Perfil**, en él es posible modificar la situación (coloraciones), comportamiento de los editores o del interprete, etc. La ayuda es excelente, resolverá su problema.

Con Word es posible crear diagramas de flujo. Entrando al menú de Insertar, seleccionar la pestaña Formas, ahí se encuentran varios elementos que pueden ayudar a crear los diagramas. Sin embargo no es posible evaluar el diagrama de flujo que genere.

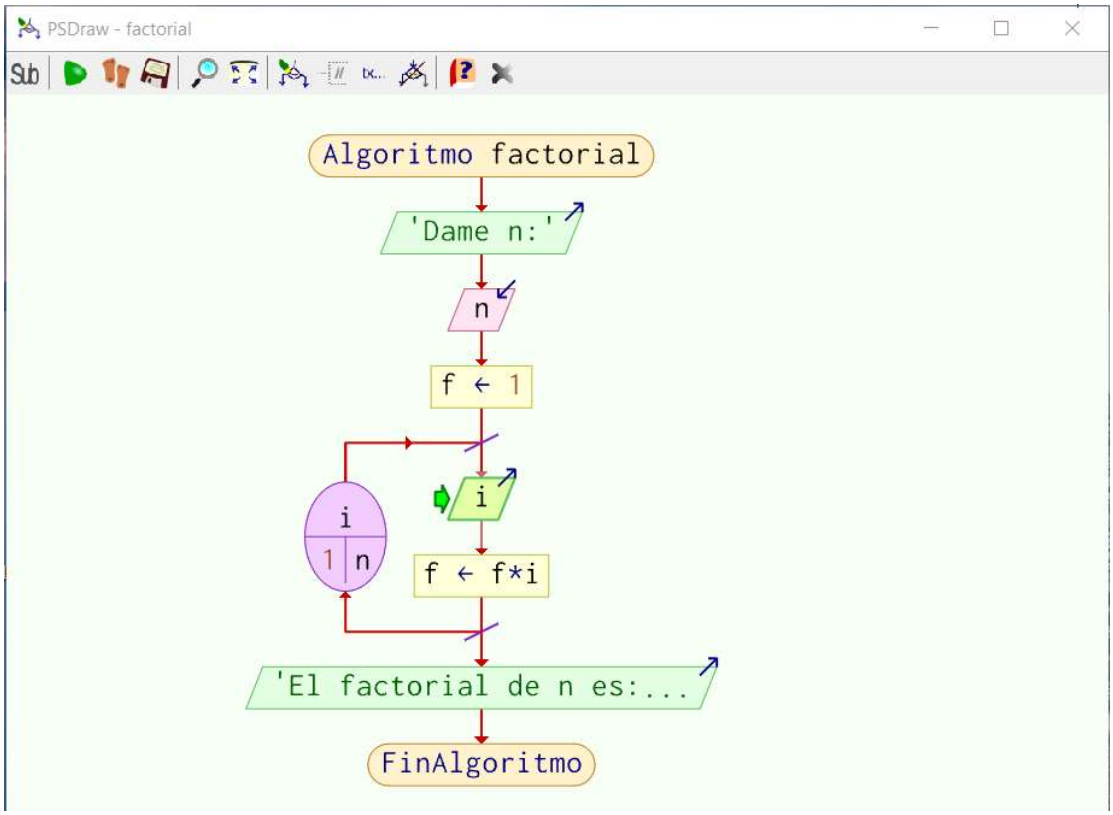


Fig. 1.5 Diagrama de flujo del algoritmo factorial en tiempo de ejecución (observe la flecha verde apuntando a la variable i que es el iterador).

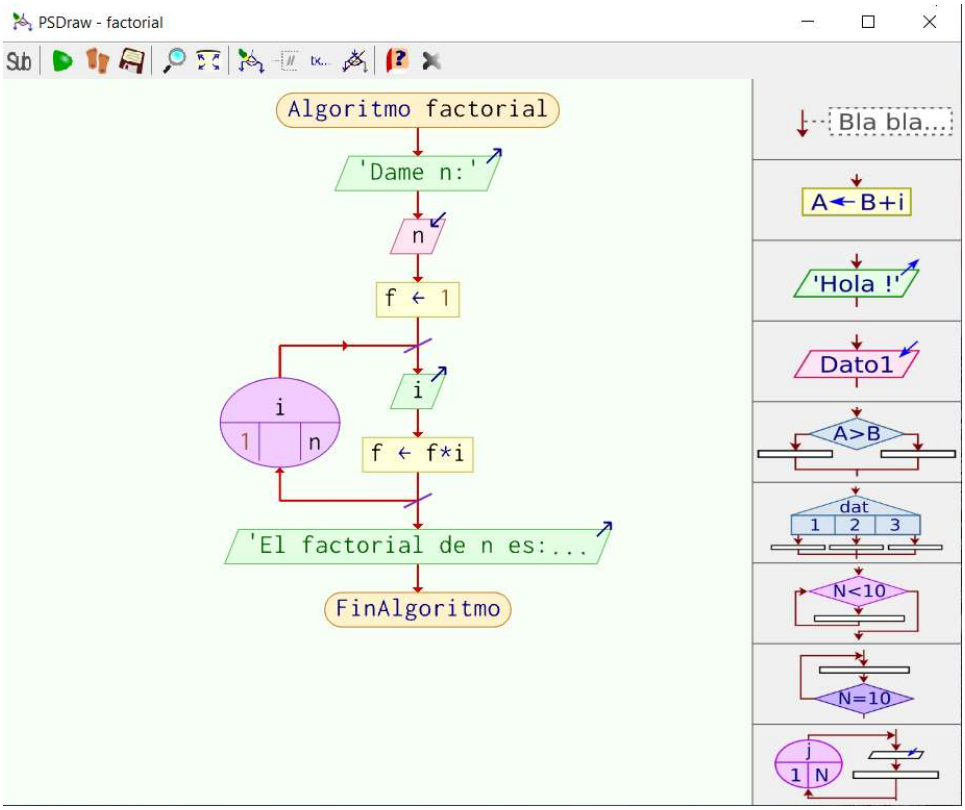


Fig. 1.6 Desplegado de la opción de selección “Comandos y Estructuras”

La opción de selección “Comandos y Estructuras” se muestra desplegada en la figura 1.6, la describiremos a continuación, indicando donde aplique su equivalente en un lenguaje de programación estructurada:

1) Comentario (texto libre que el intérprete ignora) ≡ /* */ o //
2) Asignación/Dimensión/Definición
3) Escribir (instrucción para generar salidas)
4) Leer (instrucción para obtener entradas)
5) Si-Entonces (estructura condicional simple) ≡ if-else
6) Según (estructura de selección múltiple) ≡ switch-case
7) Mientras (estructura repetitiva) ≡ while
8) Repetir-Hasta que (estructura repetitiva) ≡ do-while
9) Para (estructura repetitiva) ≡ for

Por otro lado, en el diagrama de flujo, el sentido en el que la información se desplaza (fluye) es indicado con flechas en color rojo.

Las peticiones de información (“Dame n”) y los resultados o información que el usuario debe conocer (i la iteración actual de algún ciclo) son mostradas por medio de un rombo con una flecha azul apuntando hacia afuera. La captura de

información (n el número del cual se desea el factorial) es representada por un rombo que contienen una flecha azul en su esquina superior derecha la cual apunta hacia adentro.

En los lenguajes de programación estructurada, el código que se encuentra dentro de un ciclo (for, while, do - while) es acotado por Curly Brakets ({ }), espaciamiento, etc., con el fin de definir explícitamente un bloque de código que se ejecutará dentro de una estructura determinada.

Con los diagramas de flujo sucede algo similar. En la figura 1.6 vemos que la estructura Para "encierra" por medio de dos líneas rojas terminadas en flecha (una saliendo de Para con la información que será actualizada y otra llegando a Para con información nueva) las instrucciones mostrar la iteración y actualizar el valor del factorial que está siendo calculado, y en consecuencia tenemos bien definido el bloque de instrucciones correspondiente a Para.

Algunos preferirían algo más compacto que un diagrama de flujo. PSDraw tiene capacidad de representar un diagrama de flujo como diagrama Nassi-Shneiderman. La figura 1.7 muestra el diagrama Nassi-Shneiderman para el algoritmo factorial.

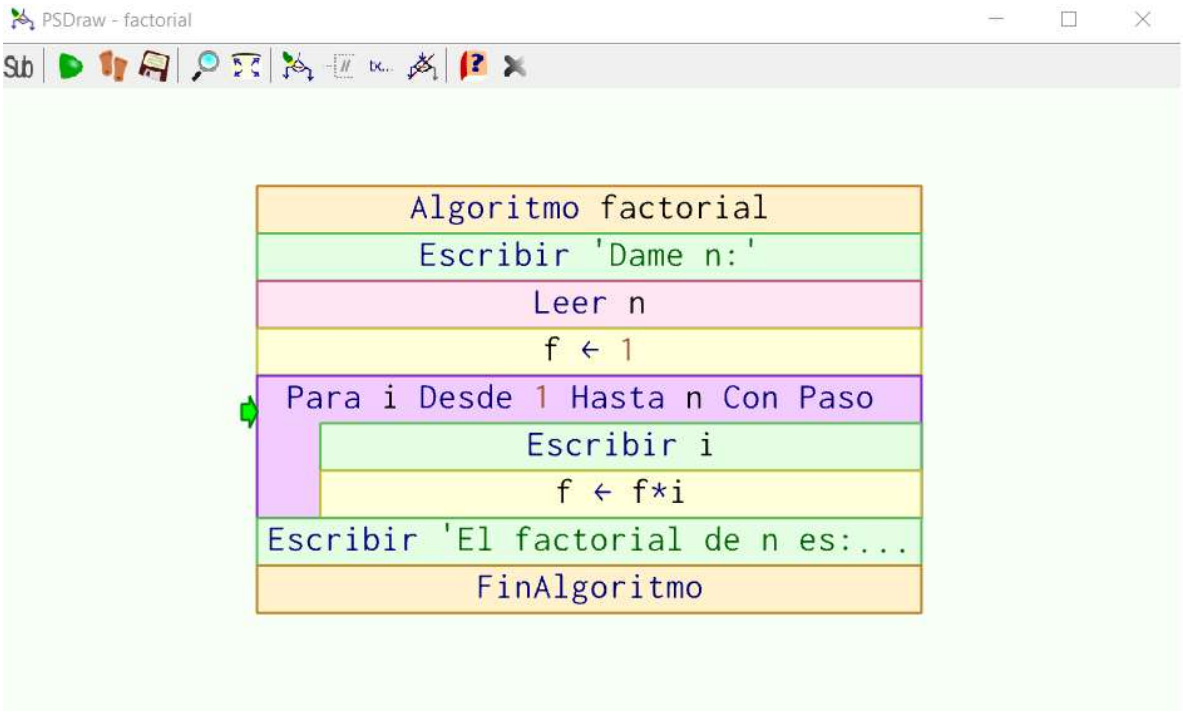


Fig. 1.7 Diagrama Nassi-Shneiderman para el algoritmo factorial en tiempo de ejecución (observe la flecha verde a la izquierda).

1.5. Importancia del compilador en los lenguajes de alto nivel.

Una vez que se haya escrito el texto de un programa es necesario compilarlo. El compilador traduce el texto del programa fuente a otro texto, que aún no es ejecutable, llamado programa objeto (es un archivo que se guarda en disco con la extensión **.obj** o algunas veces dependiendo del compilador con la extensión **.o**).

Si en el proceso de compilación se detecta algún error (error de sintaxis), irán apareciendo (normalmente) en la parte inferior de la pantalla (IDE) los mensajes correspondientes.

De aquí que la importancia del compilador radica en que revisa por errores de sintaxis los programas fuente. Si no encuentra algún error de sintaxis, entonces cada instrucción es convertida cadenas de unos y ceros, que no es otra cosa diferente para generar código de máquina (código al más bajo nivel) orientado a un procesador en particular.

Intentar hacer el trabajo del compilador de manera manual, es una tarea casi imposible, ya que en la actualidad cualquier aplicación consta de más de un archivo conteniendo código fuente.

2. Elementos del lenguaje de programación

2.1. Introducción al entorno de programación.

Vivimos en un mundo globalizado con avances tecnológico a diario, lo cual hace necesaria la actualización y capacitación continua, para estar al día con esos cambios tecnológicos, nuevas teorías, nuevas formas de hacer las cosas. Todo está cambiando rápidamente, siendo urgente la adaptación a esos cambios tan pronto como sea posible.

En la industria, se llevan registros del ¿a quién (proveedor) se le pide determinado insumo?, ¿con qué características?, ¿cuál es la cantidad requerida?, ¿cuándo se pide?, ¿cuál es su tiempo de entrega? En cuanto esté disponible dicho insumo,¿ llegó con la calidad requerida?, ¿dónde se almacenará?, etc. De todo hay registro.

Si hablamos de maquinaria, equipo, personal, etc., la historia se repite. De todo hay registro.

Es imposible, un control y registros de manera manual. Todo proceso o casi todos los procesos son automatizados, y es necesario que alguien o algo se encargue de llevar a cabo dicha tarea.

Son demasiados detalles de los que es necesario llevar registros, generar reportes con la información correcta. Que esos informes lleguen a tiempo a la persona indicada, etc. Si la programación no existiera, tratar de atender adecuadamente tanto detalle, sería una tarea abrumadora. Nos damos cuenta de que “La programación es como el oxígeno”, está en todos lados. Por medio de la programación controlamos equipos, maquinarias, conocemos la magnitud de sus parámetros importantes, determinamos cuando debemos detener la maquinaria y hacer mantenimiento, llevar registros de producción, y muchas otras tareas más. Esta es la importancia de la programación, es necesaria casi para cualquier cosa.

Code::Blocks 20.03 es un software por medio del cual se crea código ejecutable, tomando como fuente el lenguaje C, C++ o Fortran. Es buena opción, ya que es un software muy completo, en el cual se ha puesto gran empeño en su desarrollo. Cuenta con una gran cantidad de herramientas con lo que es poco probable que alguien sienta la necesidad de obtener un software comercial. Code::Blocks puede ser descargado de manera gratuita a través de: https://sourceforge.net

Una vez que se inicia Code::Blocks nos muestra la pantalla de bienvenida y nos da opción de seleccionar alguno de los últimos archivos con los que se trabajó,

proyectos, hacer en lace con los foros de Code::Blocks, reportar algún problema detectado y consejo del día.

La mayoría de las veces estaremos trabajando con archivos. De la pantalla de bienvenida podemos seleccionar alguno de ellos o ir al menú de Archivo y seleccionar Archivos recientes o Nuevo y después Archivo si lo que deseamos es iniciar un nuevo programa. Podemos compilar algún programa reciente después de modificarlo con la opción Construir o Construir y ejecutar, del menú de Construir. En caso de errores aparecerán los mensajes en la ventana inferior del IDE y el texto causante del mensaje aparecerá marcado con un cuadro rojo en el texto del programa indicando con ello la línea del error. Corrija el error y repita la operación. Una vez que el programa está libre de errores tendremos la oportunidad de evaluar los resultados del programa. No es necesario guardar el programa con el que trabajamos actualmente. Cuando Construimos y ejecutamos la operación de guardar es automática.

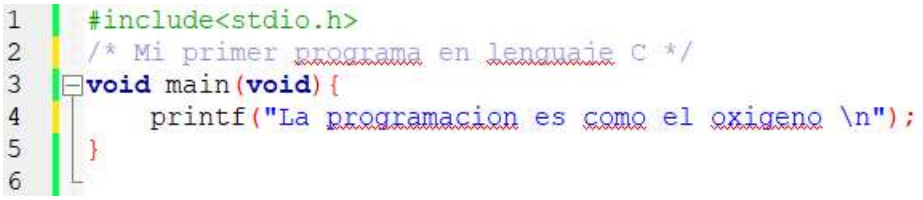


Fig. 2.1 El primer programa en lenguaje C.

La línea en texto verde es una indicación para el compilador (también se le conoce como orden del preprocesador). Sirve para incluir en un programa una “biblioteca de entrada/salida”.

El carácter “\n” que significa “comenzar una línea nueva” no es indispensable, pero contribuye a mejorar la legibilidad del texto de salida.

En la mayoría de los lenguajes que han aparecido recientemente, las entradas\salidas no forman parte del lenguaje. Sin embargo, tienen mejor acogida los lenguajes que disponen de un mínimo de posibilidades de entrada/salida, que es el caso del lenguaje C, donde se tienen algunos de estos procedimientos, presentándolos en forma de archivo con el nombre de biblioteca de entrada/salida. Esta biblioteca puede construirse:

- Como un conjunto de instrucciones o declaraciones escritas en C, para que el programador pueda incluirlas en su programa cuando las necesite.
- En forma compilada, cómo módulo que se añade al programa del usuario en el enlace.

En nuestro primer programa,

main indica que se trata del programa principal, no de una función (el C no reconoce procedimientos, solo funciones) llamada desde el programa principal,

( ) = (**void**) indica que el programa no admite argumentos (si un programa los utiliza, estos se escriben a continuación del nombre de programa cuando se le llame para su ejecución),

{ y } son el inicio y final del bloque de código.

printf orden que escribe en la pantalla los mensajes que van entre comillas ("...").

\n representa al carácter de control "nueva línea"

; indica el final de una instrucción.

La codificación no está sujeta a normas de encolumnamiento o formato.

2.2.1. Comentarios.

Los comentarios son importantes ya que por medio de ellos estaremos en posibilidades de escribir las notas pertinentes, tales como: ¿Cuál es el objetivo del proyecto o un programa escrito en lenguaje C?, ¿Qué es lo que deseamos probar?, ¿Qué algoritmo intentamos implementar?, etc. Esto nos permitirá ahorrar tiempo que se invertirá para recordar cual fue la intención primaria en el desarrollo de determinado proyecto o programa, evitando que supongamos metas u objetivos equivocados llevándonos a que inevitablemente se pierda el tiempo, programa o proyecto.

El lenguaje C acepta dos maneras por medio de las cuales escribiremos nuestros comentarios.

Comentarios en una o múltiples líneas empleando el estilo original o tradicional:

/* Este es un comentario en una línea */

/* Este también

es un comentario, pero empleando más de una línea */

El comentario inicia con "/*" y termina con "*/"

Comentarios en solo una línea usando "//" (el estilo más reciente):

// Comentario en una sola línea.

Algunos programadores emplean "//" para sus comentaros de la siguiente manera:

// Primera línea de comentarios

// Segunda línea de comentarios

// etc.

El comentario inicia con "//" y termina cuando es encontrado el final de la línea.

2.2.2. Identificadores.

Los identificadores (también conocidos como símbolos) son los nombres dados a variables, constantes, tipos de datos y funciones en los programas creados por los usuarios (programadores) y deben de ser diferentes en mayúsculas y minúsculas (aunque algunos expertos también exigen diferencias ortográficas) a cualquier palabra clave.

2.2.3. Palabras reservadas.

Las palabras reservadas del lenguaje **no** pueden ser empleadas como nombres de variables, constantes, funciones o tipos de datos, como fue hecho notar con anterioridad (en 2.2.2. Identificadores).

La Tabla 2.1 muestra las palabras reservadas del lenguaje C y esta tabla contiene dos secciones [4, p. 23]. La parte superior es la serie de palabras reservadas propuestas originalmente por Dennis Ritchie y Brian Kernighan [3, p. 10] (que fueron los diseñadores del lenguaje C, haciendo público su desarrollo en "*The C Programming Language*" en 1978), mientras que la parte inferior son palabras reservadas que introdujo el comité de normalización ANSI (*American National Standards Institute*).

Es importante hacer notar que el contenido de palabras reservadas de Tabla 2.1 no incluye las palabras reservadas propuestas por el fabricante de algún compilador de lenguaje C en particular (Microsoft, Borland, etc.).

Tabla 2.1 Palabras reservadas del lenguaje C.

PALABRAS RESERVADAS DEL LENGUAJE C		
auto	extern	short
break	float	sizeof
case	for	static
char	goto	struct
continue	if	switch
default	int	typedef
do	long	union
double	register	unsigned
else	return	while

const	signed	volatile
enum	void	

2.2.4. Tipos de datos:

2.2.4.1. Simples.

El tipo de dato simple es aquel que no permite su fragmentación en elementos más pequeños. En otras palabras, es indivisible. Los tipos de datos numéricos como *int*, *float*, *long*, *double* y *char* son ejemplos de tipos simples, no es posible fragmentarlos, tampoco mezclarlos con otro tipo de datos simple, esto es porque son tipos de datos básicos. El tipo de datos simple emplea solo un tipo de datos de manera individual. Sin embargo, es posible que sean empleados para generar tipos de datos más complejos (una colección de diferentes tipos de datos simples).

2.2.4.2. Compuestos.

El tipo de dato compuesto es obtenido por medio del agrupamiento de varios elementos de un solo tipo de dato simple (el caso más sencillo) como lo es un vector o matriz de enteros (*int*), flotantes (*float*), enteros largos (*long*), flotantes de doble precisión (*double*), etc., o un vector de caracteres también conocido como cadena (*string*).

Las estructuras (*struct* es un tipo de dato del lenguaje C) permiten la creación de tipos de datos compuestos más complejos que las matrices. Estas estructuras pueden estar formadas por enteros, flotantes, caracteres, cadenas de caracteres, todos formando un solo elemento. Algunas veces a estos agrupamientos de datos simples con los que se obtiene un tipo de dato compuesto se les conoce como registros.

2.2.5. Variables y Constantes.

Las variables y las constantes del mismo tipo de dato (por ejemplo, un entero *int*) tienen las mismas características (su tamaño en cantidad de bytes es el mismo y se almacenan en memoria RAM (*Random Access Memory* o Memoria de Acceso Aleatorio) para su uso. Sin embargo, mientras que una *variable* cambia de valor tantas veces como sea necesario en la ejecución de un programa, la *constante* conserva su valor original de inicio a fin durante la ejecución de dicho programa. En otras palabras, el valor de una constante no cambia su valor (es constante).

Una variable o constante deben ser declaradas antes de ser usadas, por ejemplo,

sí i es una variable entera su declaración sería *int i;*

sí j es una constante del tipo *float* su declaración sería *const float j=1.234;*

Esta diferencia en la declaración para las constantes es muy importante, pero muy sencilla, ya que solo debemos anteponer la palabra reservada const (de *constant* o constante). Su ubicación en un programa también es importante y crítica, como es mostrada en un ejemplo posteriormente.

2.2.6. Atributos.

Un atributo es una característica o propiedad modificable de algún elemento del programa que acepta valores diversos.

Por ejemplo, en el caso de un grupo de personas, nos pudiera interesar el color de su pelo, el color de sus ojos y su idioma, estos serían los atributos. En el caso del color de pelo que es un atributo, su valor dependería de su tonalidad.

En Unidad 6 estaremos trabajando con puertos de comunicación, e históricamente el envío y recepción de datos entre nuestro equipo de cómputo y el equipo remoto de adquisición de datos o de control, se ha efectuado considerándolos como una clase especial de archivos en los que leemos y escribimos.

Cuando necesitamos trabajar con un archivo determinado, lo primero que hacemos es buscarlo en algún directorio del disco duro o memoria. En el listado del directorio observamos tanto archivos como subdirectorios. En el caso de los archivos, no todos son visibles y probablemente no nos demos cuenta de ello, esto es porque existen archivos que están ocultos o son parte del Sistema Operativo (SO). También existen archivos que son visibles, pero no es posible que sean modificados ni borrados ya que son archivos de solo lectura. Finalmente encontramos los archivos que permiten su lectura y escritura. Las diversas variantes descritas en las características de los archivos para el sistema operativo llegan a ser atributos con alguna valoración.

La figura 2.2 muestra los atributos de los archivos referidos a sus permisos de lectura/escritura y su valoración [4, p. 155].

Otros atributos de un archivo son la fecha y la hora de la última vez que fue actualizado.

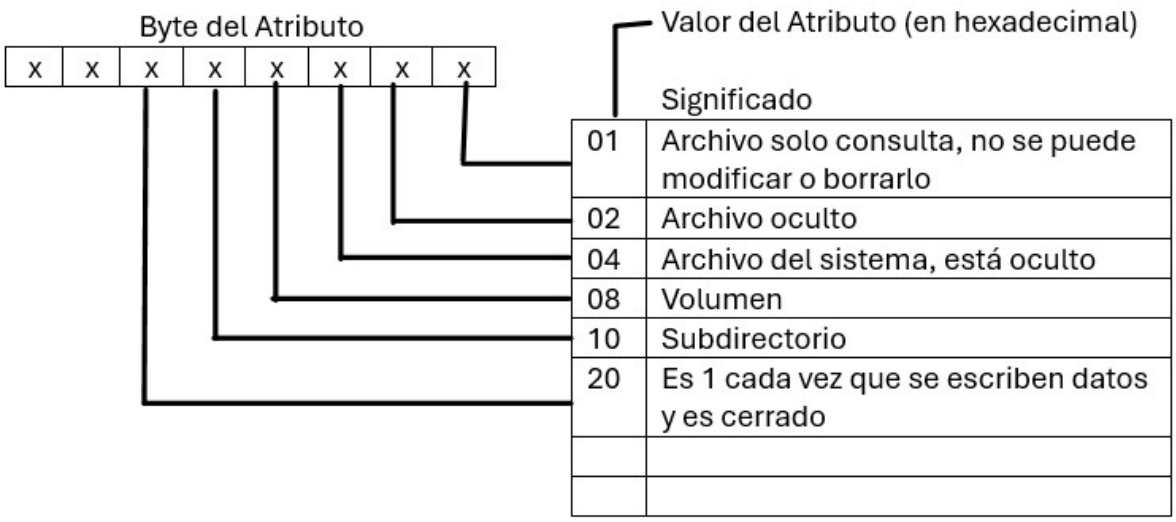


Figura 2.2 Atributos de los archivos referidos a sus permisos de lectura/escritura y su valoración.

2.2.7. Operadores

2.2.7.1. Aritméticos

Los operadores aritméticos son [4, p.34]:

+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de la división entera (Residuo)

La operación % no se aplica a los *float* ni a los *double*.

2.2.7.2. Lógicos

Orden decreciente de prioridad [4, p. 36]:

> >= < <=	Comprobaciones de mayor y menor que
== !=	Comprobaciones de igualdad y desigualdad
&& ||	Y y O (And y Or) lógicos
!	NO

El operador ! invierte una condición.

Si i > j es verdadero entonces !(i > j) es falso

Si i > j es falso entonces !(i > j) es verdadero

Si i es distinto de 0 entonces !i es igual a 0

Si i vale 0 entonces !i es distinto de 0

Si (i) equivale a si (i !=0)

Si (!i) equivale a si (i==0)

2.2.7.3. Condicionales

El operador condicional (? :) permite evaluar la veracidad o falsedad de una expresión y asignar un resultado dependiente de dicha evaluación a una variable. Este operador se dice que es un operador ternario, ya que requiere tres operandos y en nuestro caso les llamaremos C de Condición, V de Verdadera, F de Falsa y R de Resultado que es una variable del tipo adecuado para almacenar la evaluación de la condición C, entonces R=C?V:F.

Suponga que la calificación de Pedro en algún examen es de 70.0, y se desea determinar si Pedro aprobó el examen o no. El código requerido que emplea el operador ternario es como sigue:

```
float CP=70.0;              // La Calificación de Pedro
char *P= "Pasa";            // Texto si la calificación es aprobatoria
char *R= "Reprueba";        // Texto si la calificación no es aprobatoria
char Resultado[10];         // Un vector para almacenar el resultado de la evaluación
                            // del operador ternario.
(CP>=70.0)?strcpy(Resultado,P):strcpy(Resultado,R); // Evaluación de condición y
                            // copia del texto respectivo.
printf("El resultado es que Pedro %s el curso de programación", Resultado);
// Mensaje indicando el resultado de la evaluación del operador ternario
```

Este operador frecuentemente es construido por medio de sentencias else, como lo veremos en Unidad 4.

2.2.7.4. De desplazamiento [4, p. 38]

Los operadores de desplazamiento son:

<<	Desplazamiento a la izquierda
>>	Desplazamiento a la derecha

Estos operadores son muy importantes, ya que:

Aplicando un desplazamiento a la izquierda a un entero es como efectuar una operación aritmética de multiplicación por 2.

Si k = 5, este entero es representado en el sistema numérico binario por

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	1	0	1

El código necesario para realizar el desplazamiento a la izquierda por 2 sería

```
int k=5;        // declaramos k y la inicializamos a 5
int m;          // declaramos m que tomará el resultado del desplazamiento
m=k<<2;
```

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	0	1	0	0

Por lo que m=20 después de los 2 desplazamientos a la izquierda aplicados a k.

Aplicando un desplazamiento a la derecha a un entero es equivalente a efectuar una división entre 2.

Ahora sí: h=64, su representación en el sistema numérico binario es

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	0	0	0	0

El código necesario para realizar el desplazamiento a la derecha por 3 sería

```
int h=64;       // declaramos h y la inicializamos a 64
int p;          // declaramos p a la que se le asignará el resultado del desplazamiento
p=h>>3;
```

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	0	0

De aquí que p=8 después de 3 desplazamientos a la derecha aplicados a h.

2.2.8. Manejo de cadena de caracteres [4, p. 55-56]

Una cadena (*string*) se compone de caracteres alfanuméricos (texto). Es un caso particular de matriz de caracteres que se declara como tal. El final de la cadena se indica con un carácter especial, el carácter cero. Por lo tanto, el tamaño de la cadena debe ser suficiente para contener dicho carácter inclusive: una cadena declarada como **ch[10]** podrá contener como máximo 9 caracteres.

No existe ningún operador que compare dos cadenas de caracteres o copie una cadena en otra. Por ello es obligado emplear funciones de procesamiento de cadenas de caracteres. A continuación se muestra el caso particular de la inicialización de una cadena, auxiliados por un programa y resultados de su ejecución.

```c
#include<stdio.h>
#include<string.h>

void main(void){
char ch[10], ch1[10];

strcpy(ch, "Soleado"); //Copia la cadena
printf("Contenido de ch : %s \n", ch);
printf("Otra cadena \n");
gets(ch1);
if(strcmp(ch, ch1)) printf("NO son iguales \n");
}
```

```
Contenido de ch : Soleado
Otra cadena
nadar
NO son iguales

Process returned 0 (0x0)
Press any key to continue.
```

Fig. 2.3. Programa para copia y comparación de cadenas, y su corrida.

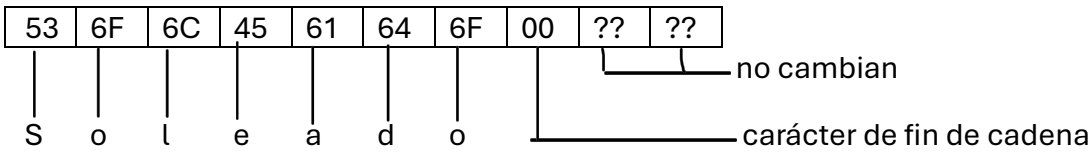
¡¡ La asignación y la comparación de cadenas no se realiza como con los enteros o los reales!!

Escribir **if(ch1==ch2)** ... es correcto sintácticamente, pero no significa que se compruebe que las cadenas **ch1 y ch2** contienen los mismos caracteres. En realidad la condición anterior comprueba si las direcciones en memoria son iguales.

La función **strcpy** (contracción de **str**ing **c**o**py**) copia el contenido de una cadena (la que especifica el segundo argumento) en otra (el primer argumento).

En el ejemplo anterior, el segundo argumento no es una variable del tipo cadena de caracteres sino una constante del mismo tipo (un texto entre comillas).

Después de la ejecución de la función **strcpy**, el contenido (en hexadecimal) de **ch** es



La función **strcmp** permite la comparación de cadenas de caracteres. La comparación se realiza solo hasta el carácter 0x00 de final de cadena, pero no más allá.

Es necesario distinguir claramente entre **'A'** y **"A"**. **'A'** designa el carácter (char) A que se codifica con un byte. **"A"** designa una cadena de caracteres. En realidad, desde el punto de vista del usuario solo contiene un carácter, pero se ha codificado con dos bytes, el segundo es el carácter 0x00 que indica el fin de la cadena.

```c
#include <stdio.h>
#define Largo 20

int main(void)
{
char c[Largo];
char ci[Largo];
int i=0,j;

printf("Generare una cadena caracter por caracter\n");
while(i<(Largo-1))
{
j=getche();
if(j==13) break;
else c[i++]=j;
}
c[i]=0;
printf("\nLa cadena generada es : %s\n",c);
printf("\nAhora invertire la cadena\n");

j=0;
ci[i--]=0;

while(i>-1) ci[j++]=c[i--];
printf("La cadena invertida es: %s\n",ci);

getch();
return 0;
}
```

```
Generare una cadena caracter por caracter
Almazan Cuellar S
La cadena generada es : Almazan Cuellar S

Ahora invertire la cadena
La cadena invertida es: S ralleuC nazamlA
```

Fig. 2.4. Programa para generar una cadena por medio de la captura de caracteres, su inversión, y su corrida.

3. Arreglos

3.1. Definición e importancia de los arreglos en la programación.

Un arreglo (matriz) es un conjunto de elementos dispuestos secuencialmente que contienen (o pueden contener) datos del mismo tipo.

El tamaño de un arreglo (matriz) es su número de elementos. El tamaño se debe fijar cuando se declare el arreglo (matriz) (el tamaño y el tipo determinan el tamaño de memoria que se destinará al arreglo).

Un arreglo pude ser de una o varias dimensiones (este caso es el más común). No hay límite (excepto el que determine físicamente la memoria de su equipo) para el número de dimensiones.

3.2. Declaración de arreglos unidimensionales y multidimensionales [4, pp. 53-54].

Declaremos un arreglo unidimensional que pueda contener 4 enteros:

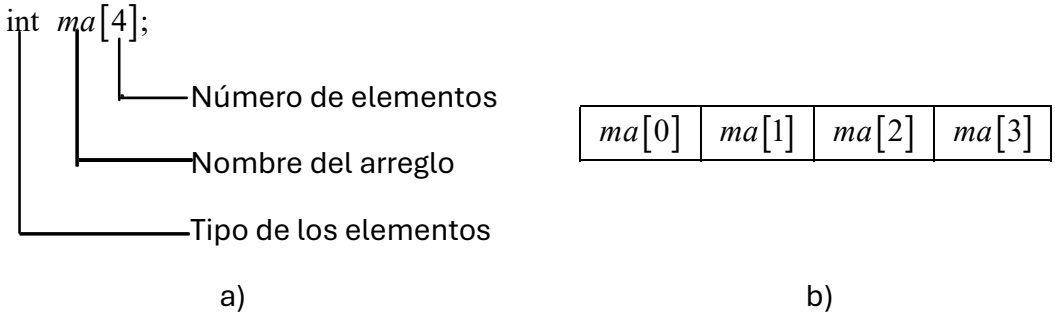


Fig. 3.1 a) Declaración de un arreglo unidimensional de 4 elementos, b) Disposición en la memoria de los elementos del arreglo en a).

Un arreglo bidimensional de 3 filas por 2 columnas contendrá 3*2=6 elementos del tipo float y es declarada e inicializada como sigue:

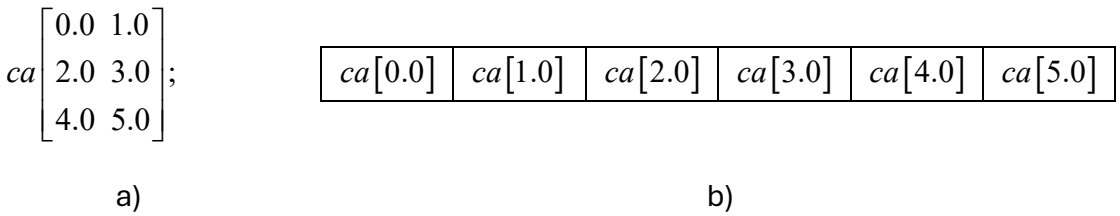


Fig. 3.2 a) Empleo de un arreglo bidimensional de 3 filas por 2 columnas (3*2= 6 elementos), b) Disposición en la memoria de los elementos del arreglo en a).

float ca[3][2];

$$float\ ca[3][2] = \{\{0.0,\ 1.0\},\ \{2.0,\ 3.0\},\ \{4.0,\ 5.0\}\};$$

En el caso de arreglos con dimensiones superiores a 2 (bidimensionales) el problema pareciera se vuelve más complejo, pero la verdad es que no es así. Lo que se requiere es un poco más de atención. Vamos con una analogía, y para facilitar las cosas pensemos que la variable que manejaremos son caracteres (char).

1) Pensemos en un arreglo unidimensional cuya variable es el carácter, con capacidad para 70 de ellos. Imaginemos que es un renglón (una fila con 70 caracteres o 70 columnas).
2) Supongamos que tenemos muchos renglones y como somos muy ordenados, decidimos hacer grupos con cantidades iguales de renglones también muy en orden, y les llamamos página a cada grupo. Pensemos que cada página contiene 40 renglones (40 filas y cada fila tiene 70 caracteres).
3) Finalmente, de manera ordenada agrupamos las páginas, ese conjunto de páginas consta de la fabulosa cantidad de 5, 5 páginas.

El resultado final es que tenemos un libro, en el que contamos por ejemplo, la historia de nuestra infancia, adolescencia o juventud.

En términos de programación lo que tenemos es un arreglo tridimensional con las siguientes características.

$char\ \ milibro[40][70][5]$.

Equivalentemente es lo mismo que decir: Se tiene un libro de 5 páginas y cada página cuenta con 70 caracteres por renglón y contiene 40 renglones por página.

Un ejemplo sencillo y práctico de un arreglo tridimensional el cual es declarado como sigue:

$short\ \text{int}\ mitriple[2][2][3];$ y empleando la terminología del ejemplo inmediato anterior decimos que tenemos un arreglo tridimensional de 2 filas, 2 columnas y 3 páginas (2*2*3=12) en total 12 elementos. Su inicialización llega a ser:

$$short\ \text{int}\ mitriple[2][2][3] = \{\ \{\{1,\ 1\},\ \{2,\ 2\},\ \{3,\ 3\}\},\ \{\{1,\ 1\}\ \{2,\ 2\}\}\ \{3,\ 3\}\}\ \};$$

1	1	1	1	2	2	2	2	3	3	3	3

Fig. 3.3 Disposición en la memoria de los elementos del arreglo *mitriple*.

Como habrá notado, en todos los casos, los arreglos unidimensionales, bidimensionales y tridimensionales, y en general hablando de arreglos, todos sus elementos están dispuestos en línea, de ahí la expresión de memoria plana (flat). Precisamente este es el punto de apoyo para decir que el significado de columna, renglón, etc. se lo damos por medio de la programación.

En algunas áreas del conocimiento a la tercera dimensión se le conoce como profundidad, y no es difícil que estemos de acuerdo con el término si tomamos a Cálculo Vectorial como referencia. Tome el eje x sobre la vertical positiva, al eje y tómelo sobre la horizontal positiva aplique la regla de la mano derecha, esto dará como resultado que z se dirige al interior de alguna página sobre la cual se encuentran los ejes x y y. Esta es la justificación al término profundidad.

3.3. Lectura de arreglos unidimensionales y multidimensionales.

Para materializar lo útil de los arreglos primero deben ser declarados (como ocurre con cualquier variable o constante) e inicializados. Para el arreglo unidimensional de sección 3.2 $\text{int } ma[4];$ la inicializamos con valores arbitrarios ma[4]={3,2,1,0};

De manera que podemos leer alguno de sus elementos por medio de int a=ma[2]; a tomará el valor de 1. Así, si a=ma[0], a será igual a 3. (Todo índice inicia en cero).

El arreglo bidimensional float ca[3][2]; fue previamente inicializado, por lo que si deseamos el elemento float b=ca[1][1], localizamos dicho elemento en la fila 1 columna 1, esto nos lleva a concluir que b=3. (Todo índice inicia en cero).

En el caso del arreglo tridimensional, si deseamos obtener el elemento de fila cero, columna 1 y página 2 (profundidad 2) tendremos short int c = mitriple[0][1][2] lo cual resulta en que c=3. (Todo índice inicia en cero).

3.4. Operaciones con arreglos.

Las operaciones matemáticas básicas con arreglos unidimensionales (vectores) son sencillas. A continuación se muestra en figura 3.4 el texto de un programa por medio del cual se realiza la captura de dos arreglos a y b, posteriormente se realiza la suma, resta y producto entre dichos arreglos. Es importante notar que estas operaciones son elemento a elemento entre los vectores (arreglos unidimensionales).

```c
#include<stdio.h>
#include<string.h>
void main (void){
short int a[10], b[10];
short int i,j,d;
char buf[10];

printf("Los vectores a y b tienen una longitud maxima de 10 elementos !!!\n");
printf("Captura de vector a \n");
for(i=0; i<10; i++){
    printf("\nDame elemento #%d de :a ",i);
    gets(buf);
    sscanf(buf,"%d",&d); a[i]=d;
}
for(i=0; i<10; i++) printf("%d  ",a[i]);

printf("\nCaptura de vector b \n");
for(i=0; i<10; i++){
    printf("\nDame elemento #%d de :b ",i);
    gets(buf);
    sscanf(buf,"%d",&b[i]);
}
for(i=0; i<10; i++) printf("%d  ",b[i]);

printf("\nLa suma de a+b es :\n");
for(i=0; i<10; i++) printf("%d  ",a[i]+b[i]);

printf("\nLa resta de a-b es :\n");
for(i=0; i<10; i++) printf("%d  ",a[i]-b[i]);

printf("\nEl producto entre a*b es :\n");
for(i=0; i<10; i++) printf("%d  ",a[i]*b[i]);
}
```

Fig. 3.4 Listado del programa con el que se obtiene la Suma, Resta y Producto entre arreglos unidimensionales (vectores).

```
Los vectores a y b tienen una longitud maxima de 10 elementos !!!
Captura de vector a
1  1  1  1  1  1  1  1  1  1
Captura de vector b
2  2  2  2  2  2  2  2  2  2
La suma de a+b es :
3  3  3  3  3  3  3  3  3  3
La resta de a-b es :
-1  -1  -1  -1  -1  -1  -1  -1  -1  -1
El producto entre a*b es :
2  2  2  2  2  2  2  2  2  2
```

Fig. 3.5 Corrida del programa mostrado en Fig. 3.4 donde se omiten las secciones de captura por razones de espacio.

El listado siguiente (figura 3.6) corresponde a la ejecución de operaciones matemáticas básicas con arreglos bidimensionales. Se realiza la captura de dos

arreglos bidimensionales, para posteriormente efectuar la suma, resta y producto entre dichos arreglos.

Para la operación del producto matricial se hace uso de 2 vectores auxiliares, cuya función es la de almacenar un renglón (fila) de un arreglo o alguna columna del arreglo restante. También, es necesario un arreglo bidimensional auxiliar para almacenar los resultados de la suma de productos entre filas y columnas de los vectores auxiliares. Al final de este proceso, dicho arreglo bidimensional auxiliar contendrá el resultado del producto matricial, que es el resultado buscado.

```c
#include<stdio.h>
#include<string.h>
void main (void){
short int A[3][3], B[3][3], C[3][3];
short int i, j, r, c, m, s, d;
short int rg[3], co[3];
char buf[10];

printf("Los arreglos bidimensionales A y B ");
printf("son cuadrados 3x3 \n");
printf("Captura de arreglo A \n");

for(i=0; i<3; i++){
    for(j=0;j<3;j++){
    printf("\nDame elemento %d, %d de :A ",i,j);
    gets(buf);                  // Captura una cadena
    sscanf(buf,"%d",&d); A[i][j]=d; // Convierte la cadena a decimal y la asigna a A[i][j]
    }
}
printf("Captura de arreglo B \n");

for(i=0; i<3; i++){
    for(j=0;j<3;j++){
    printf("\nDame elemento %d, %d de :B ",i,j);
    gets(buf);
    sscanf(buf,"%d",&d); B[i][j]=d;
    }
}
printf("La suma de A+B\n");  // La suma y resta matricial es elemento a elemento
for(i=0;i<3;i++){
    for(j=0;j<3;j++) printf("%d\t",A[i][j]+B[i][j]);
    printf("\n");
}
printf("La resta de A-B\n");
for(i=0;i<3;i++){
    for(j=0;j<3;j++) printf("%d\t",A[i][j]-B[i][j]);
    printf("\n");
}
printf("El producto matricial de A*B\n"); // El producto matricial es la suma
                                          // de los productos entre fila A y
for(i=0; i<3; i++){                       // columna en B
    for(c=0; c<3; c++)rg[c]=A[i][c];      // Obtine la fila de A
        for(j=0; j<3; j++){
            s=0;                          // De inicio la suma es CERO, s=0
            for(r=0; r<3; r++) co[r]=B[r][j];  //obtiene la columna de B
                for(m=0; m<3; m++) s+=rg[m]*co[m]; // Hace la suma de Prod
                    C[i][j]=s; // La suma de productos se asigna a C
        }// rg y co son vectores auxiliares, C es una matriz auxiliar

}
    for(i=0;i<3;i++){
        for(j=0;j<3;j++) printf("%d\t", C[i][j]);
        printf("\n");
    }
```

Fig. 3.6 Listado del programa con el que se obtiene la Suma, Resta y Producto entre arreglos bidimensionales.

```
Los arreglos bidimensionales A y B son cuadrados 3x3
Captura de arreglo A            Captura de arreglo B

Dame elemento 0, 0 de :A 1      Dame elemento 0, 0 de :B 3

Dame elemento 0, 1 de :A 1      Dame elemento 0, 1 de :B 3

Dame elemento 0, 2 de :A 1      Dame elemento 0, 2 de :B 3

Dame elemento 1, 0 de :A 2      Dame elemento 1, 0 de :B 2

Dame elemento 1, 1 de :A 2      Dame elemento 1, 1 de :B 2

Dame elemento 1, 2 de :A 2      Dame elemento 1, 2 de :B 2

Dame elemento 2, 0 de :A 3      Dame elemento 2, 0 de :B 1

Dame elemento 2, 1 de :A 3      Dame elemento 2, 1 de :B 1

Dame elemento 2, 2 de :A 3      Dame elemento 2, 2 de :B 1

La suma de A+B          La resta de A-B         El producto matricial de A*B
4       4       4       -2      -2      -2      6       6       6
4       4       4       0       0       0       12      12      12
4       4       4       2       2       2       18      18      18
```

Fig. 3.7 Corrida del programa mostrado en figura 3.6

En la figura 3.8 se muestra el listado del programa empleado para obtener las operaciones matriciales elementales en arreglos tridimensionales. La suma, resta y producto se efectúan en páginas (profundidades) diferentes, seleccionadas del mismo arreglo (3D).

Comparando los códigos para arreglos bidimensionales y tridimensionales no es complejo concluir que son muy similares, y que la diferencia entre un listado y otro es la estructura de selección múltiple Switch-Case, la cual facilita enormemente la codificación para la selección del par de páginas de interés, dicho sea de paso, es más práctica y clara que haber empleado una serie de combinaciones if, else if, etc.

```c
#include<stdio.h>
#include<string.h>
void main (void){
short int T[3][3][3], C[3][3];
short int co[3], rg[3];
short int i, j, k, o, c, d, m, r, s, p0, p1;
char buf[10];
// Captura de datos para el arreglo tridimensional
for (k=0;k<3;k++){
for(i=0; i<3; i++){
    for(j=0;j<3;j++){
    printf("Dame elemento %d, %d de :T%d ",i,j,k);
    gets(buf);                  // Captura una cadena
    sscanf(buf,"%d",&d); // Convierte la cadena a decimal y la asigna a T[i][j][k]
    T[i][j][k]=d;
    }
}
for(i=0;i<3;i++){
        for(j=0;j<3;j++) printf("%d\t", T[i][j][k]);
        printf("\n");
    }
}
printf("Selecciona 2 pag. entre las que se realizaran las op. matematicas\n");
printf("0 op(0,1) \n");
printf("1 op(0,2) \n");
printf("2 op(1,2) \n");
gets(buf);
sscanf(buf,"%d",&o);
switch (o){
    case 0: printf("Seleccionaste 0 op(0,1) \n"); p0=0; p1=1; break;
    case 1: printf("Seleccionaste 1 op(0,2) \n"); p0=0; p1=2; break;
    case 2: printf("Seleccionaste 2 op(1,2) \n"); p0=1; p1=2; break;
    default:o=0; printf("El default es 0 op(0,1) \n"); p0=0; p1=1;
}
printf("La suma de T[i][j][%d] + T[i][j][%d]\n", p0, p1);
          // La suma y resta matricial es elemento a elemento
for(i=0;i<3;i++){
    for(j=0;j<3;j++) printf("%d\t",T[i][j][p0]+T[i][j][p1]);
    printf("\n");
}
printf("La resta de T[i][j][%d] - T[i][j][%d]\n", p0, p1);
for(i=0;i<3;i++){
    for(j=0;j<3;j++) printf("%d\t",T[i][j][p0]-T[i][j][p1]);
    printf("\n");
}
printf("El producto matricial de T[i][j][%d] * T[i][j][%d]\n", p0, p1);
// El producto matricial es la suma de los productos entre fila y
for(i=0; i<3; i++){                                    // columna
    for(c=0; c<3; c++)rg[c]=T[i][c][p0];      // Obtine la fila
        for(j=0; j<3; j++){
            s=0;                          // De inicio la suma es CERO, s=0
            for(r=0; r<3; r++) co[r]=T[r][j][p1];  //Obtiene la columna
                for(m=0; m<3; m++) s=s+rg[m]*co[m]; // Hace la suma de Prod
```

Fig. 3.8 Listado del programa con el que se obtiene la Suma, Resta y Producto entre páginas del arreglo tridimensional. (Continúa)

```
54                      C[i][j]=s; // La suma de productos se asigna a C
55          }// rg y co son vectores auxiliares, C es una matriz auxiliar
56      }
57      for(i=0;i<3;i++){
58          for(j=0;j<3;j++) printf("%d\t", C[i][j]);
59          printf("\n");
60      }
61  }
62
```

Fig. 3.8 Listado del programa con el que se obtiene la Suma, Resta y Producto entre páginas del arreglo tridimensional. (Continuación)

```
Dame elemento 0, 0 de :T0 1
Dame elemento 0, 1 de :T0 2
Dame elemento 0, 2 de :T0 3
Dame elemento 1, 0 de :T0 4
Dame elemento 1, 1 de :T0 5
Dame elemento 1, 2 de :T0 6
Dame elemento 2, 0 de :T0 7
Dame elemento 2, 1 de :T0 8
Dame elemento 2, 2 de :T0 9
1       2       3
4       5       6
7       8       9
Dame elemento 0, 0 de :T1 9
Dame elemento 0, 1 de :T1 8
Dame elemento 0, 2 de :T1 7
Dame elemento 1, 0 de :T1 6
Dame elemento 1, 1 de :T1 5
Dame elemento 1, 2 de :T1 4
Dame elemento 2, 0 de :T1 3
Dame elemento 2, 1 de :T1 2
Dame elemento 2, 2 de :T1 1
9       8       7
6       5       4
3       2       1
Dame elemento 0, 0 de :T2 1
Dame elemento 0, 1 de :T2 1
Dame elemento 0, 2 de :T2 1
Dame elemento 1, 0 de :T2 1
Dame elemento 1, 1 de :T2 1
Dame elemento 1, 2 de :T2 1
Dame elemento 2, 0 de :T2 1
Dame elemento 2, 1 de :T2 1
Dame elemento 2, 2 de :T2 1
1       1       1
1       1       1
1       1       1
Selecciona 2 pag. entre las que se realizaran las op.  matematicas
0 op(0,1)
1 op(0,2)
2 op(1,2)
0
Seleccionaste 0 op(0,1)
La suma de T[i][j][0] + T[i][j][1]
10      10      10
10      10      10
10      10      10
La resta de T[i][j][0] - T[i][j][1]
-8      -6      -4
-2      0       2
4       6       8
          El producto matricial de T[i][j][0] * T[i][j][1]
          30      24      18
          84      69      54
          138     114     90

          Process returned 3 (0x3)
          Press any key to continue.
```

Fig. 3.9 Corrida del programa mostrado en figura 3.8.

En la figura 3.10 se muestra el listado del programa para calcular el Producto Cruz. Para tal efecto empleamos el método de cofactores del cual aprovechamos su enfoque vectorial-matricial. Requerimos 2 vectores (3D): A y B.

```c
#include <stdio.h>
int main(void){
int i,d=3;
float b,c,VA[3],VB[3],VC[3];

printf("Con este programa obtienes el producto CRUZ o externo entre A y B\n");
printf("Las dimension de A o B siempre son 3\n");
printf("Obtenemos el producto CRUZ por medio de cofactores \n");

i=0;
do{
printf("Dame la componente a%d ",i);
scanf("%f",&c);
VA[i]=c;
printf("Dame la componente b%d ",i);
scanf("%f",&b);
VB[i]=b;
i++;
}while(i<d);
VC[0]=VA[1]*VB[2]-VA[2]*VB[1];
VC[1]=VA[0]*VB[2]-VA[2]*VB[0];
VC[2]=VA[0]*VB[1]-VA[1]*VB[0];

printf("El producto CRUZ es \n");
printf("%f i\t %f j\t %f k \n",VC[0],-1.0*VC[1],VC[2]);
printf("Y los vectores fueron \n");
printf("A\t\t\t\tB\n");
for(i=0;i<d;i++) printf("a%d=%f\t\tb%d=%f\n",i,VA[i],i,VB[i]);
getch();
return 0;
}
```

Fig. 3.10 Listado del programa para el cálculo del Producto Cruz

Capturamos las componentes de cada vector empleando una estructura do-while, aplicamos cofactores para obtener el Producto Cruz mostrando el resultado y los vectores participantes en el cálculo.

```
Con este programa obtienes el producto CRUZ o externo entre A y B
Las dimension de A o B siempre son 3
Obtenemos el producto CRUZ por medio de cofactores
Dame la componente a0 1
Dame la componente b0 2
Dame la componente a1 -1
Dame la componente b1 3
Dame la componente a2 2
Dame la componente b2 1
El producto CRUZ es
-7.000000 i      3.000000 j      5.000000 k
Y los vectores fueron
A                               B
a0=1.000000             b0=2.000000
a1=-1.000000            b1=3.000000
a2=2.000000             b2=1.000000

Process returned 0 (0x0)   execution time : 75.430 s
Press any key to continue.
```

Fig. 3.11 Corrida del programa para el cálculo del Producto Cruz.

```c
#include <stdio.h>
int main(void){
int i,d;
float a,b,c,VA[10],VB[10];

printf("Con este programa obtienes el producto PUNTO o interno entre A y B\n");
printf("Dame la dimension de A o B : ");
scanf("%d",&d);
i=1;
a=0.0;//suma de productos
do{
printf("Dame la componente a%d ",i);
scanf("%f",&c);
VA[i]=c;
printf("Dame la componente b%d ",i);
scanf("%f",&b);
VB[i]=b;
a+=c*b;
i++;
}while(i<=d);
printf("El producto punto es %f\n",a);
printf("Y los vectores fueron \n");
printf("A\t\t\t\tB\n");
for(i=1;i<=d;i++) printf("a%d=%f\t\tb%d=%f\n",i,VA[i],i,VB[i]);
getch();
return 0;
}

```

Fig. 3.12 Listado del programa para el cálculo del Producto Punto.

La figura 3.12 muestra el listado del programa empleado para calcular el Producto Punto. Este producto es empleado con frecuencia, ya que por medio de el es muy

simple obtener la magnitud al cuadrado de algún vector sin importar la dimensión en la que se encuentre.

El programa captura cada una de las componentes de los vectores participantes en el cálculo empleando una estructura do-while. Cada vez que se obtiene el par de componentes es actualizado el acumulado de los productos. Una vez que concluye este proceso, se muestra el resultado del producto punto calculado, así como las componentes de los vectores participantes.

```
Con este programa obtienes el producto PUNTO o interno entre A y B
Dame la dimension de A o B : 3
Dame la componente a1 1
Dame la componente b1 -2
Dame la componente a2 1
Dame la componente b2 2
Dame la componente a3 -1
Dame la componente b3 1
El producto punto es -1.000000
Y los vectores fueron
A                              B
a1=1.000000             b1=-2.000000
a2=1.000000             b2=2.000000
a3=-1.000000            b3=1.000000

Process returned 0 (0x0)
Press any key to continue.
```

Fig. 3.13 Corrida del programa para obtener el Producto Punto.

4. Estructuras de control

Las estructuras de control brindan flexibilidad y versatilidad a un programa, ya que por medio de estas, dependiendo de las condiciones de operación dadas, el flujo de instrucciones a ejecutar tomará alguna dirección entre las diversas opciones que la complejidad del problema a resolver requiera. Para situaciones similares a la recientemente descrita, las sentencias IF, IF/ELSE, IF/ELSEIF serán imprescindibles. Por otro lado, las sentencias SWITCH/CASE son de gran ayuda en el manejo y procesamiento de las decisiones que el usuario tome por medio de menús que el programador diseña y le muestra. Los resultados del programa dependerán de las decisiones (selecciones) del usuario.

En ingeniería, los ciclos FOR que son ciclos de repeticiones fijas y los ciclos WHILE y DO WHILE que son ciclos de repeticiones condicionadas, son muy importantes.

En la práctica diaria requerimos efectuar productos matriciales, esto es la selección de la fila de una matriz (llámela A) y de una columna de otra matriz (puede ser B), realizar el producto elemento a elemento entre dicha fila y columna, para finalmente realizar la suma de productos dando como resultado algún elemento de una matriz que será nuestro producto matricial.

Los ciclos WHILE y DO WHILE (son ciclos de repeticiones condicionadas, son muy similares, su diferencia será explicada en breve) es posible emplearlos para calcular factoriales (n!), minimizar el error de alguna función por medio de la aproximación de uno de sus parámetros, ajustar algún parámetro de un equipo de control para obtener una respuesta deseada, etc.

4.1. Estructuras de selección [4, pp. 43-46].

IF (Si)

<table>
<tr><td colspan="2">Si … entonces una instrucción</td></tr>
<tr><td>if (condición) instrucción 1;</td><td>if (i>0) printf (“positivo \n”);</td></tr>
<tr><td colspan="2">Si … entonces varias instrucciones</td></tr>
<tr><td>if (condición)
{
instrucción 1;
…
instrucción n;
}</td><td>if (i>0)
{
printf (“i es positivo \n”);
j=0;
}</td></tr>
</table>

IF/ELSE

<table>
<tr><td colspan="2">Si … entonces una sola instrucción
Si … no una sola instrucción</td></tr>
<tr><td>if (condición) instrucción 1;
else instrucción 2;</td><td>if (i>0) printf (“i es positivo \n”);
else printf (“i <= 0 \n”);</td></tr>
<tr><td colspan="2">Si … entonces una sola instrucción
Si … no varias instrucciones</td></tr>
<tr><td>if (condición) instrucción 1;
else
{
instrucción 2;
…
instrucción n;
}</td><td>if (i>0) printf (“i es positivo \n”);
else
{
printf (“i <= 0 \n”);
j=0;
}</td></tr>
<tr><td colspan="2">Si … entonces varias instrucciones
Si … no una sola instrucción</td></tr>
<tr><td>if (condición) {
instrucción 1;
…
instrucción 2;
}
else instrucción 3;</td><td>if (i>0)
{
printf (“i es positivo \n”);
j = 1;
}
else printf (“i <= 0 \n”);</td></tr>
<tr><td colspan="2">Si … entonces varias instrucciones
Si … no varias instrucciones</td></tr>
<tr><td>if (condición) {
instrucción 1;
…
instrucción 2;
}
else
{ instrucción 3;
…
instrucción 4;
}</td><td>if (i>0)
{
printf (“i es positivo \n”);
j = 1;
}
else
{ printf (“i <= 0 \n”);
j=0;
}</td></tr>
</table>

Programa-Práctica-Análisis

```c
#include <stdio.h>

void main(void){
int i,j;
printf("Escribir i y j\n");
scanf("%d %d", &i, &j);
if(i>j) printf("i es mayor que j \n");
if(i>j) printf("i es mayor que j \n");
else printf("i es menor o igual a j \n");

if(i>j)
{
    printf("i es mayor");
    printf(" que j \n" );
}
else{
    printf("i es menor");
    printf(" o igual a j \n");
}
if(i>j) printf("i es mayor que j \n");
else
{
    printf("i es menor");
    printf(" o igual a j \n");
}
if(i>j)
    {
    printf("i es mayor");
    printf(" que j \n" );
    }
else
    printf("i es menor o igual a j \n");
if(i==j) printf("i es igual que j\n" );
if(i!=j) printf("i no es igual que j \n");
}
```

```
Escribir i y j
2
3
i es menor o igual a j
i es menor o igual a j
i es menor o igual a j
i es menor o igual a j
i no es igual que j

Process returned 0 (0x0)
Press any key to continue.
```

Fig. 4.1 Programa y prueba para Estructuras de selección (IF).

RECUERDE QUE:
= es el signo de asignación Mientras que == comprueba la igualdad

IF/ELSEIF

ELSEIF es una contracción de ELSE e IF, su funcionalidad es la misma. En nuestro caso tomando en cuenta Tabla 2.1 que indica que ELSE e IF son palabras reservadas, emplearemos la forma no contraída (ELSE IF).

Existen varias maneras de codificar else if, algunas más legibles que otras. Mostraremos 3 de ellas.

Forma 1:

```
    if(condición 1) instrucción1;
else if(condición 2) instrucción 2;
else if(condición 3) instrucción 3;
else instrucción 4;
```

Forma 2:

```
If(condición 1) instrucción 1;
        else if (condición 2) instrucción 2;
                else if(condición 3) instrucción 3;
                        else instrucción 4;
```

Forma 3:

```
if(condición 1) instrucción 1;
        else{
            if(condición 2) instrucción 2;
                        else {
                                if(condición 3) instrucción 3;
                                                else instrucción 4;
                                 }
```

}

En las tres formas, existe lo que se conoce como condiciones anidadas (una dentro de otra), sin embargo un else siempre hace referencia al último if ejecutado.

Como ejemplo, en el bloque

```
if (n>0)
  if(a>b) z=a;
     else z=b;
```

else hace referencia al segundo caso de if(a>b).

Si deseamos que se haga referencia a if(n>0), se deberá escribir

```
if(n>0) {if(a>b) z=a;}
        else z=b;
```

Otra forma de escribir

```
if(a>b) z=a;
        else z=b;
```

sería empleando el operador condicional (operador ternario) de Unidad 2.

```
z=(a>b)?a: b;
```

4.2. Estructuras de repetición [4, pp. 48-51].

WHILE

Mientras sea cierta la condición expresada entre paréntesis se efectuará la (o las) instrucción (instrucciones). La condición se evalúa cada vez antes de que se ejecute la primera instrucción del ciclo.

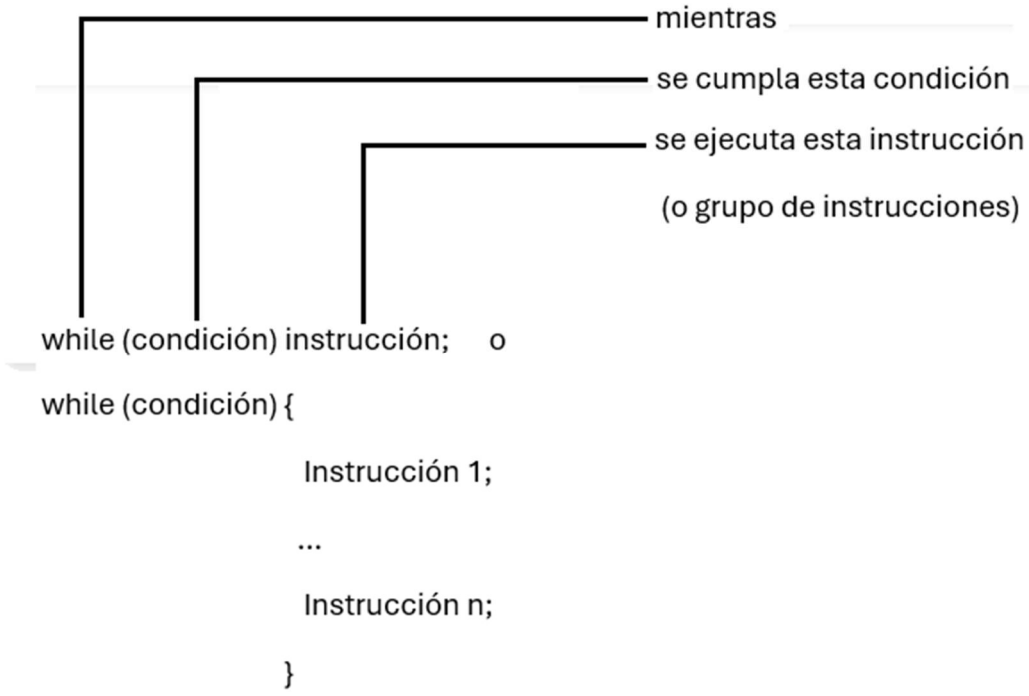


Fig. 4.1 Ciclo mientras que (WHILE).

DO WHILE

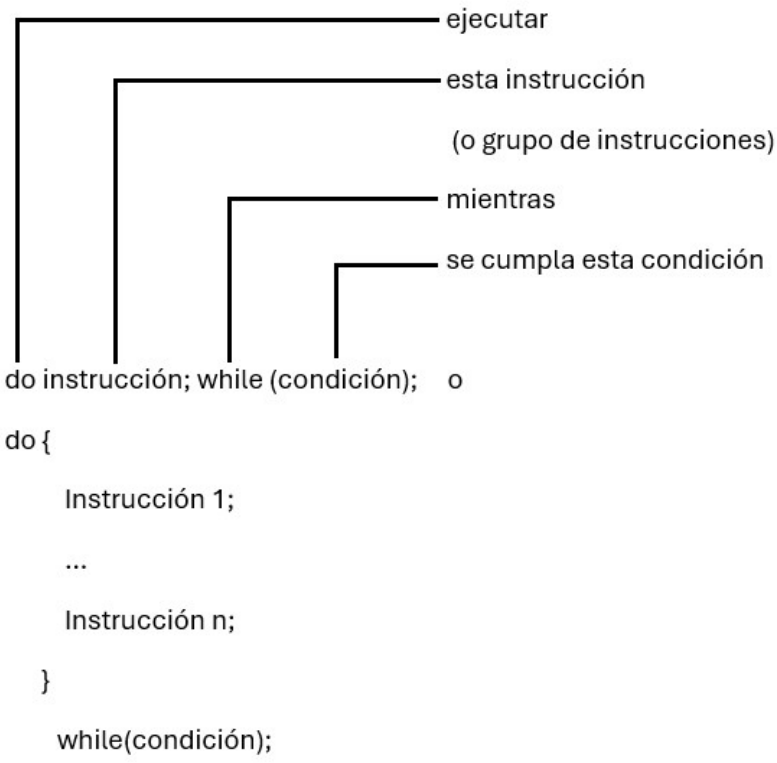


Fig. 4.2 Ciclo hacer ... mientras (DO WHILE)

En una estructura **do while**, la condición, que siempre se especifica entre paréntesis, se evalúa al final del ciclo. Así pues, las instrucciones se ejecutan por lo menos una vez.

Programa-Práctica-Análisis

```
// Ciclo while
#include<stdio.h>
void main(void){
int i=2, j=7;
while(i<j)
{
   printf("i : %d y j ; %d \n", i, j);
   i++;j--;
}
}
```

La instrucción puede escribirse como:

```
        while (i<j) printf (“i : %d y j ; %d \n", i++, j--);
```

Programa-Práctica-Análisis

```
// Ciclo do while
#include<stdio.h>
void main(void){
int i=2, j=0;
do{
   printf("i : %d y j : %d \n", i, j);
   i++; j++;
}
while(i<j);
}
```

FOR

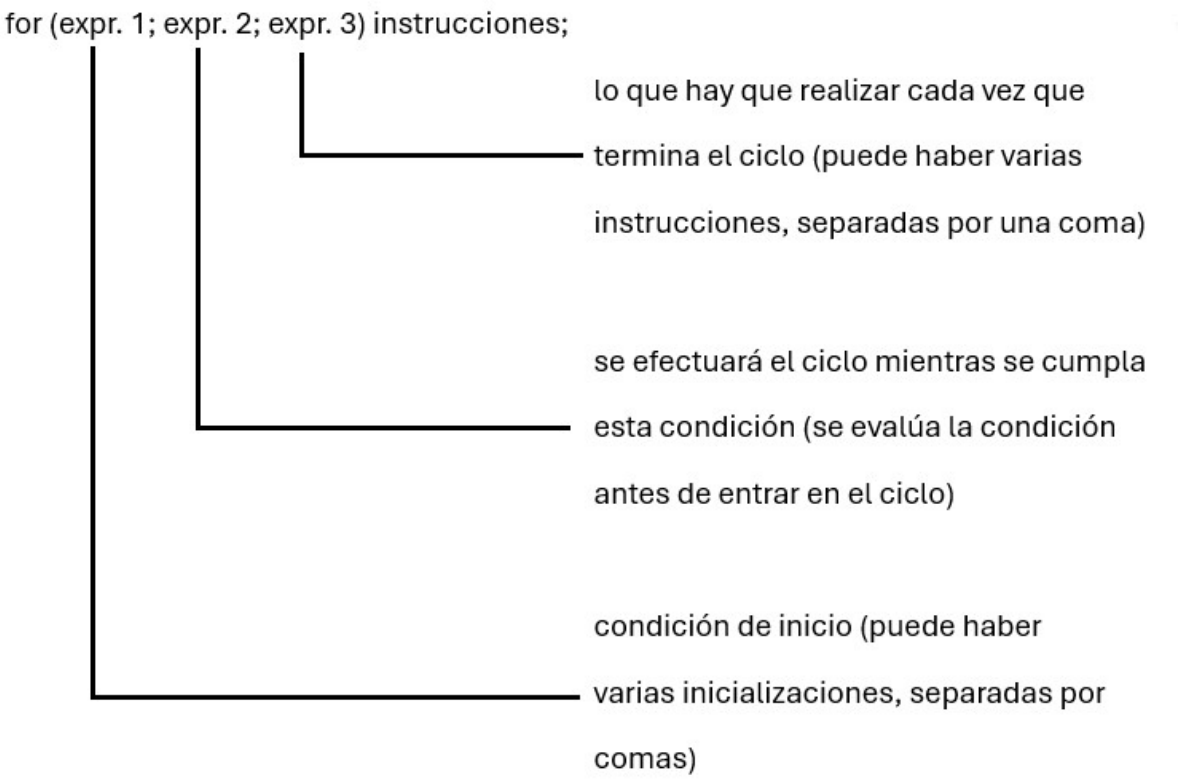


Fig. 4.3 Ciclo de número fijo de repeticiones (FOR)

Programa-Práctica-Análisis

```
// Ciclo for
#include<stdio.h>
void main(void){
int i;
for(i=2;i<5;i++) printf("%d \n", i);
}
```

Programa-Práctica-Análisis

```
// Ciclo for (2do)
#include<stdio.h>
void main(void){
int i, j;
for(i=2, j=4; i<5 && j>2; i++, j--)
```

```
   printf("i : %d y j : %d\n", i, j);
}
```

4.3. Estructuras de múltiple selección SWITCH/CASE [4, pp. 46-47].

La instrucción switch direcciona a distintas instrucciones según el valor que tome una variable de control (que solo puede ser int, long o carácter). Esta variable de control es también conocida como selector (del switch). La instrucción break es la única que permite salir del switch.

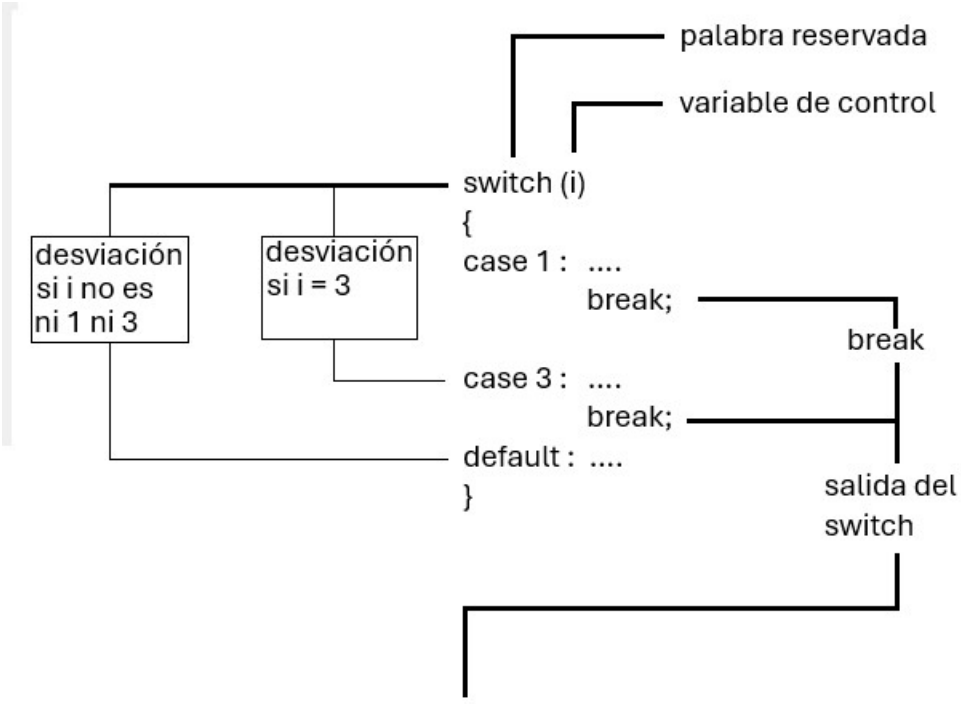


Fig. 4.4 Estructura de múltiple selección (SWITCH/CASE)

Programa-Práctica-Análisis

```
#include<stdio.h>
void main(void){
int i;
printf("Escriba i \n");
scanf("%d", &i);
switch (i)
```

```
{
   case 0: printf("cero \n"); break;
   case 1:
   case 2: printf("Un 1 o un 2 \n"); break;
   case 4: break; // si es 4 no escribe
   case 5:
   case 6: printf("Un 5 o un 6\n"); break;
   default: printf("Otros casos \n");
}
}
```

4.4. Formulación y aplicación de algoritmos utilizando estructuras de control

Como fue hecho notar con anterioridad, en ingeniería es frecuente la implementación de segmentos de código o funciones con las cuales obtengamos el factorial de algún número entero (siempre).

Todos sabemos que el factorial del número n (n!) es el producto de la serie 1*2*3...*n.

A continuación se presentan 3 listados con los cuales n! es obtenido empleando los 3 diferentes ciclos de repetición expuestos en esta Unidad (FOR, WHILE y DO WHILE), todo con el objetivo de mostrar que la solución de un problema no es más fácil o difícil con un ciclo en particular. Todo dependerá del que tan claro se tiene el problema a resolver, el planteamiento de la solución junto con la generación de su respectivo código. Posteriormente inician las pruebas, añadiendo y corrigiendo lo necesario. Finalmente optimizamos por funcionamiento (resultados) y legibilidad sustituyendo secciones de código que son (o serán) nuestras funciones (Unidad 5) cada una dedicada a resolver una pequeña parte de algún problema.

Programa-Práctica-Análisis

```
#include<stdio.h>
void main(void){        // Calcula el factorial de n
int i, n=5, f=1;        // i es el contador, n es el número del cual se desea el factorial
char b[10];             // área de memoria al capturar n
```

```
printf ("Dame el número del cual deseas su factorial ");
gets(b);                    // se obtiene una cadena que contiene a n
sscanf(b, "%i", &n);   // se convierte la cadena contenida en b a un entero n
for(i=1; i<=n; i++) f*=i;          // se obtiene n! iterativamente empleando a f
                                   // para almacenar los resultados intermedios
printf("El factorial de n = %d es %d", n, f);
}
```

Programa-Práctica-Análisis

```
#include<stdio.h>
void main(void){
int i=1, n=5, f=1;
char b[10];
printf ("Dame el número del cual deseas el factorial ");
gets(b);
sscanf(b, "%i", &n);
while(i<=n) f*=i++;
printf("El factorial de n = %d es %d", n, f);
}
```

Programa-Práctica-Análisis

```
#include<stdio.h>
void main(void){
int i=1, n=5, f=1;
char b[10];
printf ("Dame el número del cual deseas el factorial ");
gets(b);
sscanf(b, "%i", &n);
```

do f*=i++; while(i<=n);

printf("El factorial de n = %d es %d", n, f);

}

```c
#include<stdio.h>
#include<math.h>

int main(void){
char b[10];
int i=0, imax=15, j;
float ea=100.0, es=5.0, xl, xu, xr, xra, fl, fr, test;
//ea=error relativo porcentual, es=error preestablecido
//xl=valor inferior, xu=valor superior
//xr=aprox. a la raiz, xra=aprox. a la raiz anterior
//fl=evaluacion de xl, fr=evalucion de xr
//test determina direccion de crecimiento
printf("Corriendo Biseccion .....\n\n");
printf("Introduce x minima \n");
gets(b);
sscanf(b, "%f", &xl);
printf("Introduce x maxima \n");
gets(b);
sscanf(b, "%f", &xu);
printf("x min : %f ..... x max : %f\n\n", xl,xu);
printf("i     xl       xu       xr     ea\n");

fl=pow(xl,10)-1.0; //(xl)**10-1;
xr=0.0;

do{
    xra=xr;
    xr=(xl+xu)/2.0;
    fr=pow(xr,10)-1.0;

    if(xr>0.001) ea=100.0*fabs((xr-xra)/xr);
        else ea=100.0;
    test=fl*fr;
    printf("%3d %f %f %f %f n", i,xl,xu,xr,ea);
    i++;
    if(test<0.0) xu=xr;
    else if(test>0.0){
        xl=xr;
        fl=fr;
    }else ea=0.0;
}while(ea>es && i<imax);
printf("\nLa raiz es : %f \n", xr);
printf("Presiona una tecla para finalizar ");
scanf("%d", &j);
return 0;
```

Fig. 4.5 Listado del programa para el algoritmo de Bisección.

```
Corriendo Biseccion .....

Introduce x minima
2
Introduce x maxima
3
x min : 2.000000 ..... x max : 3.000000

i     xl       xu       xr     ea
  0 2.000000 3.000000 2.500000 100.000000 n  1 2.500000 3.000000
 2.750000 9.090909 n  2 2.750000 3.000000 2.875000 4.347826 n
La raiz es : 2.875000
Presiona una tecla para finalizar 1

Process returned 0 (0x0)   execution time : 39.304 s
Press any key to continue.
```

Fig. 4.6 Corrida de prueba del algoritmo de Bisección.

La figura 4.5 es el listado para el algoritmo de Bisección el cual es parte de la Materia de Métodos Numéricos. El algoritmo fue implementado empleando la estructura de control DO-WHILE.

La figura 4.6 muestra la corrida de prueba para el algoritmo de Bisección.

5. Funciones

5.1. Estructura de la función [4, pp. 68-70].

Las funciones sirven para descomponer un programa en módulos de código pequeños, lo que simplifica la redacción y puesta a punto de los programas.

Una función:

- Puede estar en el mismo módulo que el programa principal (esto es las declaraciones y las instrucciones que la componen), inclusive automáticamente en el texto del programa empleando la orden **#include**, o compilarse por separado uniéndola al programa principal durante la fase enlace (*link*).
- Puede ser llamada por el programa principal, por otra función o por ella misma (recursividad).
- Llevar o no argumentos.
- Devolver o no un valor.
- Poseer sus propias variables, e incluso que algunas puedan guardar su valor para las sucesivas veces que se invoque la función.

También, una función:

- Tiene solo un punto de entrada, la primera instrucción de la función.
- Puede tener diversos puntos de salida, empleando la instrucción **return** o de manera automática después de la última instrucción de la función.

A diferencia de otros lenguajes de programación:

- El C no distingue entre procedimiento y función; un procedimiento no es más que una función de la que puede despreciarse el valor que devuelva.
- El C no permite el anidamiento de funciones: no pude declararse una función dentro de otra, pero, una función puede llamar a otra función.
- A una función sólo se le pueden pasar argumentos por valor.

El C realiza el intercambio de argumentos por referencia ya que el argumento que se pasa es la dirección de memoria donde esté el contenido de la variable, todo esto empleando el operador de cálculo de direcciones **&** (*ampersand*).

```c
#include<stdio.h>

void main(void){
int i,j; /* variabñes que pertenecen a main*/

i=2;
j=triple(i); /* llamada a la funcion triple */
printf("i*3=%d\n",j);
j=triple(4); /* otra llamada a funcion triple */
printf("4*3=%d\n",j);
}

int triple(int e){ /* Implementacion de la funcion
    return (3*e);  /* devuelve un entero */
}
```

```
i*3=6
4*3=12

Process returned 7 (0x7)
Press any key to continue.
```

Fig. 5.1 Listado para programa que emplea la función triple y salida de la corrida de prueba.

En el programa de la figura 5.1, i y j son variables del programa principal (más concretamente de la función **main**). Fuera de **main** son desconocidas. Cualquier referencia a i o a j hecha desde la función **triple** dará lugar a un error de compilación.

j = triple(i); /* esta línea es una instrucción */

- argumento de la función; cuando se usan varios argumentos deben separarse por comas; un argumento puede ser una expresión.
- nombre de la función
- el valor devuelto por la función se asigna a **j**

int triple(int e) /* esta línea es la declaración de una función */

- ¡no se pone punto y coma (;)!
- argumentos (si hay varios se separan por comas)
- nombre de la función
- tipo del valor devuelto (si no existe, el C supone que es un entero; se puede indicar que la función devuelve nada empleando el tipo **void**)

int triple(int e)

{

tipo y nombre de los argumentos

variables locales de la función

instrucciones de la función

}

5.2. Llamado o invocación de una función.

```
1   #include<stdio.h>
2   void main(void){
3   int i,j;
4   float x, mitad();/* mitad devuelve un float */
5
6   i=3, j;
7   x=mitad(i); /* llamada a mitad con el */
8               /* valor 3 como argumento */
9   printf("La mitad de 3 es %f \n",x);
10  x=mitad(i*7); /* valor del argumento : 21 */
11  printf("La mitad de i*7 es %f \n",x);
12  x=mitad(j=triple(i-1)); /* valor del argumento : 3*2 */
13  printf("El triple de i-1 es j=%d\n",j);
14  printf("La mitad de j es %f\n",x);
15  }
16
17  float mitad(int num){ /* declaracion de la funcion */
18      return(num/2.0); /* 2.0 forza division con reales */
19  }
20  int triple(int e){ /* Implementacion de la funcion */
21      return (3*e);  /* devuelve un entero */
22  }
23
```

```
La mitad de 3 es 1.500000
La mitad de i*7 es 10.500000
El triple de i-1 es 6
La mitad de j es 3.000000

Process returned 26 (0x1A)
Press any key to continue.
```

Fig. 5.2 Listado para programa que emplea las funciones mitad, triple y salida de la corrida de prueba (en la parte inferior).

En el programa de figura 5.2 se realizan varias llamadas a funciones, la primera es una llamada a la función mitad (en renglón 7) con argumento igual a 3 (**a los argumentos de las funciones también se le conoce como parámetros, en este caso el parámetro es i cuya magnitud es 3 y es un parámetro de entrada a la función**) que es el valor de i (en renglón 6). En renglón 10 nuevamente se hace una llamada a mitad ahora con argumento i*7 cuyo valor de retorno (10.5) es asignado a x. En el renglón 12 primero se llama a triple con argumento (i-1) y valor de retorno de magnitud 6 que a su vez es ahora el argumento de mitad, cuyo valor de retorno es un flotante de magnitud 3.0.

La secuencia de eventos descrita en el párrafo anterior es mostrada en la en la pantalla de salida de la corrida de prueba para este programa (en la parte inferior de la figura 5.2).

5.3. Uso de funciones con parámetros.

5.3.1. De entrada

En esta sección nos dedicaremos a transformar en funciones las secciones implementadas como parte del código de un programa para lograr un objetivo específico como es el caso del factorial.

El factorial para ser calculado requiere el número **n** del cual se desea el factorial, este número llega a ser el parámetro de entrada de la función. En este sencillo caso el parámetro de salida es el resultado del cálculo del factorial y sin problemas lo logramos por medio del **return** regresándolo como un entero (**int**).

La función factorial fue implementada empleando las tres diferentes estructuras de repetición con las que cuenta el lenguaje C (for, while y do while). Implementaremos la función factorial empleando la estructura for.

```
int factorial (int n){

int i; // el iterador

int f=1;  // variable que almacenará el resultado del factorial

for(i=1; i<=n; i++) f*=i;  // es lo mismo que f=f*i

return f;

}
```

```c
#include<stdio.h>

int factorial(int n){
int i;    // el iterador
int f=1; // variable que almacenará el
         // resultado del factorial
for(i=1; i<=n; i++) f*=i;

    return f;
}

void main(void){
char buf[10];
int n, f;

printf("Este programa calcula el factorial\n");
printf("de un entero n para obtener n!\n");

printf("Dame n \n");
gets(buf);
sscanf(buf,"%d",&n);

   f=factorial(n);

printf("El factorial de n=%d es %d \n",n,f);
}
```

```
Este programa calcula el factorial
de un entero n para obtener n!
Dame n
6
El factorial de n=6 es 720

Process returned 28 (0x1C)
Press any key to continue.
```

Fig. 5.3. Listado de programa que contiene la función factorial, la cual tiene en parámetro de salida, empleando el **return** para comunicar su resultado a **main**. En la parte inferior de la figura se tiene una corrida de prueba.

5.3.2. De salida

El **return** es la forma más simple por medio de la cual una función hace públicos sus resultados, para que sean empleados por otra u otras funciones incluida **main** que es la función principal. En la figura 5.3. se muestra un listado para la función factorial y esta regresa el resultado del cálculo empleando el **return**. En este caso el factorial es el único parámetro de retorno o regreso.

El C permite formas más complejas en el retorno de resultados, y entre estas se encuentran las estructuras (**struct**) y los arreglos, en las cuales es posible "empaquetar" gran cantidad de parámetros, tantos como sean necesarios. Es importante señalar que por medio de struct podemos empaquetar datos de diferentes tipos, esto es mezclar carácter, entero, flotante, arreglos, otra struct, etc. En el caso de los arreglos todos los datos son del mismo tipo.

5.4. Funciones externas

5.4.1. De usuario

Una forma de ser eficientes en la generación de programas para resolver algún problema en particular es analizar que método o estrategia emplearemos para darle solución de adecuada, invirtiendo la mínima cantidad de tiempo en buscar corregir algunos errores que se presenten en el proceso propio de la programación.

Haciendo eco al dicho "divide y vencerás", en programación no significa algo diferente a dividir en secciones pequeñas de código el método o estrategia seleccionado.

Mejor aún, en alguna área del conocimiento frecuentemente encontraremos problemas con características muy similares (tal vez las mismas) que en el pasado resolvimos. Se ahorra tiempo en el desarrollo si empleamos algunas secciones del código implementado en el pasado. Estas secciones de código no son otra cosa más que las funciones, con las que se ha trabajado.

A continuación se mostrarán los listados de programas realizados en Unidad 3 empleando funciones.

```
1   #include<stdio.h>
2
3   int cap_vec(int* A, int* B, int* c){
4   int i,j,d,*pc;  //un apuntador a c
5   char buf[10];   //bufer para conversion a numericos
6   pc=&c;  //asignamos la direccion de c a pc
7   printf("La longitud maxima de A y B es 10 \n");
8   printf("Dame la cantidad de componentes ");
9       gets(buf);                  // Captura una cadena
10      sscanf(buf,"%d",&c);// y la convierte a decimal
11  printf("Captura de vectores A y B\n");
12  for(i=0;i<*(pc);i++){
13      printf("A[%d]=",i);
14      gets(buf);                  // Captura una cadena
15      sscanf(buf,"%d",&d); A[i]=d;// y la convierte a
16  }                               // decimal
```

Fig. 5.4 Listado de programa para operaciones con Vectores empleando funciones (Parte 1).

```
17  for(i=0;i<*(pc);i++){
18      printf("B[%d]=",i);
19      gets(buf);                    // Captura una cadena
20      sscanf(buf,"%d",&d); B[i]=d;// y la convierte a
21  }                                  // decimal
22  for(i=0;i<*(pc);i++)
23      printf("A[%d]=%d\tB[%d]=%d\n",i,A[i],i,B[i]);
24  c=*(pc); // Aseguramos la magnitud de c
25  printf("cantidad de comp= %d\n",c);
26  getch();
27  return c;  // espera que el usuario continue
28  }
29
30  void suma_y_resta_vec(int* A,int* B,int r,int* S,int* R){
31  int i;
32  printf("La suma y resta\n");
33  for(i=0;i<r;i++){
34          S[i]=A[i]+B[i];
35          R[i]=A[i]-B[i];
36          printf("S[%d]=%d\t\tR[%d]=%d \n",i, S[i],i,R[i]);
37  }
38   getch(); // espera que el usuario continue
39  }
40
41  void producto_vec(int* A, int* B, int r, int* P){
42  int i;
43  printf ("El producto\n");
44  for(i=0;i<r;i++){
45      P[i]=A[i]*B[i];
46      printf("P[%d]=%d \n",i, P[i]);
47  }
48  getch(); // espera que el usuario continue
49  }
50
51  void main(void){
52  int A[10], B[10], C[10], S[10], R[10], P[10];
53  int c, r;
54  int i;
55
56  r=cap_vec(&A, &B, &c); //guarda valor de retorno r=c
57  printf("regreso c de r= %d \n",r);// se verifica
58  //veriiificamos la magnitud de componentes
59  //antes de continuar
60  for(i=0;i<r;i++) printf("A[%d]=%d\t\tB[%d]=%d\n",i,A[i],i,B[i]);
61  suma_y_resta_vec(&A, &B, r, &S, &R);
62  producto_vec(&A, &B, r, &P);
63  }
```

Fig. 5.4 Listado de programa para operaciones matemáticas con Vectores empleando funciones (Continuación Parte 2).

```
La longitud maxima de A y B es 10
Dame la cantidad de componentes 3
Captura de vectores A y B
A[0]=0
A[1]=1
A[2]=2
B[0]=2
B[1]=1
B[2]=0
A[0]=0  B[0]=2
A[1]=1  B[1]=1
A[2]=2  B[2]=0
cantidad de comp= 3
regreso c de r= 3
A[0]=0          B[0]=2
A[1]=1          B[1]=1
A[2]=2          B[2]=0
La suma y resta
S[0]=2          R[0]=-2
S[1]=2          R[1]=0
S[2]=2          R[2]=2
El producto
P[0]=0
P[1]=1
P[2]=0

Process returned 32 (0x20)   execution time : 73.429 s
Press any key to continue.
```

Fig. 5.5 Corrida de prueba para programa operaciones matemáticas con Vectores empleando funciones.

```
1   #include<stdio.h>
2   #include<string.h>
3
4   const int ren=3, col=3; //Dimensiones
5   int A[3][3],B[3][3],C[3][3],S[3][3],R[3][3],*pm;
6   int rg[3], co[3];
7
8   int captura(char nm){
9   int i,j,d;
10  char b[10];
11
12  printf("\nCapturando Matriz : %c \n\n",nm);
```

Fig. 5.6 Listado de programa para operaciones matemáticas con arreglos Bidimensionales empleando funciones (Parte 1).

```
13  for(i=0;i<ren;i++){
14      printf("\n");
15      for(j=0;j<col;j++)
16      {
17      printf("E%d,%d = ",i, j);
18      gets(b);
19      sscanf(b,"%d",&d);
20      *(pm+i*col+j)=d;
21  }
22  }
23  return 0;
24  }
25
26  int suma(void){
27  int i,j;
28  int *pma, *pmb, *pmc;
29  pma=&A[0][0];
30  pmb=&B[0][0];
31  pmc=&C[0][0];
32
33  for(i=0;i<ren;i++)
34      for(j=0;j<col;j++)
35          *(pmc+i*col+j)= *(pma+i*col+j)+*(pmb+i*col+j);
36      return 0;
37  }
38
39  int resta(void){
40  int i,j;
41  int *pma, *pmb, *pmc;
42  pma=&A[0][0];
43  pmb=&B[0][0];
44  pmc=&C[0][0];
45
46  for(i=0;i<ren;i++)
47      for(j=0;j<col;j++)
48          *(pmc+i*col+j)= *(pma+i*col+j)-*(pmb+i*col+j);
49      return 0;
50  }
51
52  void renglon(int r){
53  int j;
54  for(j=0;j<col;j++) rg[j]=A[r][j];
55  }
56
57  void columna(int c){
58  int i;
59  for(i=0;i<ren;i++) co[i]=B[i][c];
60  }
61
```

Fig. 5.6 Listado de programa para operaciones matemáticas con arreglos Bidimensionales empleando funciones (Continuación Parte 2).

```
62  int producto(void){
63  int i,j, m, s;
64  for(i=0;i<ren;i++){
65      renglon(i);
66      for(j=0;j<col;j++){
67          s=0;
68          columna(j);
69          for(m=0;m<ren;m++)
70              s+=rg[m]*co[m];
71              C[i][j]=s;
72          }
73  }
74  }
75
76  int muestra(char *o){
77  int i, j, e, *pmc;
78  pmc=&C[0][0];
79
80  printf("\nMostrando Resultados de : %s\n\n",o);
81
82  for(i=0;i<ren;i++){
83      for(j=0;j<col;j++){
84              e=*(pmc+i*col+j);
85              printf("%d\t",e);
86      }
87      printf("\n");
88  }
89  return 0;
90  }
91
92  int main (void){
93  char adios[10];
94  char *opera;
95  int k;
96
97  pm=&A[0][0];
98  captura('A');
99
100 pm=&B[0][0];
101 captura('B');
102
103 suma();
104 muestra("SUMA");
105
106 resta();
107 muestra("RESTA");
108
```

Fig. 5.6 Listado de programa para operaciones matemáticas con arreglos Bidimensionales empleando funciones (Continuación Parte 3).

```
109    producto();
110    muestra("PRODUCTO");
111    printf("\nPresiona una tecla para finalizar \n");
112    gets(adios);
113    sscanf(adios,"%d",&k);
114    return 0;
115    }
```

Fig. 5.6 Listado de programa para operaciones matemáticas con arreglos Bidimensionales empleando funciones (Continuación Parte 4).

```
Capturando Matriz : A   Capturando Matriz : B


E0,0 = 1                E0,0 = 3
E0,1 = 2                E0,1 = 3
E0,2 = 3                E0,2 = 1

E1,0 = 3                E1,0 = 1
E1,1 = 2                E1,1 = 3
E1,2 = 1                E1,2 = 2

E2,0 = 1                E2,0 = 2
E2,1 = 3                E2,1 = 3
E2,2 = 2                E2,2 = 1
Mostrando Resultados de : SUMA

4       5       4
4       5       3
3       6       3

Mostrando Resultados de : RESTA

-2      -1      2
2       -1      -1
-1      0       1

Mostrando Resultados de : PRODUCTO

11      18      8
13      18      8
10      18      9
```

Fig. 5.7 Corrida de prueba para operaciones matemáticas con arreglos Bidimensionales empleando funciones. (Editado por cuestiones de espacio)

```c
#include<stdio.h>
#include<string.h>

const int ren=3, col=3; //Dimensiones
int X[3][3], T[3][3][3];
int rg[3], co[3];
int sP[3];

int capturaM3D(void){
int i,j,k,d;
char buf[10];

for (k=0;k<3;k++){
for(i=0; i<3; i++){
    for(j=0;j<3;j++){
    printf("Dame elemento %d, %d de :T%d ",i,j,k);
    gets(buf);                  // Captura una cadena
    sscanf(buf,"%d",&d); // Convierte la cadena a decimal y la asigna a T[i][j][k]
    T[i][j][k]=d;
    }
}
for(i=0;i<3;i++){
        for(j=0;j<3;j++) printf("%d\t", T[i][j][k]);
        printf("\n");
    }
} return 0;
}

int seleccion(void){
char o, q, buf[10];
int p0,p1;
int *pvs;
int a,b;
pvs=&sP[0];
printf("Selecciona 2 pag. entre las que se realizaran las op. matematicas\n");
printf("0 op(0,1) \n");
printf("1 op(0,2) \n");
printf("2 op(1,2) \n");
gets(buf);
sscanf(buf,"%d",&o);
switch (o){
    case 0: printf("\nSeleccionaste 0 op(0,1) \n"); p0=0; p1=1; break;
    case 1: printf("\nSeleccionaste 1 op(0,2) \n"); p0=0; p1=2; break;
    case 2: printf("\nSeleccionaste 2 op(1,2) \n"); p0=1; p1=2; break;
    default:o=0; printf("\nEl default es 0 op(0,1) \n"); p0=0; p1=1;
}
*(pvs)=p0; *(pvs+1)=p1;
a=*(pvs);
b=*(pvs+1);
return 0;
}

int suma3D(void){
int *pvs, p0, p1, i, j;
```

Fig. 5.8 Listado de programa para operaciones matemáticas con arreglos Tridimensionales empleando funciones (Parte 1).

```
55  pvs=&sP[0];
56
57  p0=*(pvs);
58  p1=*(pvs+1);
59
60  printf("La suma de T[i][j][%d] + T[i][j][%d]\n", p0, p1);
61            // La suma y resta matricial es elemento a elemento
62  for(i=0;i<3;i++){
63      for(j=0;j<3;j++) printf("%d\t",T[i][j][p0]+T[i][j][p1]);
64      printf("\n");
65  } return 0;
66  }
67
68  int resta3D(void){
69  int *pvs, p0, p1, i, j;
70  pvs=&sP[0];
71
72  p0=*(pvs);
73  p1=*(pvs+1);
74
75  printf("La resta de T[i][j][%d] - T[i][j][%d]\n", p0, p1);
76            // La suma y resta matricial es elemento a elemento
77  for(i=0;i<3;i++){
78      for(j=0;j<3;j++) printf("%d\t",T[i][j][p0]-T[i][j][p1]);
79      printf("\n");
80  }  return 0;
81  }
82
83  int producto3D(void){
84  int *pvs, p0, p1, c, s, r, m, i, j;
85  pvs=&sP[0];
86
87  p0=*(pvs);
88  p1=*(pvs+1);
89
90  printf("El producto matricial de T[i][j][%d] * T[i][j][%d]\n", p0, p1);
91  // El producto matricial es la suma de los productos entre fila y
92  for(i=0; i<3; i++){                                    // columna
93      for(c=0; c<3; c++)rg[c]=T[i][c][p0];       // Obtine la fila
94          for(j=0; j<3; j++){
95              s=0;                                // De inicio la suma es CERO, s=0
96              for(r=0; r<3; r++) co[r]=T[r][j][p1];  //Obtiene la columna
97                  for(m=0; m<3; m++) s=s+rg[m]*co[m]; // Hace la suma de Prod
98                      X[i][j]=s; // La suma de productos se asigna a X
99          }// rg y co son vectores auxiliares, X es una matriz auxiliar
100 }
101     for(i=0;i<3;i++){
102         for(j=0;j<3;j++) printf("%d\t", X[i][j]);
103         printf("\n");
104     }
105 }
106
107 int main (void){
108 char adios[10];
```

Fig. 5.8 Listado de programa para operaciones matemáticas con arreglos Tridimensionales empleando funciones (Continuación Parte 2).

```
109    int k;
110    capturaM3D();
111    seleccion();
112    suma3D();
113    resta3D();
114    producto3D();
115    printf("\nPresiona una tecla para finalizar \n");
116    gets(adios);
117    sscanf(adios,"%d",&k);
118    return 0;
119    }
120
```

Fig. 5.8 Listado de programa para operaciones matemáticas con arreglos Tridimensionales empleando funciones (Continuación Parte 3).

```
Dame elemento 0, 0 de :T0 1  Dame elemento 0, 0 de :T1 3  Dame elemento 0, 0 de :T2 1
Dame elemento 0, 1 de :T0 1  Dame elemento 0, 1 de :T1 3  Dame elemento 0, 1 de :T2 1
Dame elemento 0, 2 de :T0 1  Dame elemento 0, 2 de :T1 3  Dame elemento 0, 2 de :T2 1
Dame elemento 1, 0 de :T0 2  Dame elemento 1, 0 de :T1 2  Dame elemento 1, 0 de :T2 3
Dame elemento 1, 1 de :T0 2  Dame elemento 1, 1 de :T1 2  Dame elemento 1, 1 de :T2 3
Dame elemento 1, 2 de :T0 2  Dame elemento 1, 2 de :T1 2  Dame elemento 1, 2 de :T2 3
Dame elemento 2, 0 de :T0 3  Dame elemento 2, 0 de :T1 1  Dame elemento 2, 0 de :T2 2
Dame elemento 2, 1 de :T0 3  Dame elemento 2, 1 de :T1 1  Dame elemento 2, 1 de :T2 2
Dame elemento 2, 2 de :T0 3  Dame elemento 2, 2 de :T1 1  Dame elemento 2, 2 de :T2 2
1       1       1            3       3       3            1       1       1
2       2       2            2       2       2            3       3       3
3       3       3            1       1       1            2       2       2

Selecciona 2 pag. entre las que se realizaran las op. matematicas
0 op(0,1)
1 op(0,2)
2 op(1,2)
0

Seleccionaste 0 op(0,1)
La suma de T[i][j][0] + T[i][j][1]   La resta de T[i][j][0] - T[i][j][1]
4       4       4                    -2      -2      -2
4       4       4                    0       0       0
4       4       4                    2       2       2

El producto matricial de T[i][j][0] * T[i][j][1]
6       6       6
12      12      12
18      18      18
```

Fig. 5.9 Corrida de prueba para operaciones matemáticas con arreglos Tridimensionales empleando funciones. (Editado por cuestiones de espacio)

En la programación existen una gran cantidad de funciones que aparecen de manera frecuente casi en cualquier programa que se realice, en alguna de las áreas del conocimiento. Rápidamente nos daremos cuenta de que sería muy útil para nosotros

tener una colección de funciones que minimizaran el tiempo de desarrollo de alguna aplicación. Lo primero en lo que pensaríamos es tener una librería de la cual sea posible hacer uso de las funciones que contenga, y al paso del tiempo que esa librería se mejore continuamente con nuevas funciones o funciones mejoradas por nosotros.

Pareciera una tarea complicada, sin embargo no lo es. Todo es cuestión de voluntad, paciencia y actitud en mantener funcional nuestra idea, nuestra librería y mejorarla en la medida de lo posible, haciéndola crecer con nuevas funciones que nos dicte la propia experiencia.

Hagamos el intento de crear una librería muy sencilla.

En este caso crearemos una librería con tres funciones:

a) Simpson 3/8 útil en la solución de integrales.
b) Obtención del factorial.
c) Un simple producto.

A la librería no le debe faltar información, por lo que es necesario:

a) Crear un archivo de encabezados (archivo.h) conteniendo los encabezados de cada una de nuestras funciones componentes de nuestra librería (en este caso 3).
b) Crear un archivo (archivo.c) donde declararemos cada una de nuestras funciones.
c) Indicarle a Code::Blocks que tipo de librería requerimos (queremos una librería estática), nombre (en este, MI caso, Mi_Lib) y su ubicación.

El paso siguiente es obtener la librería. Iniciemos por el archivo.h (Mi_Lib.h)

```
1   #ifndef MI_LIB_H_INCLUDED
2   #define MI_LIB_H_INCLUDED
3
4   #endif // MI_LIB_H_INCLUDED
5   #include<stdio.h>
6
7   float Simpson38(float li, float ls,float f0,float f1, float f2, float f3);
8   int factorial(int n);
9   int triple(int a);
10
```

Fig. 5.10 Código del archivo de encabezados para Mi_Lib

Para este archivo el compilador inserta el texto de las líneas 1,2 y 4 (conocidas como órdenes del preprocesador y las identificamos fácilmente cuando encontramos #).

Requerimos funciones de entrada y salida que son proporcionadas por <stdio.h> agregando la línea 5 (nosotros agregamos esta orden al preprocesador por eso el #).

Cada una de las funciones que forman la librería y sus prototipos, en líneas 7,8 y 9.

Continuamos con el archivo.c (en este caso Mi_Lib.c)

```
1   float Simpson38(float li, float ls,float f0,float f1, float f2, float f3)
2   {
3   float I=(ls-li)*(f0+3*f1+3*f2+f3)/8.0;
4   return I;
5   }
6
7   int factorial(int n){
8   int i;    // el iterador
9   int f=1; // variable que almacenará el
10            // resultado del factorial
11  for(i=1; i<=n; i++) f*=i;
12
13      return f;
14  }
15
16  int triple(int a){
17      return a*3;
18  }
19
```

Fig. 5.11 El archivo Mi_Lib.c mostrando la declaración de sus funciones.

Code::Bloks nos ayuda en la creación de la librería por medio de un Proyecto. En la pantalla de inicio nos muestra diversas opciones. Para este caso el proyecto que requerimos es para generar una **Librería Estática**, seleccione el icono y Code::Blocks lo llevará paso a paso.

En este punto la librería ya está lista para ser empleada en alguna aplicación. En este caso la aplicación que generé para probar Mi_Lib la llame Prueba_Mi_Lib.c. En la figura 5.12 se muestra el código de prueba para Prueba_Mi_lib.c, mientras que en la figura 5.13 se muestra la corrida de prueba para Prueba_Mi_Lib. El proyecto generado para esta aplicación también tiene el nombre de Prueba_Mi_Lib.

```c
#include <stdio.h>
#include <stdlib.h>
#include "C:\\\\mi_lib\Mi_Lib.H"

int main(void)
{
int num, n, f;
float I;
char buf[10];

    printf("Probemos FACTORIAL\n");
    printf("Dame un entero del cual deseas el Factorial n!\n");
    gets(buf);
    sscanf(buf, "%d", &num);
    f=factorial(num);
    printf("\nEl factorial de %d es %d\n",num, f);

    printf("Probemos a Simpson 3 octavos\n");
    I=Simpson38(0.0,2.0,0,2.0/3.0,4.0/3.0,2.0);
    printf("El resultado de la integral es %f",I);
return 0;
}
```

Fig. 5.12 Código de Prueba_Mi_Lib.c

```
Probemos FACTORIAL
Dame un entero del cual deseas el Factorial n!
6

El factorial de 6 es 720
Probemos a Simpson 3 octavos
El resultado de la integral es 2.000000
Process returned 0 (0x0)   execution time : 3.930 s
Press any key to continue.
```

Fig. 5.13 Corrida de prueba para Prueba_Mi_Lib.c

5.4.2. De bibliotecas.

El estándar ANSI/ISO para C define el contenido y la forma de la librería estándar para C. El estándar especifica un conjunto de funciones que todos los compiladores deben soportar.

A continuación la tabla de encabezados definida y soportada por el estándar C89.

Tabla 5.1 Tabla de encabezados definida y soportada por el estándar C89 [3, p. 306].

Encabezado	**Propósito**
<assert.h>	Define el macro **assert()**
<ctype.h>	Manejo de caracteres

<errno.h>	Reporte de errores
<float.h>	Define la implementación -de límites de punto flotante
<limits>	Define varios límites dependientes de implementación
<locale.h>	Soporta localización
<math.h>	Varias definiciones empleadas por la librería matemática
<setjmp.h>	Soporta saltos no locales
<signal.h>	Soporta manejo de señales
<stdarg.h>	Soporta listas con argumentos variables
<stddef.h>	Define algunas constantes comúnmente usadas
<stdio.h>	Soporta el sistema de I/O
<stdlib.h>	Declaraciones misceláneas
<string.h>	Soporta manejo de cadenas
<time.h>	Soporta el sistema de funciones de tiempo

Los archivos de encabezados agregados por C99 son mostrados en la tabla siguiente.

Tabla 5.2 Tabla de encabezados agregados por C99 [3, p. 307].

Encabezado	**Propósito**
<complex.h>	Soporta aritmética compleja
<fenv.h>	Da acceso a las banderas de estado y otros aspectos del ambiente de punto flotante
<inttypes.h>	Define un estándar para nombres de un conjunto de tipos enteros
<iso646.h>	Define macros que corresponden a varios operadores tales como && y ^
<stdbool.h>	Soporta tipos de datos booleanos, define el macro bool que ayuda en la compatibilidad con C++
<stdint.h>	Define un estándar para nombres de un conjunto de tipos enteros. Este archivo es incluido por <inttypes.h>
<tgmath.h>	Define macros de tipo genérico para tipo flotante
<wchar.h>	Soporta funciones multibyte y carácter ancho
<wctype.h>	Soporta funciones de clasificación de multibyte y carácter ancho

Las dos tablas mostradas nos dan el listado completo de encabezados de C y sus propósitos.

Conceptos tales como manejo de archivos y cadenas (strings) son importantes en la actividad profesional. Las máquinas o equipos nos entregan regularmente secuencias de datos que algunas veces se requiere decodificar con el objetivo de conocer cierta información que nos interesa y procesarla de alguna manera.

Almacenar dicha información o recuperarla de un archivo, también llega a ser una tarea cotidiana.

A continuación un ejemplo en la figura 5.14 tomado de [3, p. 311], donde se manejan archivos realizando escrituras y lecturas binarias.

```c
/* Copia un archivo a otro */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
FILE *in, *out;
char ch;
if(argc!=3) { //argc es el conteo de argumentos
         // que se encuentran en argv[]
         // argv[0] siempre es el nombre del
         // programa que se esta ejecutando.
printf("Olvidaste el nombre de un archivo.\n");
exit(1);
}
if((in=fopen(argv[1], "rb")) == NULL) {// "rb" lectura binaria
printf("No se puede abrir el archivo de entrada.\n");
exit:(1);
}
if((out=fopen(argv[2], "wb")) == NULL) {// "wb" escritura binaria
printf("No se puede abrir el archivo de salida.\n");
exit(1);
}

while(!feof(in)) {//mientras no lleguemos al final del archivo
ch = getc(in);  // lee un caracter
if(ferror(in)) {// si hay error en el archivo de entrada
printf("Read Error");
clearerr(in); //borra el error en el archivo de entrada
break;
} else {
if(!feof(in)) putc(ch, out);//ssi no hay error en archivo
    //de entrada escribe el caracter en archivo de salida
if(ferror(out)) {// si hay error en archivo de salida
printf("Write Error");
clearerr(out);//borra el error en el archivo de salida
break;
}
}
}
fclose(in); //cierra archivo de entrada
fclose(out);//cierra archivo de salida
return 0;
}
```

Fig. 5.14 Listado de programa que copia un archivo en otro, la lectura y escritura son binarias.

```c
#include <stdio.h>
/* Archivos y caracteres.
El programa escribe caracteres en un archivo. */
void main(void)
{
char p1;
FILE *ar;
ar = fopen("arc.txt","w"); /* Se abre el archivo arc.txt para escritura. */
if (ar != NULL)
{
while ((p1=getchar()) != "\n")
/* Se escriben caracteres en el archivo mientras no se detecte el caracter
que indica el fin de la línea. */
fputc(p1, ar);
fclose(ar); /* Se cierra el archivo. */
}
else
printf("No se puede abrir el archivo");
}
```

Fig. 5.15 Listado de programa que escribe caracteres en archivo. Ejemplo tomado de [5, p. 337].

```c
#include <stdio.h>
/* Archivos y caracteres.
El programa lee caracteres de un archivo. */
void main(void)
{
char p1;
FILE *ar;
if((ar=fopen("arc.txt","r"))!=NULL) /* Se abre el archivo para lectura */
/* Observa que las dos instrucciones del programa 9.1 necesarias para abrir un
archivo y verificar que éste en realidad se haya abierto, se pueden agrupar
en una sola instrucción. */
{
while (!feof(ar))
/* Se leen caracteres del archivo mientras no se detecte el fin del
➥archivo. */
{
p1 = fgetc(ar); /* Lee el caracter del archivo. */
putchar(p1); /* Despliega el caracter en la pantalla. */
}
fclose(ar);
}
else
printf("No se puede abrir el archivo");
}
```

Fig. 5.16 Listado de programa que lee caracteres de un archivo. Ejemplo tomado de [5, p. 337].

En las figuras 5.15 y 5.16 son ejemplos tomados de [5], en los que se muestra la escritura y lectura de archivos de texto.

Tabla 5.3 Funciones del archivo de encabezados <string.h>

Función	Propósito
char *strcpy(s,ct)	Copia la cadena ct a la cadena s incluyendo '\0', retorna el apuntador al inicio de la cadena s.
char *strncpy(s,ct,n)	Copia hasta n caracteres de la cadena ct a la cadena s rellena con '\0' si ct tiene menos de n caracteres, retorna el apuntador al inicio de la cadena s.
char *strcat(s,ct)	Concatena la cadena ct al final de la cadena s, retorna el apuntador al inicio de la cadena s.
char *strncat(s,ct,n)	Concatena hasta n caracteres de la cadena ct al final de la cadena s incluyendo '\0', retorna el apuntador al inicio de la cadena s.
int *strcmp(cs,ct)	Compara la cadena cs con la cadena ct, retorna <0 si cs<ct, 0 si cs==ct y >0 si cs>ct.
int *strncmp(s,ct,n)	Compara hasta n caracteres de la cadena cs con la cadena ct, retorna <0 si cs<ct, 0 si cs=ct Y >0 si cs>ct.
char *strchr(cs,c)	Retorna un apuntador a la primera ocurrencia de c en cs o el NULO si c no está presente.
char *strrchr(sc,c)	Retorna un apuntador a la última ocurrencia de c en cs o el NULO si c no está presente.
size_t *strspn(cs,ct)	Retorna la longitud del prefijo de cs que contiene los caracteres de ct.
size_t *strcspn(cs,ct)	Retorna la longitud del prefijo de cs que consiste en los caracteres que no están ct.
char *strpbrk(cs,ct)	Retorna un apuntador a la primera ocurrencia en la cadena cs de cualquier carácter de la cadena ct o el NULO sí ninguno está presente.
char *strstr(cs,ct)	Retorna un apuntador a la primera ocurrencia en la cadena ct en la cadena cs o el NULO si no está presente.
size_t *strlen(cs)	Retorna la longitud de la cadena cs.
char *strerror(n)	Retorna un apuntador a una cadena definida por la implantación, correspondiente al error n.
char *strtok(s,ct)	Busca en s tokens delimitados por caracteres de ct.

La tabla 5.3 es un listado de funciones y propósitos de las funciones componentes en el archivo de encabezados string.h [6, p. 128]. Mientras que en la figura 5.17 se tiene el listado de un programa [6, p. 129], donde se ejemplifica el uso de algunas de las funciones componentes del archivo de encabezados string.h.

Un detalle importante es el hecho de que en este listado se hace uso de la instrucción GOTO, no se modificó, es trabajo de los alumnos hacer las adecuaciones necesarias para sustituir esta instrucción.

```c
#include <stdio.h>
#include <string.h>

#define L1 "RUTINAS DE PRUEBA DE LAS FUNCIONES DE MANEJO DE CADENAS"
#define L2 "\t\t 5upongase que S1 y S2 son cadenas\n\n"
#define L3 "\tstrcat"
#define l3 "\tagrega una copia de S2 al final de S1 \n"
#define L4 "\tstrncat"
#define l4 "\tagrega n caracteres de S2 a S1 \n"
#define L5 "\tstrcmp"
#define l5 "\tcompara S2 y S1 \n"
#define L6 "\tstrncmp"
#define l6 "\tcompara n caracteres de S2 con S1 \n"
#define L10 "n\tswab"
#define l10 "\tintercambia de posicion los caracteres de S2 cada 2 y los pasa a S1\n"
#define L7 "\tstrcpy"
#define l7 "\tcopia S2 en S1 \n"
#define L8 "\tstrncpy"
#define l8 "\tcopia n caraeteres de S2 en S1 \n"
#define L9 "\tstrlen"
#define l9 "\tda la longitud de S2\n"
/* Rutina de espera para lectura del instructivo en pantalla */
#define letrsper printf("Para continuar oprime S");
#define espera letrsper while (getchar()!= 'S')
#define pagina printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n")
int comando,N,RCOMP;
char S[1], S1[40], S2[10];
comparacion(int com)
{
if (com == 3){ printf("No. de caracteres a comparar\n");
scanf("%d",&N); RCOMP = strncmp(S1,S2,N);}
else RCOMP = strcmp(S1,S2);
pagina; printf("RE5ULTADO DE LA COMPARACION\n");
if (RCOMP ==0)printf("\n\ncadenas iguales\n");
else printf("\n\ncadenas diferentes\n");
printf("Resultado de la comparacion RCOMP = %d\n\n\n\n",RCOMP);
getchar(); espera;
}
main()
{
pagina; printf("%s%s%s%s%s%s%s%s%s%s",L1,L2,L3,l3,L4,l4,L5,l5,L6,l6);
printf("%s%s%s%s%s%s%s%s\n\n\n\n\n\n\n",L10,l10,L7,l7,L8,l8,L9,l9);
espera; pagina;
CICLO:
printf("INDlCA TU ELECCION CON EL NUMERO A LA DERECHA DE LA INST Y<cr>\n");
```

Fig. 5.17 Listado de programa que hace uso de algunas de las funciones componentes del archivo de encabezados string.h (Parte 1).

```
46  printf("\n \n%s\t0\n%s\t1 \n%s\t2\n%s\t3\n%s\t4\n" ,L3,L4, L5 ,L6 ,L10);
47  printf("l%s\t5\n%s\t6\n%s\t7\n\n\n\n\n\n\n\n\n",L7,L8,L9);
48  scanf("%d",&comando); pagina;
49  if (comando < 4) {printf("CADENA S1\n"); scanf("%s",S1);}
50  if (comando < 7) {printf("CADENA S2\n"); scanf("%s",S2);}
51  switch (comando)
52  {
53  case 0:{ strcat(S1,S2); break;}
54  case 1:{
55  printf("No. de caracteres a agregar\n"); scanf("%d",&N);
56  strncat(S1,S2,N); break;}
57  case 2:
58  case 3:{comparacion(comando); break;}
59  case 4:{ printf("No de bytes a \n"); scanf("%d",&N);
60  swab(S2,S1,N); break;}
61  case 5:{strcpy(S1,S2); break;}
62  case 6:{
63  printf("No. de caracteres a copiar\n"); scanf("%d",&N);
64  strncpy(S1,S2,N); break;}
65  case 7:{printf("CADENA\n"); scanf("%s",S1);
66  printf("La longitud es de %d caracteres\n\n\n",strlen(S1));
67  getchar(); espera;}
68  }
69  pagina; printf("ESTADO FINAL DE LAS CADENAS\n");
70  printf("\n\nCADENA S1 \n%s\n\n\nCADENA S2\n%s\n",S1,S2);
71  printf("\nContinuamos?\n"); scanf("%s",S);
72  if ((S[0] =='s')|| (S[0] == 'S') )goto CICLO;
73  }
```

Fig. 5.17 Listado de programa que hace uso de algunas de las funciones componentes del archivo de encabezados string.h (Continuación Parte 2).

La Tabla 3 y el programa de manejo de cadenas fueron tomados de [6].

6. Uso de puertos de comunicación

6.1. Tipos de puertos de comunicación.

Básicamente los puertos de comunicación son clasificados en dos grandes grupos de acuerdo con la forma en que la información es transportada o transmitida, estos son:

a) Transmisión en serie
b) Transmisión en paralelo

La transmisión en serie consiste en el envío de secuencias de bits enviados o recibidos uno a uno de unos y ceros.

La transmisión en paralelo permite el envío o recepción de más de un bit a la vez.

Cada tipo de transmisión tiene sus ventajas e inconvenientes. Lo sobresaliente de la transmisión en serie respecto a la transmisión en paralelo es que:

- Es más barata, puesto que requiere menos conductores.
- No tiene tantos problemas de autoinducción en sus líneas, pues requiere menos conductores.

Estas características hacen más adecuada la transmisión en serie para distancias de algunos metros (15m máximo), donde el costo del cableado sería un factor para considerar.

Las transmisiones en paralelo son las más adecuadas para alcanzar altas razones de transferencia a nivel de circuitos impresos o circuitos integrados.

Sin embargo, los avances tecnológicos en electrónica cada vez hacen menores las ventajas entre uno y otro grupo en la transmisión de información.

6.2. Especificaciones de los puertos de comunicación.

RS232 [7]

La fig. 6.1 muestra los conectores macho y hembra tipo DB-25, mientras que la tabla 6.1 muestra un listado con número de pin del conector macho o hembra tipo DB-25 así como el nombre de señal asignada a dicho pin.

Fig. 6.1 Conectores macho y hembra tipo DB-25

Tabla 6.1 Nombre de señal asignados a los pines del conector DB-25

Pin	Señal
1	PGND (Protective Ground)
2	TXD (Transmit Data)
3	RXD (Receive Data)
4	RTS (Ready to Send)
5	CTS (Clear to Send)
6	DSR (Data Set Ready)
7	SG (Signal Ground)
8	CD (Carrier Detect)
20	DTR (Data Terminal Ready)
22	RI (Ring Indicator)

La fig. 6.2 muestra los conectores macho y hembra tipo DB-9, mientras que la tabla 6.2 muestra un listado con número de pin del conector macho o hembra tipo DB-9 así como el nombre de señal asignada a dicho pin.

Fig. 6.2 Conectores macho y hembra tipo DB-9

Tabla 6.2 Nombre de señal asignados a los pines del conector DB-9

Pin	Señal
1	CD (Data Carrier Detect)
2	RXD (Recive Data)
3	TXD (Trasmit Data)
4	DTR (Data Terminal Ready)
5	SG (Signal Ground)
6	DSR (Data Set Ready)
7	RTS (Request To Send)
8	CTS (Clear To Send)
9	RI (Ring Indicator)

Frecuentemente no son empleadas todas las señales , y en el caso del conector DB-9 es común que solo los pines 2, 3 y 5 sean empleados (RXD, TXD y SG de un lado y del otro TXD, RXD y SG), donde claramente se deberá apreciar el cruce de líneas, como lo muestra la figura 6.3. A esta configuración en la interconexión se le conoce como Null-Modem.

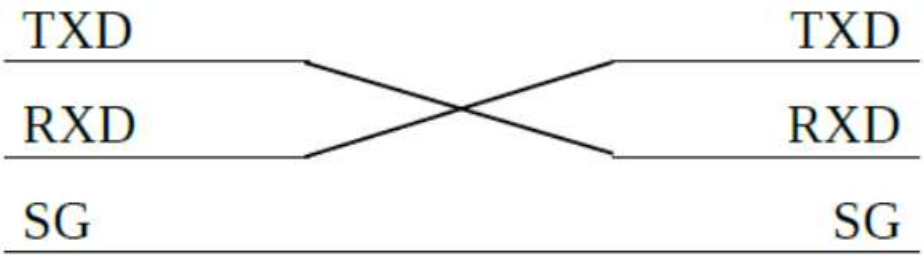


Fig. 6.3 Conexión Null-Modem.

Un aspecto importante es que la comunicación empleando RS232 es punto a punto, no es posible una red.

Por otro lado la razón de transferencia máxima es marcada en 20Kb/s.

USB [8]

El estándar USB soporta:

- Topología en estrella escalonada (ver fig. 6.4).
- Múltiples hubs definen hasta 5 niveles (ver fig. 6.4 y fig. 6.5).
- Cada host provee controladores de host y c/u admite hasta 127 dispositivos.
- Host oficia de máster (controla las transferencias).
- Soporta diferentes razones de transferencia (ver fig. 6.6)

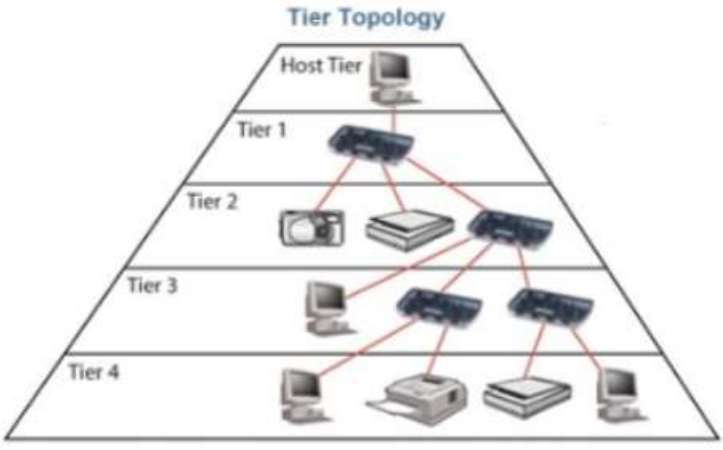


Fig. 6.4 Topología en estrella escalonada.

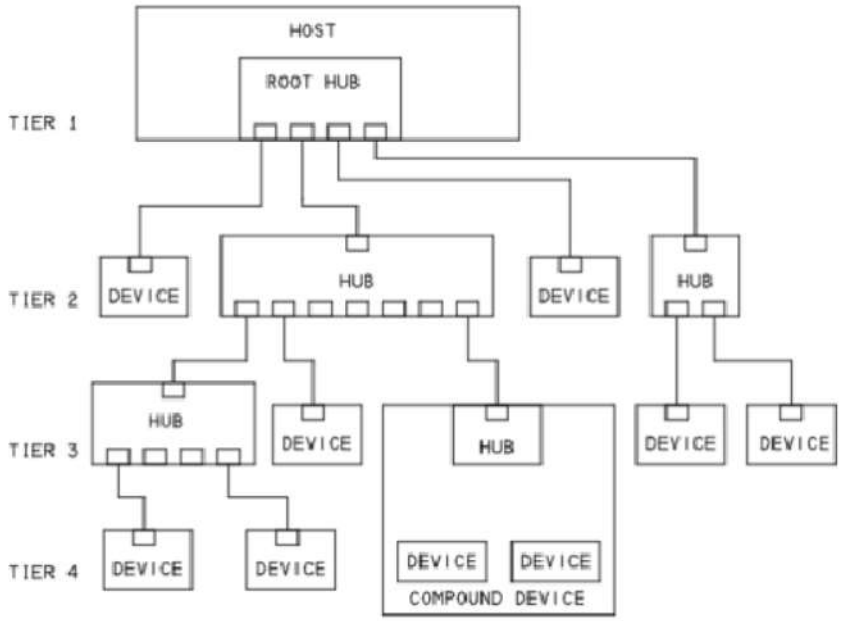


Fig. 6.5 Topología en estrella escalonada. Cada hub se comunica con el hub superior y con uno o más puertos.

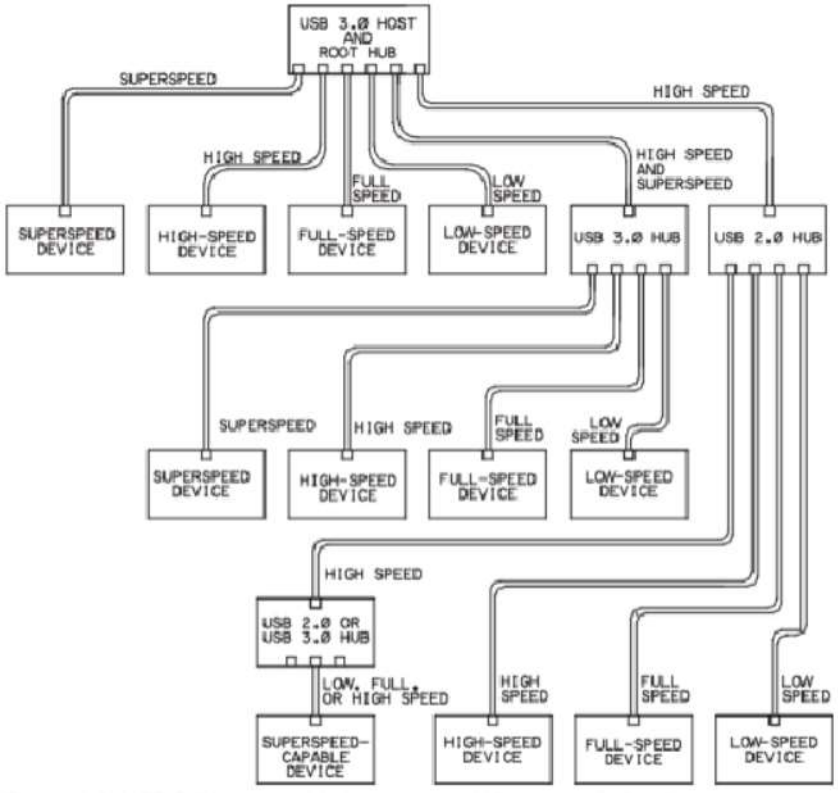


Fig. 6.6 En USB 3.0 los host y hubs soportan 4 razones de transferencia baja. (USB 1.1, 2.0 y 3.0).

Los dispositivos se componen de subdispositivos lógicos (uso de descriptores)

- Implementan una o varias configuraciones (ver fig. 6.7)
- Implementan una o varias funciones (interfaz) (ver fig. 6.7)
- Dirección propia y comunicación con endpoints vía pipes (ver fig. 6.8).

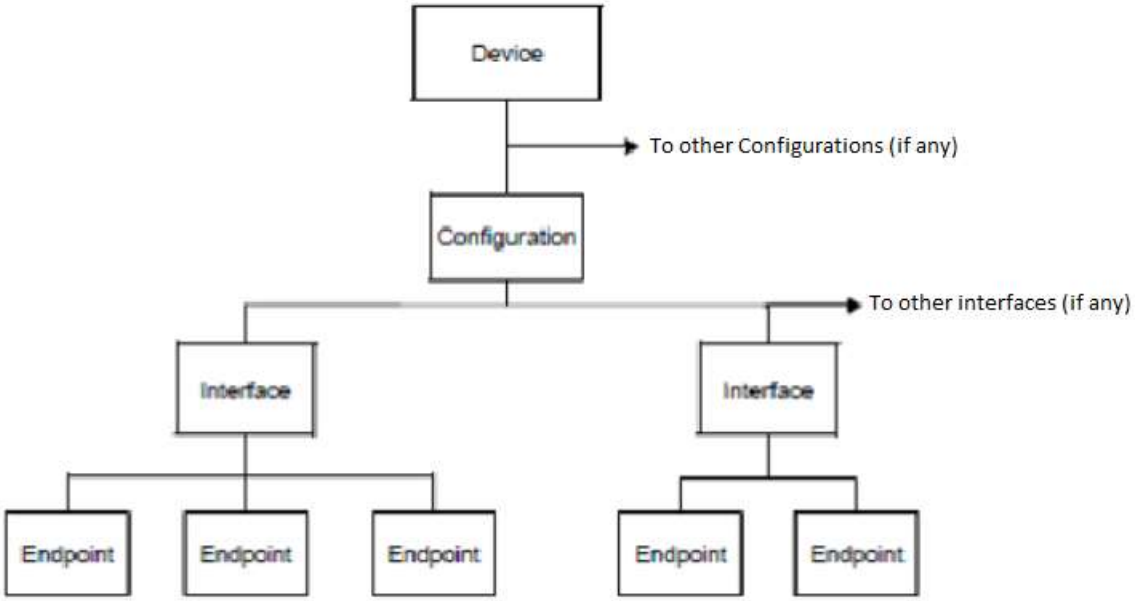


Fig. 6.7 Capas del USB.

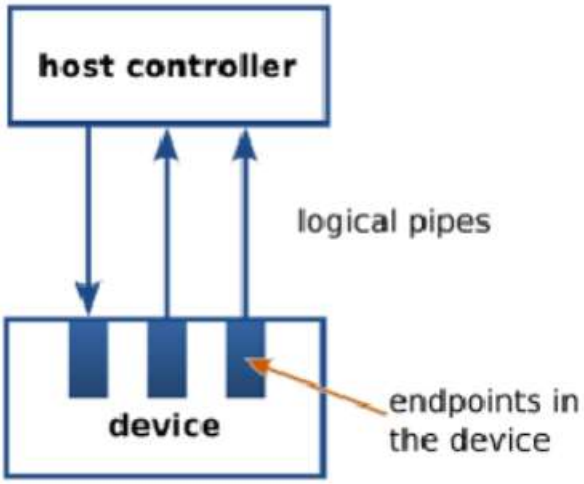


Fig. 6.8 Dirección propia y comunicación con endpoints vía pipes.

Tabla 6.3 Clases de dispositivos. Diversas clases de dispositivos para identificar la funcionalidad y cargar el driver adecuado.

Class	Usage	Description	Examples
00h	Device	Unspecified	(Interface descriptors are used for determining the required drivers.)
01h	Interface	Audio	Speaker, microphone, sound card, MIDI
02h	Both	Com.and CDC Ctrl	Ethernet adapter, modem
03h	Interface	Human interf. device (HID)	Keyboard, mouse, joystick
05h	Interface	Physical Interf. Device (PID)	Force feedback joystick
06h	Interface	Image (PTP / MTP)	Webcam, scanner
07h	Interface	Printer	Laser printer, inkjet printer, CNC machine
08h	Interface	Mass storage	USB flash drive, card reader, digital audio player, digital cam, ext drive
09h	Device	USB hub	Full bandwidth hub
0Ah	Interface	CDC-Data	(This class is used together with class 02h)
0Bh	Interface	Smart Card	USB smart card reader
0Dh	Interface	Content security	Fingerprint reader
0Eh	Interface	Video	Webcam
0Fh	Interface	Personal Healthcare	Pulse monitor (watch)
DCh	Both	Diagnostic Device	USB compliance testing device
E0h	Interface	Wireless Controller	Wi-Fi adapter, Bluetooth adapter
EFh	Both	Miscellaneous	ActiveSync device
FEh	Interface	Application-specific	IrDA Bridge, Test & Measurement Class (USBTMC), USB DFU
FFh	Both	Vendor-specific	(This class code indicates that the device needs vendor specific drivers.)

Endpoints (ver fig. 6.8)

- En los dispositivos (el host no posee)
- Extremos de cada comunicación
- Buffer en el dispositivo que transmite o recibe datos
    - Se definen con un número (0-15) y un sentido
        - In endpoint: provee datos al host
        - Out endpoint: recibe datos provenientes del host
    - Cada dispositivo debe tener configurado su endpoint 0 para control (IN-OUT)
    - Limitaciones en cantidad de endpoints en función de tipo de dispositivo: Low, high o full speed.

Flujo de datos en el bus mediante transferencias, transacciones y paquetes (ver fig. 6.9)

Tipos de transferencia:

- Isocr_onica (isochronous): garantiza tasa de transferencia a costa de perder datos (por ejemplo video en tiempo real).
- Usando interrupciones (interrupt): latencia de respuesta acotada (ejemplo mouse, teclado)
- Masivas (bulk): largas y espor_adicas (consumen todo el ancho de banda disponible { por ejemplo Hard Drives)
- De control (control): para enviar comandos cortos

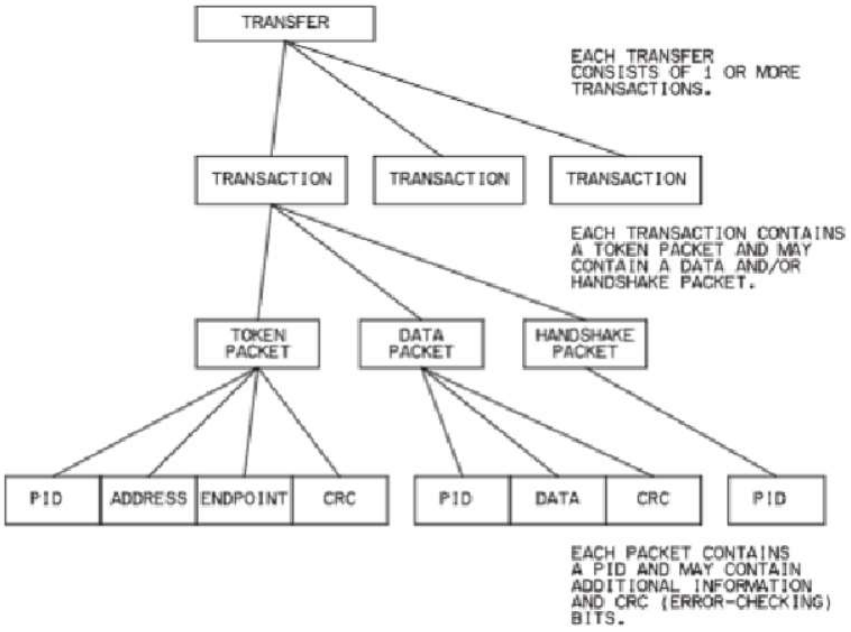


Fig. 6.9 Una transferencia de USB 2.0 consiste en transacciones. Las transacciones contienen paquetes y los paquetes contienen un paquete identificador (PID) y algunas veces información adicional.

Pipes: establecidos por el host: (ver fig. 6.8)

- Stream pipes: unidireccional (isocrónica, interrupciones o masivas).
- Message pipes: bidireccional (control).

Tipos de transferencias

Tabla 6.4 Cada uno de los cuatro tipos de transferencias USB disponibles para diferentes usos.

Transfer Type	Control	Bulk	Interrupt	Isochronous
Typical Use	Identification and configuration	Printer, scanner, drive	Mouse, keyboard	Streaming audio, video
Support required?	yes	no	no	no
Low speed allowed?	yes	no	yes	no
Maximum packet size; maximum guaranteed packets/interval (SuperSpeed).	512; none	1024; none	1024; 3 / 125 µs	1024; 48 / 125 µs
Maximum packet size; maximum guaranteed packets/interval (high speed).	64; none	512; none	1024; 3 / 125 µs	1024; 3 / 125 µs
Maximum packet size; maximum guaranteed packets/interval (full speed).	64; none	64; none	64; 1 / ms	1023; 1 / ms
Maximum packet size; maximum guaranteed packets/interval (low speed).	8; none	not allowed	8; 1 / 10 ms	not allowed
Direction of data flow	IN and OUT	IN or OUT	IN or OUT (IN only for USB 1.0)	IN or OUT
Reserved bandwidth for all transfers of the type	10% at low/full speed, 20% at high speed & SuperSpeed	none	90% at low/full speed, 80% at high speed and SuperSpeed (isochronous and interrupt combined, maximum)	
Message or Stream data?	message	stream	stream	stream
Error correction?	yes	yes	yes	no
Guaranteed delivery rate?	no	no	no	yes
Guaranteed latency (maximum time between transfers attempts)?	no	no	yes	yes

Fig. 6.10 Una transferencia de USB 2.0 consiste en transacciones. Las transacciones contienen paquetes y los paquetes contienen un identificador de paquete (PID) y algunas veces información adicional.

Componentes de una transferencia

Ver figura 6.9

Tabla 6.5 Cada transacción USB 2.0 tiene dos o tres fases.(No se muestran aquí las transacciones adicionales requeridas para dividir transacciones, el protocolo PING empleado en algunas transferencias y el PRE-paquete que precede la disminución de velocidad en paquetes de baja velocidad).

Transfer Type	Number and Direction of Transactions		Phases (packets)
Control	Setup Stage	One (SETUP)	Token
			Data
			Handshake
	Data Stage	Zero or more (IN or OUT)	Token
			Data
			Handshake
	Status Stage	One (opposite direction of the transaction(s) in the Data stage or IN if there is no Data stage)	Token
			Data
			Handshake
Bulk	One or more (IN or OUT)		Token
			Data
			Handshake
Interrupt	One or more (IN or OUT)		Token
			Data
			Handshake
Isochronous	One or more (IN or OUT)		Token
			Data

Transacciones: Cada transacción se inicia con un paquete que indica la dirección del dispositivo, el número de endpoint y su sentido: (ver fig. 6.8)

Transaction Type	Source of Data	Types of Transfers that Use the Transaction Type	Contents
IN	device	all	data or status information
OUT	host	all	data or status information
Setup	host	control	a request

Las transacciones de Setup son similares a las OUT pero no pueden ser rechazadas por el device. Inician una transferencia de control.

Paquetes USB

Tabla 6.6 Bytes de PID de USB

Type	PID value (msb-first)	Transmitted byte (lsb-first)	Name	Description
Reserved	0000	0000 1111		
Token	1000	0001 1110	**SPLIT**	High-bandwidth (USB 2.0) split transaction
	0100	0010 1101	**PING**	Check if endpoint can accept data (USB 2.0)
Special	1100	0011 1100	**PRE**	Low-bandwidth USB preamble
Handshake			**ERR**	Split transaction error (USB 2.0)
	0010	0100 1011	**ACK**	Data packet accepted
	1010	0101 1010	**NAK**	Data packet not accepted; please retransmit
	0110	0110 1001	**NYET**	Data not ready yet (USB 2.0)
	1110	0111 1000	**STALL**	Transfer impossible; do error recovery
Token	0001	1000 0111	**OUT**	Address for host-to-device transfer
	1001	1001 0110	**IN**	Address for device-to-host transfer
	0101	1010 0101	**SOF**	Start of frame marker (sent each ms)
	1101	1011 0100	**SETUP**	Address for host-to-device control transfer
Data	0011	1100 0011	**DATA0**	Even-numbered data packet
	1011	1101 0010	**DATA1**	Odd-numbered data packet
	0111	1110 0001	**DATA2**	Data packet for high-bandwidth isochronous transfer (USB 2.0)
	1111	1111 0000	**MDATA**	Data packet for high-bandwidth isochronous transfer (USB 2.0)

El dispositivo debe respetar el protocolo USB y sus restricciones temporales,

- Responder a los paquetes del host (cada cierto tiempo).
- En caso contrario, el host puede suponer una desconexión del dispositivo y la consecuente pérdida de funcionalidad digital.

Proceso de enumeración de dispositivos: al conectar por primera vez un dispositivo, el host:

- Negocia tasa de transferencia,
- Asigna dirección única de 7 bits al dispositivo,
- Lee descriptores del dispositivo,
- Asigna y carga un driver para el dispositivo,
- Selecciona una configuración de dispositivo (requerimientos de alimentación, interfaces, etc.).

Descriptores: permiten al Host descubrir las características del dispositivo que se conecta (mediante transferencias de control).

Descriptores

Todos los dispositivos USB deben responder a las solicitudes del host (descriptores estándar USB).

Veremos descriptores para:

- Device
- Configuración
- Interfaz
- Endpoint

Existen descriptores adicionales (Interface Association, SuperSpeed endpoints, String, Binary Object Store, etc.), y tambien particulares para ciertas clases de dispositivos (por ej. HID).

Descriptor USB: Punteros a los descriptores e información del soporte USB.

Tabla 6.7 El campo del bDescriptor Type en un descriptor contiene un valor que identifica al tipo de descriptor.

bDescriptorType	Descriptor Type	Required?
01h	device	Yes.
02h	configuration	Yes.
03h	string	No, unless a driver requires it. Optional descriptive text.
04h	interface	Yes.
05h	endpoint	Yes, to use other than endpoint zero.
06h	device_qualifier	Yes for devices that support both full and high speeds. Not allowed for other devices.
07h	other_speed_configuration	Yes for devices that support both full and high speeds. Not allowed for other devices.
08h	interface_power	No (proposed but never approved or implemented).
09h	OTG	Yes for On-The-Go devices.
0Ah	debug	No.
0Bh	interface_association	Yes for some composite devices.
0Ch	security	For wireless devices.
0Dh	key	
0Eh	encryption type	
0Fh	binary device object store (BOS)	Yes for SuperSpeed devices, wireless devices, and devices that support link power management.
10h	device capability	
11h	wireless endpoint companion	For wireless devices.
30h	SuperSpeed_endpoint_companion	Yes for SuperSpeed devices. Not supported for other speeds

Descriptores – Clases

Tabla 6.8 El descriptor del dispositivo identifica el producto y su fabricante, asigna el tamaño máximo del paquete para el enpoint cero y puede especiiificar una clase de dispositivo.

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes (12h)
1	bDescriptorType	1	The constant DEVICE (01h)
2	bcdUSB	2	USB specification release number (BCD)
4	bDeviceClass	1	Class code
5	bDeviceSubclass	1	Subclass code
6	bDeviceProtocol	1	Protocol Code
7	bMaxPacketSize0	1	Maximum packet size for endpoint zero
8	idVendor	2	Vendor ID
10	idProduct	2	Product ID
12	bcdDevice	2	Device release number (BCD)
14	iManufacturer	1	Index of string descriptor for the manufacturer
15	iProduct	1	Index of string descriptor for the product
16	iSerialNumber	1	Index of string descriptor for the serial number
17	bNumConfigurations	1	Number of possible configurations

Descriptores de configuración

Tabla 6.9 El descriptor de configuración especifica la máxima cantidad de corriente que requerirá el dispositivo en el bus y da la longitud total de los descriptores subordinados.

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes (09h)
1	bDescriptorType	1	The constant CONFIGURATION (02h)
2	wTotalLength	2	The number of bytes in the configuration descriptor and all of its subordinate descriptors
4	bNumInterfaces	1	Number of interfaces in the configuration
5	bConfigurationValue	1	Identifier for Set Configuration and Get Configuration requests
6	iConfiguration	1	Index of string descriptor for the configuration
7	bmAttributes	1	Self/bus power and remote wakeup settings
8	bMaxPower	1	Bus power required in units of 2 mA (USB 2.0) or 8 mA (SuperSpeed).

Descriptores de interfaz

Número de endpoints para la interfaz y clase USB (p/disp. con clase definida por la interfaz).

Tabla 6.10 El descriptor de interfaz especifica el número de endpoints subordinados y podría especificar una Clase de USB.

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes (09h)
1	bDescriptorType	1	The constant Interface (04h)
2	bInterfaceNumber	1	Number identifying this interface
3	bAlternateSetting	1	A number that identifies a descriptor with alternate settings for this bInterfaceNumber.
4	bNumEndpoints	1	Number of endpoints supported not counting endpoint zero
5	bInterfaceClass	1	Class code
6	bInterfaceSubclass	1	Subclass code
7	bInterfaceProtocol	1	Protocol code
8	iInterface	1	Index of string descriptor for the interface

Descriptores de endpoit

Tabla 6.11 El descriptor de endpoint provee información a cerca de la dirección de un endpoint.

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes (07h)
1	bDescriptorType	1	The constant Endpoint (05h)
2	bEndpointAddress	1	Endpoint number and direction
3	bmAttributes	1	Transfer type and supplementary information
4	wMaxPacketSize	2	Maximum packet size supported
6	bInterval	1	Service interval or NAK rate

Clases de dispositivos

CDC: Communications Device Class

Para dispositivos de comunicaciones: teléfonos, modems, terminales y adaptadores ISDN, dispositivos con puertos COM virtuales (ej Arduino Uno). . .

Para dispositivos con funciones de red: ADSL modems, cablemódems, adaptadores y hubs ethernet. . .

Administrar dispositivos, llamadas, transmitir datos y notificaciones.

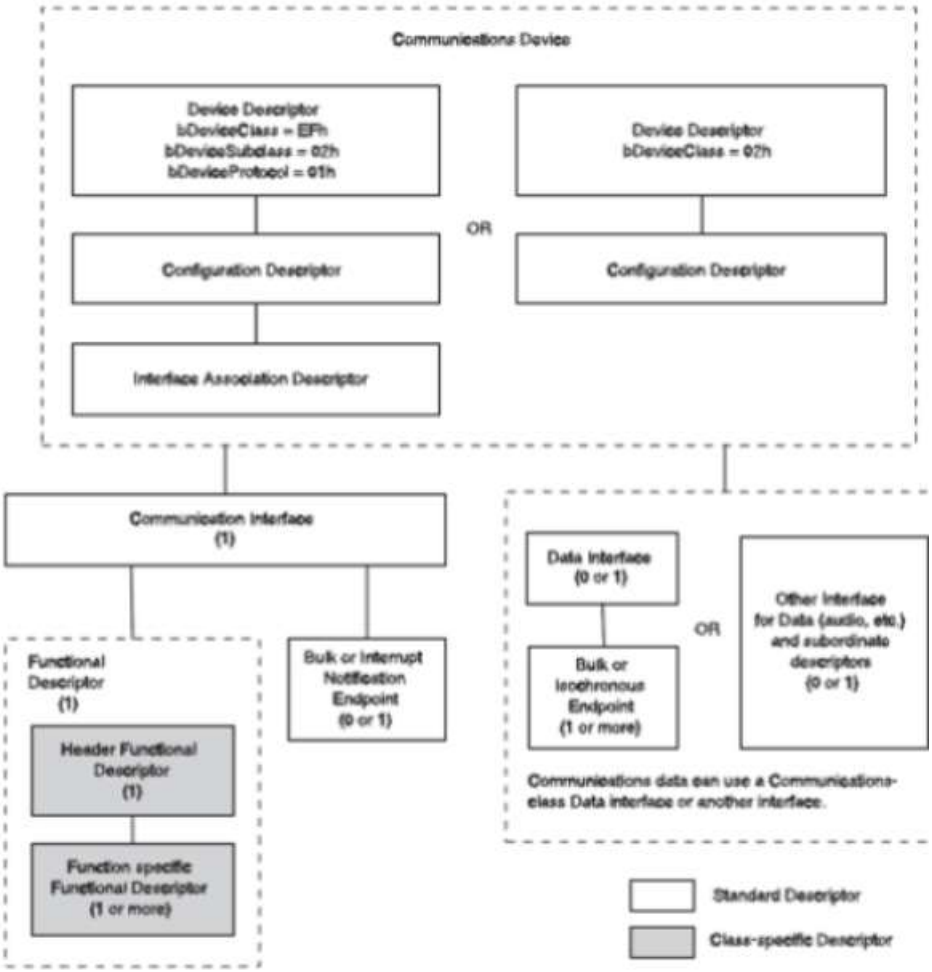


Fig. 6.10 Un dispositivo de comunicaciones provee interfases para datos y notificaciones.

Clases de dispositivos

DFU Class

Ejemplo ATmega16U2:

Permite colocar un dispositivo en modo de actualización del Firmware.

Solicitudes especificas en el protocolo.

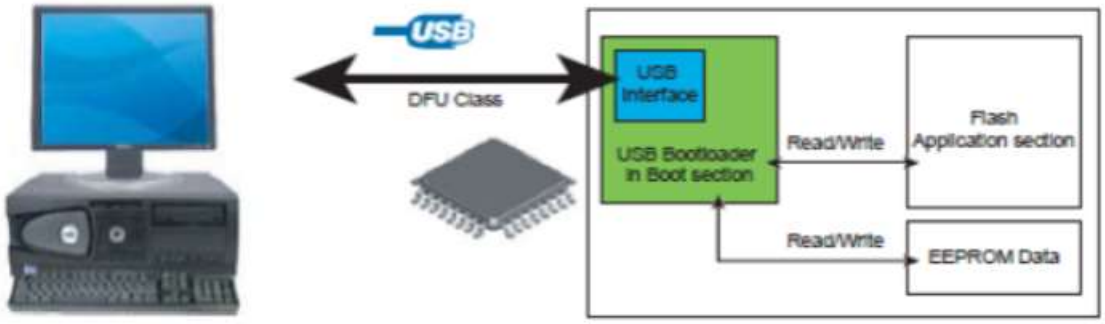


Fig. 6.11Ambiente físico.

Tabla 6.12 Solicitudes especificas en el protocolo.

Request	Description
DFU_Detach	Detach and re-attach to the bus or wait for bus reset within the time period specified in the DFU Functional descriptor. On reattach or a reset within the specified time, enumerate using the DFU-mode descriptors.
DFU_Dnload	Accept new firmware in the request's Data stage. A request with wLength = 0000h means all of the firmware has been transferred.
DFU_Upload	Send firmware to the host in the request's Data stage.
DFU_GetStatus	Return status and error information. On error, enter the dfuError state.
DFU_ClrStatus	Clear the dfuError state reported in response to a DFU_GetStatus request and enter the dfuIdle state.
DFU_GetState	Same as DFU_GetStatus but with no change in state on error.
DFU_Abort	Return to the dfuIdle state.

Clases de dispositivos

HID (Human Interface Device) Class

Incluye mouses, teclados, joysticks, etc.

Los SOs en los hosts, suelen tener drivers para HIDs.

Limitados a transferencias control e interrupt.

Los datos HID viajan en reports (estructuras bien de_nidas).

- Input item: lleva info hacia el host.
- Output item: lleva info hacia el dispositivo.
- Feature item: es bidireccional.

Solicitudes especificas en el protocolo para obtener reports.

Mass Storage (MSC):

- Para dispositivos de almacenamiento masivo.
- Discos rígidos, unidades de CD/DVD, cámaras que presentan su contenido mediante un sistema de archivos, etc.
- Usan transferencias bulk para intercambiar datos.

Media Transfer Protocol (MTP)

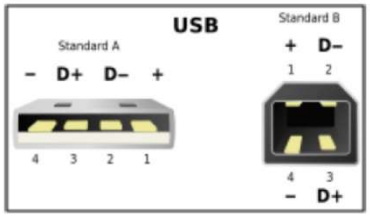
- Extensión al Picture Transfer Protocol (PTP - USB Image Class) usado para transferir imágenes (cámaras digitales)
- Para transferir archivos transaccionalmente sin requerir acceso exclusivo al medio (ejemplo acceso a memoria externa en smartphones sin bloquear apps).
- Utilizan trasferencias de tipo bulk e interrupt.

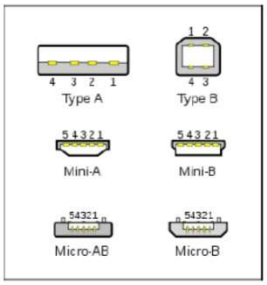
Otras clases:

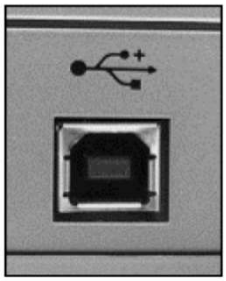
- Audio (streaming de audio / MIDI)
- IrDA Bridge: intercambio de datos por enlaces infrarrojos.
- Personal Health Care
- Printer: impresoras, CNCs
- Smart Cards
- Still Image Capture: c_amaras, scanners, PTP, MTP, etc.
- Video
- Etc. . .

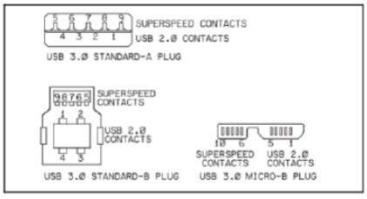
Dispositivos genéricos (HID, CDC, MSC, etc)
Dispositivos Vendor-Specific (drivers específicos).

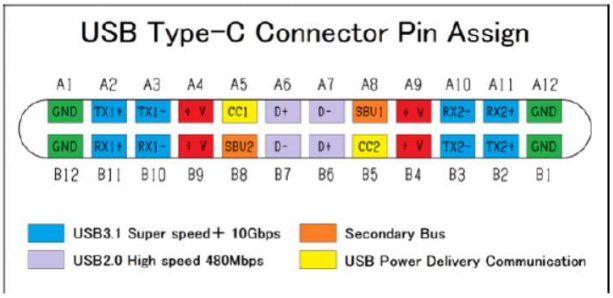
Señales y conectores

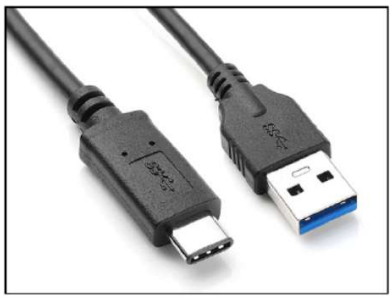


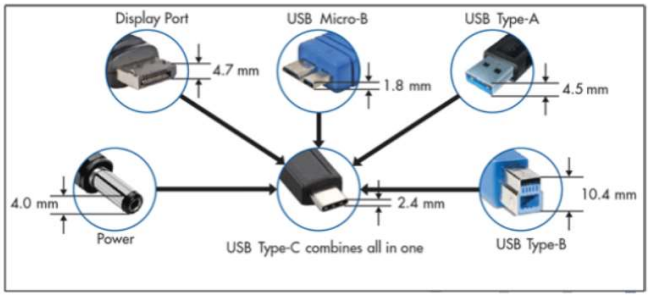












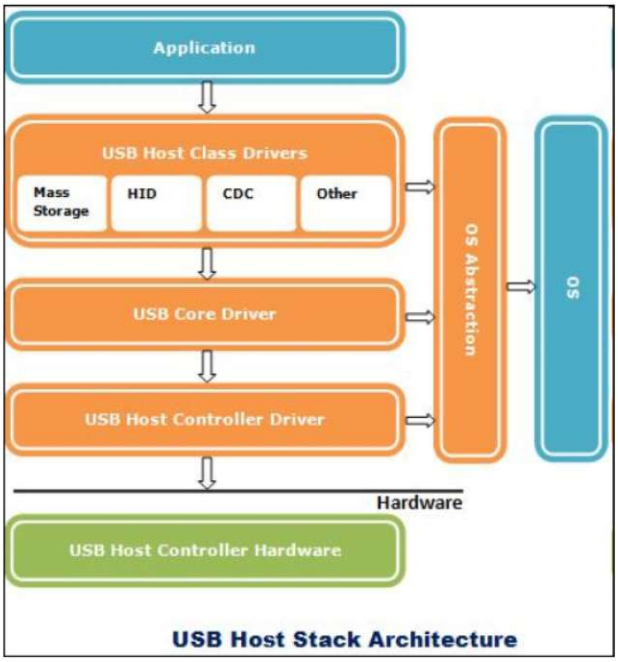
También se definen consideraciones en cuanto a consumo y alimentación de los puertos y dispositivos.

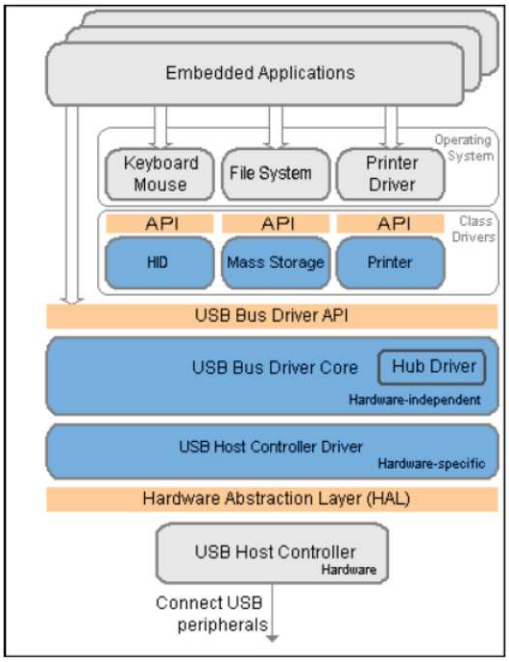
- Devices bus-powered (5V, 3.3V).
- Devices self-powered.

Numerosas variantes del estándar:

- Distintas versiones: USB 1.0, 1.1, 2.0 y 3.0 (SuperSpeed)
- Sleep & charge: Para carga con hosts suspendidos.
- Powered USB: USB para alimentación.
- USB On the Go (OTG): Para conexión de 2 dispositivos sin hosts (par a par): Uso de embedded hosts.
- Wireless USB.

Dada la complejidad del protocolo, se suele manejar con dispositivos específicos en los SoC y mediante librerías que gestionan los diversos niveles en el stack.





Bluetooth [9]

La tecnología de radio inalámbrica de corto alcance de Bluetooth permite que dos dispositivos se comuniquen, sin necesidad de infraestructura de red.

Las PCs y dispositivos móviles usan tecnología Bluetooth para conectarse a los periféricos inalámbricos como audífonos, teclados, ratones, altavoces, sistemas de navegación y de entretenimiento para automóviles e incluso dispositivos médicos.

[10] La tecnología inalámbrica de corto alcance, Bluetooth, permite que dos dispositivos se conecten directamente sin necesidad de infraestructura de red de respaldo como un enrutador inalámbrico o un punto de acceso. Hoy en día, la tecnología Bluetooth es la más comúnmente usada por las personas de todo el mundo para conectar dispositivos inalámbricos como audífonos, teclados, ratones y altavoces tanto a PCs como a dispositivos móviles.

[9] La tecnología Bluetooth funciona en frecuencias de radio en el rango de 2,4 GHz, esto significa que tiene la capacidad de atravesar paredes y maletines, por lo cual es ideal tanto para el trabajo móvil, como el trabajo en oficinas. Los dos estándares de Bluetooth actualmente en uso son:

Bluetooth Classic, que admite dos velocidades de datos distintas, la velocidad básica (BR) y la velocidad de datos mejorada (EDR).

Bluetooth Low Energy (LE), que está optimizado para bajo consumo de energía y se usa principalmente para aplicaciones que están restringidas por la autonomía de la batería. Bluetooth LE no se usa comúnmente para intercambiar grandes cantidades de datos, pero ofrecerá compatibilidad con una mayor calidad de audio y más opciones de escucha que Bluetooth Classic.

En los primeros días, muchas PCs dependían de dispositivos externos para conectarse a los periféricos de Bluetooth, a menudo mediante un puerto USB. Este enfoque proporcionó una funcionalidad esencial de Bluetooth, pero era a menudo engorroso para el usuario final. Hoy en día, la mayoría de PCs cuentan con tarjetas de red integradas con funcionalidad Wi-Fi y Bluetooth. Estas permiten un mejor desempeño y coordinación en ambas capacidades de radio, lo que ayuda a evitar interferencias y garantizar velocidades de transferencia de datos adecuadas.

El grupo Bluetooth SIG (Special Interest Group), ha desarrollado la especificación Bluetooth, que permite el desarrollo de aplicaciones de comunicación de datos de manera inalámbrica.

La especificación, define un conjunto completo de protocolos, los cuales dan gran flexibilidad al estándar para operar una cierta variedad de aplicaciones. La iniciativa Bluetooth, tiene como objetivo aumentar la efectividad de las comunicaciones entre cortas distancias, tanto en el área de trabajo como en los espacios públicos.

[11] La clasificación de los dispositivos Bluetooth como "Clase 1", "Clase 2" o "Clase 3" es únicamente una referencia de la potencia de transmisión del dispositivo, siendo totalmente compatibles los dispositivos de una clase con los de la otra. En la siguiente tabla se muestra los rangos de cada clase:

CLASE	POTENCIA MÁX. PERMITIDA (mW)	POTENCIA MÁX. PERMITIDA (dBm)	RANGO APROXIMADO
Clase 1	100 mW	20 dBm	~100 metros
Clase 2	2.5 mW	4 dBm	~20 metros
Clase 3	1 mW	0 dBm	~1 metro

La cobertura efectiva de un dispositivo de clase 2 se extiende cuando se conecta a uno de clase 1. Esto es así gracias a la mayor sensibilidad y potencia de transmisión del dispositivo de clase 1. Esto es, la mayor potencia de transmisión del dispositivo de clase 1 permite que la señal llegue con energía suficiente hasta el de clase 2. Por otro lado, la mayor sensibilidad del dispositivo de clase 1 permite recibir la señal del otro pese a ser más débil.

Redes Bluetooth

La topología de las redes Bluetooth puede ser punto-a-punto o punto-a multipunto.

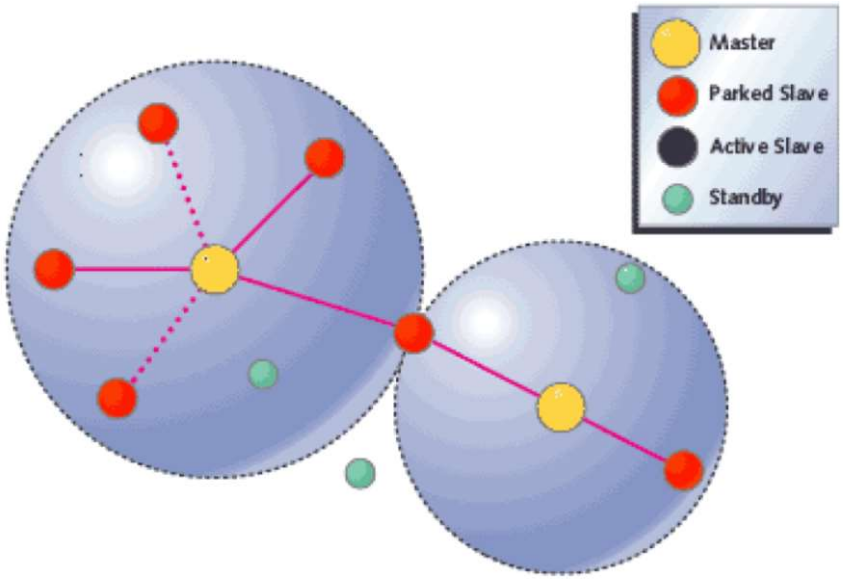


Fig. 6.12 Topología Bluetooth, mostrando la clasificación de dispositivos, piconet y scatternet.

Los dispositivos, se comunican en redes denominadas piconets. Estas redes tienen posibilidad de crecer hasta tener 8 conexiones punto a punto. Además, se puede

extender la red mediante la formación de scatternets. Una scatternet es la red producida cuando dos dispositivos pertenecientes a dos piconets diferentes, se conectan.

En una piconet, un dispositivo debe actuar como master, enviando la información del reloj (para sincronizarse) y la información de los saltos de frecuencia. El resto de los dispositivos actúan como slaves (esclavos).

Conexiones Bluetooth

Las conexiones Bluetooth, son establecidas a través de la siguiente técnica:

Standby: Los dispositivos en un "piconet" que no están conectados, están en modo standby, ellos escuchan mensajes cada 1,28 segundos, sobre 32 saltos de frecuencias.

Page/Inquiry: Si un dispositivo desea hacer una conexión con otro dispositivo, éste le envía un mensaje de tipo page, si la dirección es conocida; o una petición a través de un mensaje de page, si éste no es conocido. La unidad "master" envía 16 page message idénticos, en 16 saltos de frecuencias, a la unidad "slave". Si no hay respuesta, el "master" retransmite en los otros 16 saltos de frecuencia. El método de Petición (inquiry) requiere una respuesta extra por parte de la unidad "slave", desde la dirección MAC, que no es conocida por la unidad "master".

Active: Ocurre la transmisión de datos.

Hold: Cuando el "master" o el "slave" desean, puede ser establecido un modo en el cual no son transmitidos datos. El objetivo de esto es conservar el poder.

Sniff: El modo sniff, es aplicable solo para las unidades "slaves", es para conservar el poder. Durante este modo, el "slave", no toma un rol activo en la "piconet", pero escucha a un reducido nivel.

Park: El modo park es un nivel más reducido, que el modo hold. Durante este, el "slave" es sincronizado a la "piconet", por eso no requiere un reactivación completa, pero no es parte del tráfico. En este estado, ellos no tienen direcciones MAC y solo escuchan para mantener su sincronización con el "master" y chequear los mensajes de broadcast.

En la siguiente figura se puede ver un resumen de los distintos estados en los que se puede encontrar un dispositivo Bluetooth y sus posibles evoluciones entre estados.

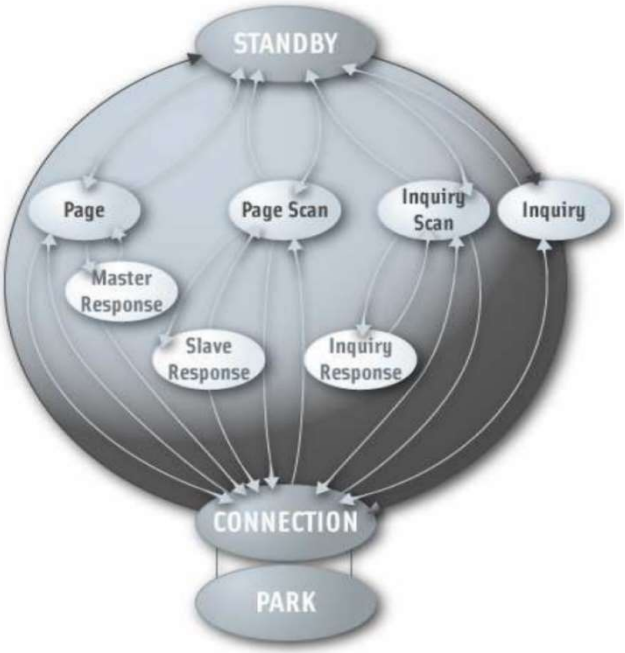


Fig. 6.13 Distintos estados en los que se puede encontrar un dispositivo Bluetooth y sus posibles evoluciones entre estados.

Protocolos del Bluetooth

Fig. 6.14 Pila de protocolos del Bluetooth.

Uno de los principales objetivos de la tecnología Bluetooth es conseguir que aplicaciones de dispositivos diferentes mantengan un dialogo fluido. Para conseguirlo, ambos, deben ejecutarse sobre la misma pila de protocolos.

La pila está constituida por dos clases de protocolos. Una primera clase llamada de protocolos específicos que implementa los protocolos propios de Bluetooth. Y una segunda clase formada por el conjunto de protocolos adoptados de otras

especificaciones. Esta división en clases en el diseño de la pila de protocolos de Bluetooth permite aprovechar un conjunto muy amplio de ventajas de ambas. Por un lado, al implementar protocolos específicos de Bluetooth permite utilizar los beneficios que aporta la adopción de la tecnología Bluetooth. Por otro lado la utilización de protocolos no específicos ofrece la ventaja de la interacción de esta tecnología con protocolos comerciales ya existentes. Así como la posibilidad de que Bluetooth este abierto a implementaciones libres o nuevos protocolos de aplicación de uso común. La pila de protocolos se puede dividir en cuatro capas lógicas:

Núcleo de Bluetooth : Radio, Banda Base, LMP, L2CAP, SDP

Sustitución de cable: RFCOMM

Protocolos adoptados: PPP, UDP, TCP, IP, OBEX, WAP, IRMC, WAE

Control de telefonía: TCS-binary, AT-Commands

El llamado núcleo de Bluetooth ha sido implementado en su totalidad por el SIG, no obstante otros como RFCOMM y TCS-binary pese a ser desarrollados por el propio SIG, los han desarrollado siguiendo las recomendaciones de otras instituciones de telecomunicaciones.

El resto de las capas lógicas de sustitución de cable, de control de telefonía y de protocolos adoptados agrupan a los protocolos orientados a aplicación, permitiendo así a las diferentes aplicaciones existentes o desarrolladas en el futuro poder correr sobre el núcleo de Bluetooth. Como es una norma abierta en cuanto a los protocolos que corren encima de los protocolos específicos de transporte, se pueden hacer implementaciones que usen protocolos tan usados como FTP o HTTP por ejemplo.

Radio Bluetooth (RF):

Bluetooth fue diseñado para operar en un entorno de radio frecuencia ruidosa (LANs, mandos, hornos microondas), y para ello utiliza un esquema de reconocimiento rápido y saltos de frecuencia para garantizar la robustez del enlace. Este sistema opera en la banda de frecuencia de 2.4 GHz, libre para ISM (Industrial, Científica, Medica) más exactamente comenzando en 2.402 GHz y acabando en 2.4835 GHz. Con canales RF de f = 2402 + k MHz siendo k = 0.78.

El espacio entre canales es de 1 MHz, no obstante es necesario tener unos márgenes de protección respecto al ancho de banda de trabajo, así pues, el límite superior de protección es de 2 MHz y un límite inferior es de 3,5 MHz.

Características de la modulación:

La modulación que emplea Bluetooth es GFSK (Gaussian Frequency Shift Keying) con un producto ancho de banda por tiempo BT=0.5. Este tipo de modulación permite un bajo coste. El índice de modulación debe estar entre 0.28 y 0.35. Un uno binario se representa por una desviación positiva de frecuencia y un cero binario como una desviación negativa. La desviación mínima no ha de ser menor de 115 KHz.

Características del dispositivo receptor:

El aspecto más importante en el dispositivo receptor es el nivel de sensibilidad. Para poder medir una tasa de error de bit, el equipo receptor envía de vuelta la información decodificada. Para una tasa de error o BER (Bit Error Rate) del 0.1% se define el nivel de sensibilidad de un receptor Bluetooth mayor o igual a –70dBm.

Banda Base (BaseBand)

Bluetooth brinda una conexión punto-a-punto o conexión punto-a-multipunto. Como se ha dicho anteriormente, dos o más unidades compartiendo el mismo canal forman una piconet o picored. Cada piconet tiene una secuencia de salto diferente y como elementos a destacar tenemos un maestro que puede tener hasta siete esclavos activos, además pueden haber muchos más esclavos en estado parked o aparcados, en realidad un número ilimitado de ellos. Estos esclavos no están activos en el canal sin embargo están sincronizados con el maestro con el fin de asegurar una rápida iniciación de comunicación. El maestro (Master) es el responsable de la sincronización entre los dispositivos de la piconet, su reloj y saltos de frecuencia controlan al resto de dispositivos. Además el maestro es quien, de manera predeterminada, lleva a cabo el procedimiento de búsqueda y establecimiento de la conexión. Los esclavos simplemente se sincronizan y siguen la secuencia de saltos determinada por el maestro.

La topología Bluetooth permite la interconexión de varias piconets formando una scatternet, Aunque no existe sincronización entre piconets, un dispositivo puede pertenecer a varias de ellas haciendo uso de la multiplexación por división del tiempo (TDD), aunque el dispositivo solo está activo en una piconet a la vez.

Técnica TDD (Time Division Duplex):

El canal físico contiene 79 frecuencias de radio diferentes, las cuales son accedidas de acuerdo a una secuencia de saltos aleatoria. El valor de saltos estándar es de 1600 saltos/s. El canal está dividido en timeslots o slots (ranuras de tiempo), cada slot corresponde a una frecuencia de salto y tiene una longitud de 625 us. Cada secuencia de salto en una piconet está determinada por la dirección del maestro (48 bits) de la piconet. Todos los dispositivos conectados a la piconet están

sincronizados con el canal en salto y tiempo. En una transmisión, cada paquete debe estar alineado con el inicio de un slot y puede tener una duración de hasta cinco timeslots. Durante la transmisión de un paquete la frecuencia es fija. Para evitar fallos en la transmisión (crosstalk), el maestro inicia enviando en los timeslots pares y los esclavos en los timeslots impares.

Existen dos tipos de enlaces físicos entre maestros y esclavos:

Enlace SCO (Synchronous Connection-Oriented):

El enlace SCO es una conexión simétrica punto-a-punto con un ancho de banda fijo entre el maestro y un esclavo específico. Para lograr la comunicación, el enlace SCO reserva slots en intervalos regulares en la iniciación, por esto el enlace puede ser considerado como una conexión de conmutación de circuitos. En este tipo de enlace no es necesario asegurar la entrega y suele ser utilizado para comunicaciones de voz.

Enlace ACL (Asynchronous Connection-Less):

El enlace ACL es una conexión simétrica o asimétrica punto-a-multipunto entre el maestro y uno o más esclavos activos en la piconet sin reserva de ancho de banda.

Este enlace de comunicación es un tipo de conexión de conmutación de paquetes. Aquí, a diferencia del anterior, se necesita asegurar la entrega de datos y es utilizado para transferencia de datos sin requerimientos temporales.

Establecimiento de conexiones en Bluetooth:

Para establecer nuevas conexiones se utilizan los procedimientos de acceso que son los de búsqueda o paging y los de pregunta o inquiry. Si no se conoce nada sobre el dispositivo remoto debe seguirse tanto el procedimiento inquiry como el de paging. Si se conocen algunos detalles del dispositivo remoto sólo será necesario el procedimiento de paging.

Pregunta (Inquiry):

El procedimiento de “inquiry” permite a un dispositivo descubrir qué dispositivos están en su zona de cobertura, determinando sus direcciones y el reloj de todos aquellos que respondan al mensaje de búsqueda. Entonces, si el dispositivo emisor lo desea, establecerá una conexión con alguno de los dispositivos descubiertos.

El mensaje de búsqueda no contiene ningún tipo de información sobre la fuente emisora del mensaje, no obstante, puede indicar qué clase de dispositivos deberían responder. Para poder conseguir esto existe un código de acceso de pregunta (GIAC)

para preguntar por algún tipo de dispositivo en especial, y una serie de códigos de acceso de pregunta dedicados (DIAC) para tipos de dispositivos.

Así pues un dispositivo que quiera conectar con otro dispositivo en concreto continuamente transmite el mensaje GIAC en diferentes frecuencias de salto. La secuencia de saltos está determinada en el parte menos significativa de la dirección del GIAC, incluso cuando se utilizan los DIAC. Un dispositivo que quiera ser descubierto, cada cierto tiempo entrará en un estado de escáner de preguntas llamado “inquiry scan” para atender a estos mensajes.

Una vez atendida la pregunta, el dispositivo destino, entrará en el modo”inquirí response” y transmite un mensaje de respuesta que consiste en una paquete FHS (Frequency Hop Synchronization), que tiene los parámetros del dispositivo. El maestro escucha las diferentes respuestas, pero nada más leer una respuesta continua escaneando por diferentes respuestas. En el caso de que exista contienda entre diferentes dispositivos, éstos, al no recibir respuesta del maestro, esperan un número aleatorio de slots y se mantienen a la escucha de un nuevo mensaje de pregunta del maestro.

Búsqueda (Paging)

El procedimiento de “paging” sigue al de “inquiry”. El procedimiento de paging pregunta por la dirección de un dispositivo Bluetooth con el que se quiere establecer la conexión. Este identificador del dispositivo se obtenido de las siguientes tres formas:

- Obtenida en la respuesta de un “inquiry”.
- Introducida por el usuario.
- Preprogramada por el fabricante del dispositivo.

Entonces el dispositivo maestro, que se encuentra en el estado page, inicia la transmisión transmite el código de acceso o DAC (Device Access Code) al dispositivo que se desea que sea esclavo de forma repetida en diferentes canales de salto. Debido a que los relojes del maestro y del esclavo no están sincronizados, el maestro no sabe exactamente cuándo y en qué frecuencia de salto se activará el esclavo por lo tanto maestro se quedará a la escucha entre los diversos intervalos de transmisión hasta recibir respuesta del esclavo.

Después de haber recibido su propio código de acceso de dispositivo, el esclavo transmite un mensaje de respuesta, simplemente indicará su código de acceso, y se queda activado en espera de la llegada del paquete FHS (Frequency Hop Synchronization). Cuando el maestro ha recibido este paquete ACK, envía un paquete

de control con información acerca de su reloj, dirección, clase de dispositivo, etc. El maestro se queda a la espera de una respuesta.

El esclavo se activa y responde con un nuevo mensaje ACK donde envía de nuevo su dirección y a la vez cambia el código de acceso del canal y su reloj, tomando los del maestro incluido en el paquete FHS. El esclavo establece la conexión usando para ello el reloj y la BD_ADDR del maestro para determinar la secuencia de salto del canal y el código de acceso.

Si el maestro no obtiene esta respuesta en un determinado tiempo, él reenvía el paquete de control. Si el esclavo excede el tiempo de espera, entonces vuelve al estado de page scan. Si es el maestro quien lo excede, entonces vuelve al estado de page e informa a las capas superiores. Con el ACK, el maestro entra en modo de conexión establecida y usa su BD_ADDR para cambiar a una nueva secuencia.

LMP: Link Manager Protocol:

El siguiente protocolo específico se encarga de la gestión del enlace entre dispositivos Bluetooth, de la seguridad, del control de paquetes, potencia, calidad del servicio y control de la piconet (conmutación maestro esclavo).

Establecimiento de conexión:

Tras haberse completado el procedimiento de búsqueda o Paging, ya se está listo para establecer una conexión LMP. En primer lugar el dispositivo emisor envía la primitiva LMP_host_connection_req. El dispositivo receptor recibe el mensaje y obtiene información sobre la conexión que se va a abrir. Este dispositivo remoto puede aceptar (LMP_accepted) o rechazar (LMP_not_accepted) esa petición de conexión. Una vez establecidas todas las configuraciones necesarias, los dos dispositivos se mandan LMP_setup_complete. Después de esto, se procederá a la transmisión de los paquetes de los diferentes canales lógicos que emplea LMP.

LMP debe gestionar un gran número de procedimientos que permitan implementar toda su funcionalidad, esto hace que cada procedimiento tenga sus propias PDUs que se deben intercambiar entre las dos entidades de comunicación. Vamos a hacer una breve referencia a los procedimientos LMP:

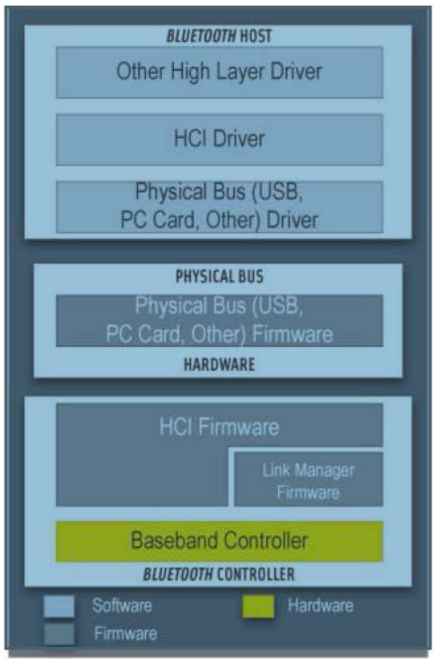
1. Autenticación.
2. Pairing.
3. Cambio de la clave de enlace.
4. Cambio de la clave de enlace actual.
5. Encriptado.
6. Petición del offset del reloj

7 Información del offset de slot.
8 Petición de información de temporización.
9 Información de la versión LMP
10 Información de las características soportadas.
11 Conmutación del papel esclavo-maestro.
12 Petición de nombre.
13 Desconexión.
14 Control de potencia.
15 Petición modo hold
16 Petición modo sniff.
17 . Petición modo park.
18 Petición calidad de servicio.
19 Establecimiento enlaces SCO.
20 Control de paquetes multi-slot.
21 Supervisión del enlace.
22 Repuesta General.

HCI: Host Controller Interface:

Proporciona una interface de comandos al controlador de banda base y al manejador de enlace para acceder a los parámetros de configuración. Esta interfaz proporciona un uniforme método de acceso a las capacidades de banda base Bluetooth.

Pueden existir varias capas entre el driver HCI sobre el sistema host y el firmware HCI en el hardware Bluetooth. Estas capas intermedias, host controller transport layer, proporcionan la capacidad de intercambio de información.



L2CAP: Logical Link Control and adaption Protocolo:

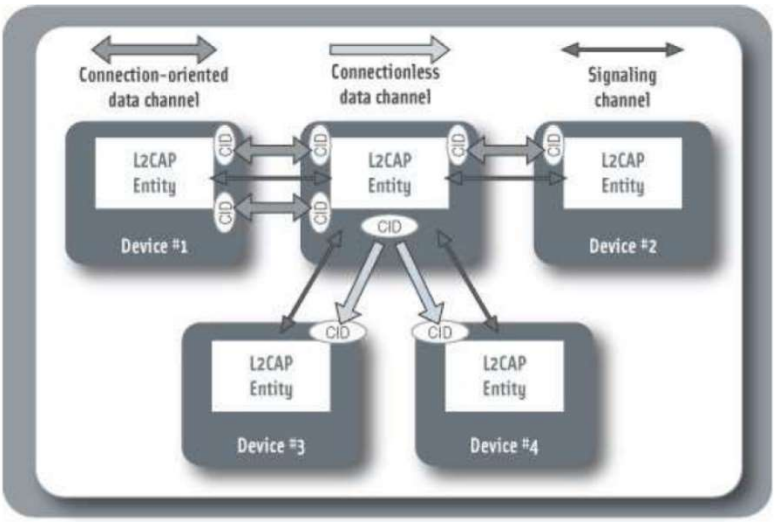
L2CAP es un protocolo que se encuentra por encima del anterior protocolo (LMP), se encarga de adaptar los protocolos superiores al protocolo de banda base. Sus tres principales funciones son:

- Multiplexación de protocolos de alto nivel
- Segmentación y reensamblado de paquetes largos (hasta 64 kbytes)
- Descubrimiento de dispositivos y calidad de servicio

Para cumplir estas funciones la arquitectura L2CAP debe cumplir ciertos requisitos:

a) L2CAP ofrece un servicio orientado a conexión donde el identificador del canal es utilizado en cada conexión, asumiendo que este canal es full-duplex y fiable, además se tiene que especificación del flujo de QoS asignada a cada dirección del canal.

b) L2CAP está basado en datagramas y no en flujos continuos. Se debe notar que en L2CAP, se conservan los límites del paquete, así como que no se realiza retransmisión ni control de flujo.



SDP: Service Discovery Protocol:

SDP proporciona un mecanismo que permite a las aplicaciones descubrir cuales son los servicios disponibles en su entorno y determinar las propiedades específicas de éstos. Los servicios disponibles cambian continuamente debido al dinamismo existente en el entorno, por lo que la búsqueda de servicios en Bluetooth difiere de la búsqueda de servicios en un red fija tradicional.

SDP debe proporcionar las siguientes funcionalidades en su versión 1.0:

- SDP debe permitir la búsqueda de servicios basados en atributos específicos.
- SDP debe permitir que los servicios sean descubiertos basándose en la clase de servicio.
- SDP debe permitir averiguar las características de un servicio sin tener conocimiento a priori de dicho servicio.
- SDP debe proporcionar medios para descubrir nuevos servicios (proximidad de un nuevo dispositivo, arranque de una aplicación) así como para indicar la no disponibilidad de servicios inicialmente visibles.
- SDP debe permitir el almacenar información sobre servicios de forma temporal para mejorar la eficiencia del protocolo.
- SDP debe descubrir la información de los servicios de forma incremental para evitar las transferencias excesivas de información sobre servicios no vayan a utilizarse.
- SDP proporcionar complejidad adecuada para ser utilizado en dispositivos con prestaciones limitadas.

No obstante SDP debe proporcionar los siguientes servicios en las sucesivas versiones:

- No se proporciona acceso a los servicios, sólo acceso a la información sobre los Servicios.
- No aparece negociación de parámetros de servicio.
- No se proporcionan mecanismos para la tarificación por el uso de los servicios.
- No se proporciona al cliente la capacidad de controlar o cambiar la operación de un servicio.
- No se proporciona notificación de eventos para los casos en que los servicios no estén disponibles o cuando se modifican los atributos de los servicios.
- SDP 1.0 no define un API (Application Programming Interface).
- No se soportan agentes que realicen funciones tales como darse de alta en un servicio.

Registro de un servicio:

El registro de servicio está formado por un conjunto de atributos que describen un servicio determinado. Existen dos tipos de atributos, los llamados atributos universales que son comunes a todos los tipos de servicio y los llamados atributos específicos que tal como indica el nombre son específicos a una clase de servicio.

Cada atributo de un registro de servicio consta de dos partes, un identificador de propiedad y un valor de propiedad. El identificador de propiedad es un único número

de 16 bits que distingue cada propiedad de servicio de otro dentro de un registro. El valor de propiedad es un campo variable que contiene la información.

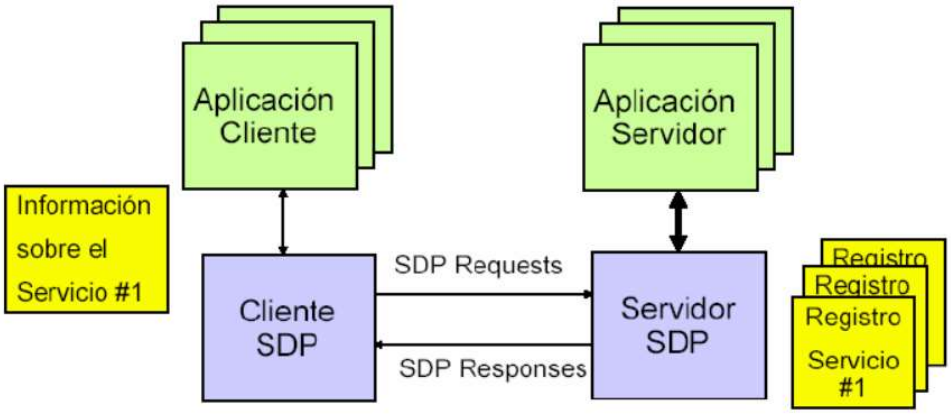
Un servidor SDP estructura los servicios en registros y mantiene una lista de apuntadores (Service Record Handle) a cada uno de ellos.

Funcionamiento de SDP

Tal como se ve en la figura, la comunicación SDP implica un servidor SDP y un cliente SDP. Se pueden dar dos escenarios posibles para que el cliente realice una petición de un servicio.

El cliente busca un servicio en un dispositivo en particular al cual el usuario se ha conectado "conscientemente".

El cliente realizando conexiones "inconscientemente" con los dispositivos vecinos.



Vistos estos dos escenarios se pueden definir varios mensajes de petición/repuesta por parte tanto del cliente como del servidor:

- Mensajes de petición del cliente:

- Petición de búsqueda de servicio: el cliente genera una petición para localizar los registros de servicio que concuerden con un patrón de búsqueda dado como parámetro.
- Petición de propiedad de servicio: Una vez el cliente ya ha recibido los servicios deseados, puede obtener mayor información de uno de ellos dando como parámetros el registro de servicio y la lista de propiedades deseadas.
- Petición de búsqueda y propiedad de servicio: se suministran un patrón de servicio con servicios deseados y una lista de propiedades deseadas que concuerden con la búsqueda.

- Mensajes de respuesta del servidor:

- Repuesta a búsqueda de servicio: se genera por el servidor después de recibir una petición de búsqueda de servicio válida.
- Repuesta a propiedad de servicio: el SDP genera una respuesta a una petición de propiedad de servicio. Ésta contiene una lista de propiedades de registro requerido.
- Repuesta de búsqueda y propiedad de servicio: como resultado se puede obtener una lista de servicios que concuerden con un patrón dado y las propiedades deseadas de estos servicios.

Algunas apreciaciones sobre el funcionamiento es que un dispositivo Bluetooth puede actuar tanto cliente SDP como servidor SDP. Otro aspecto importante es que cuando el servidor deja de estar en la zona de acción del cliente, no realiza ninguna notificación vía SDP, por lo que el cliente en este caso deducirá que no está, porque ya no recibe las respuestas del servidor ante sus continuas peticiones.

RFCOMM

El protocolo RFCOMM permite emular el funcionamiento de los puertos serie sobre el protocolo L2CAP .Se basa en el estándar ETSI TS 07.10, tomando de éste un subconjunto con las partes más relevantes y realizando algunas adaptaciones.

RFCOMM permite emular los nueve circuitos de la norma RS-232(ETIATIA-232.E).Soporta hasta 60 conexiones simultáneas entre dos dispositivos Bluetooth.

Ante una configuración RFCOMM nos encontramos básicamente con dos tipos de dispositivos:

Tipo 1: Se trata de dispositivos terminales de comunicación, como los ordenadores, las impresoras.

Tipo 2: Son aquellos que forman parte de un segmento de comunicación, como por ejemplo, los módems.

RFCOMM no hace distinción entre ambos tipos, pero el acomodarse a ellos tiene sus consecuencias en el protocolo. Por lo tanto, la transferencia de información entre dos entidades RFCOMM, se define tanto para los dispositivos tipo 1 y 2. Una parte de la información sólo se necesitará para el segundo tipo, mientras que otra se pretende

que sea usada por ambos. Debido a que un dispositivo no es consciente del tipo del otro dispositivo en el camino de comunicación, cada uno debe pasar toda la información disponible especificada por el protocolo.

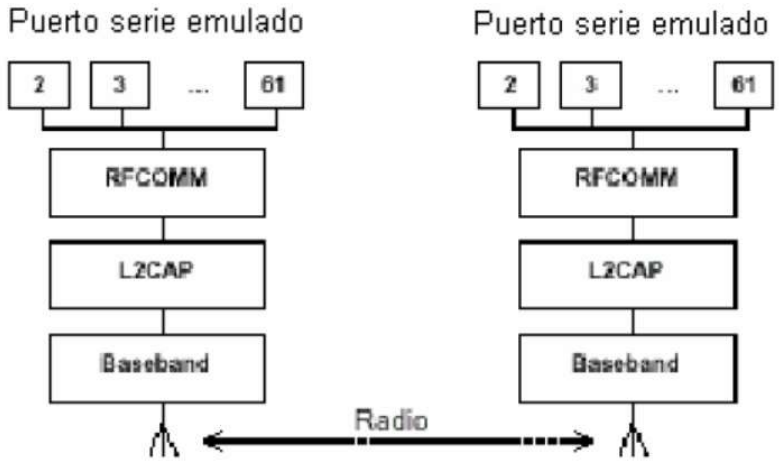
Emulación de múltiples puertos serie:

- Entre dos dispositivos:

Dos dispositivos Bluetooth usando RFCOMM en su comunicación pueden abrir múltiples puertos serie. RFCOMM soporta hasta 60 puertos emulados abiertos, aunque esto es fuertemente dependiente de la implementación específica.

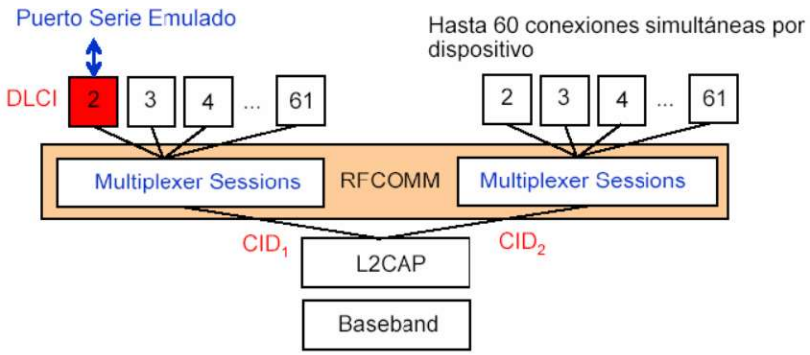
El identificador de Conexión de Datos de Enlace (DLCI) será el encargado de identificar la conexión entre una aplicación cliente y una servidora. El DLCI se representa con 6 bits pero su rango útil es [2,61], debido a que en TS 07.10 DLCI = 0 es

el canal de control dedicado, DLCI =1 no es útil y DLCI = 62-63 se encuentran reservados. El DLCI es único para una sesión RFCOMM establecida entre dos dispositivos. Para tener en cuenta que tanto la aplicación cliente como la servidora pueden residir en ambos lados de una sesión RFCOMM, con clientes en cualquier extremo realizando conexiones independientes, el valor del DLCI se divide entre los dos dispositivos que se comunican.



- Con múltiples dispositivos Bluetooth:

Si un dispositivo Bluetooth soporta emulación de múltiples puertos serie y las conexiones realizadas son con puntos terminales en diferentes dispositivos Bluetooth, entonces la entidad que implementa RFCOMM debe ser capaz de correr múltiples sesiones TS 07.10 multiplexadas, cada una con su propio identificador de canal L2CAP.



Esta capacidad de multiplexación es un elemento opcional a la hora de implementar RFCOMM, según la Especificación de Bluetooth.

4.7. Perfiles Bluetooth:

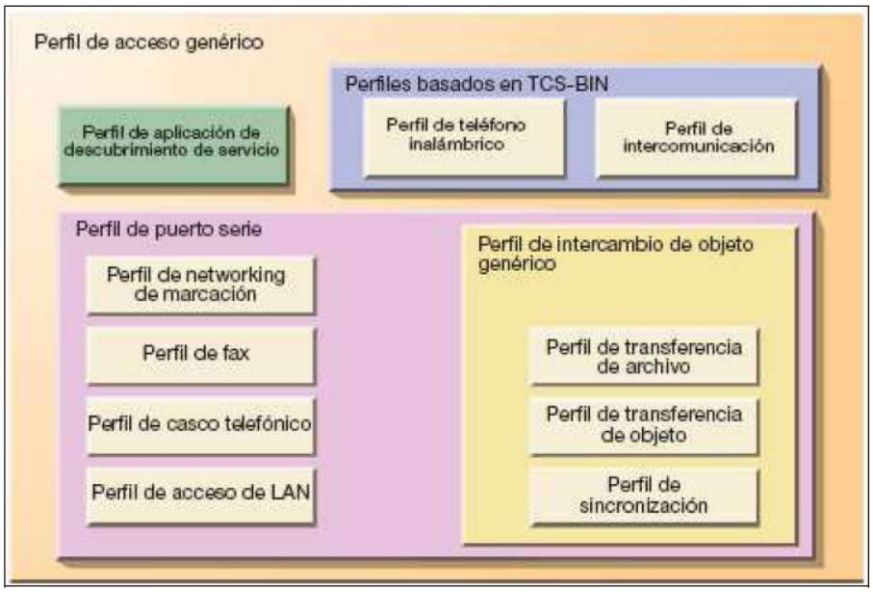
Son un conjunto de mensajes y procedimientos de la especificación Bluetooth para una situación de uso concreta del equipo. Los perfiles se encuentran asociados con las aplicaciones. Los perfiles permiten que no sea necesario implementar en un determinado dispositivo toda la pila de protocolos, sólo la parte que va a necesitar. Si el dispositivo tiene muy poca memoria y/o capacidad de procesamiento e implementamos en él toda la pila de protocolos con la carga de proceso y espacio que ello implica puede que provoquemos que el dispositivo sea totalmente ineficiente para la comunicación, por ejemplo, ratones, auriculares.

Además de la ventaja anterior, el concepto de perfil se utiliza para asegurar la interoperabilidad entre varias unidades Bluetooth que cumplan los mismos perfiles.

Cada dispositivo Bluetooth tiene al menos un perfil, es decir, una aplicación para la cual se puede utilizar el dispositivo. Cuando dos dispositivos deben ínter operar, es decir, comunicarse entre ellos, deben tener un perfil compartido. Si por ejemplo quiere transferir un archivo desde un ordenador preparado para Bluetooth a otro, ambos ordenadores deben admitir el perfil de transferencia de archivos.

Todos los dispositivos Bluetooth deben soportar el perfil de acceso genérico (Generic Access Profile) como mínimo. Este perfil en particular define el descubrimiento o hallazgo de dispositivos, procedimientos de conexión y procedimientos para varios niveles de seguridad. También se describen algunos requerimientos de interfaz al usuario. Otro perfil universal, aunque no es requerido, es el perfil de acceso a descubrimiento de servicios (Service Discovery Access Profile), el cual define los protocolos y parámetros asociados requeridos para acceder a los perfiles. Un número de perfiles han sido definidos incluyendo TCS, RFCOMM y OBEX. Algunos de estos

requieren la implementación de otros, y todos ellos requieren la implementación de perfiles genéricos.



A continuación se describen de forma breve los perfiles más utilizados en aplicaciones Bluetooth.

Perfil de acceso genérico. (GAP).

Este perfil define los procedimientos generales para el descubrimiento y establecimiento de conexión entre dispositivos Bluetooth. El GAP maneja el descubrimiento y establecimiento entre unidades que no están conectadas y asegura que cualquier par de unidades Bluetooth, sin importar su fabricante o aplicación, puedan intercambiar información a través de Bluetooth para descubrir qué tipo de aplicaciones soportan las unidades. También define aspectos relacionados con los niveles de seguridad.

Perfil de Aplicación del Descubrimiento de Servicio (SDAP).

Este define las características y los procedimientos que un dispositivo Bluetooth usa para descubrir los servicios registrados en otros dispositivos de Bluetooth y para recuperar cualquier información disponible deseada pertinente a estos servicios. El SDAP es dependiente del GAP.

Perfil de Puerto Serie. (SPP)

El Serial Port Profile (SPP) define los requerimientos específicos para posibilitar la emulación del puerto serie en los dispositivos Bluetooth usando el protocolo RFCOMM entre parejas de dispositivos

Perfil genérico de intercambio de objetos. (GOEP)

Este perfil define como los dispositivos Bluetooth deben soportar los modelos de intercambio de objetos, incluyendo el perfil de transferencia de archivos, el de carga de objetos y el de sincronización. Los dispositivos más comunes que usan este perfil son las agendas electrónicas, PDAs y teléfonos móviles.

Perfil de Teléfono Inalámbrico.

Este perfil define como un teléfono móvil puede ser usado para acceder a un servicio de telefonía de red fija a través de una estación base. Es usado para telefonía inalámbrica de hogares u oficinas pequeñas. El perfil incluye llamadas a través de una estación base, haciendo llamadas de intercomunicación directa entre dos terminales y accediendo adicionalmente a redes externas. Es usado por dispositivos que implementan el llamado “teléfono 3-en-1”

Perfil de Intercomunicación. (IP)

El Intercom Profile (IP) define los requerimientos necesarios por parte de los dispositivos Bluetooth para soportar comunicaciones entre parejas de teléfonos con soporte Bluetooth. Estos requerimientos son expresados en términos de servicios para el usuario final. Popularmente, este perfil se conoce con el nombre de "Walkie-Talkie".

Perfil de acceso telefónico a redes o perfil de networking de marcación (DNP)

Este perfil define los protocolos y procedimientos que deben ser usados por dispositivos que implementen el uso del modelo llamado Puente Internet. Este perfil es aplicado cuando un teléfono o modem es usado como un modem inalámbrico.

Perfil de FAX (FP)

El perfil Fax Profile define los requerimientos necesarios para que los dispositivos Bluetooth adquieran soporte de transferencia de Fax. Permitirá que teléfonos Bluetooth puedan enviar y recibir faxes.

Perfil de Manos Libres o de casco telefónico.

Este perfil define los requerimientos, para dispositivos Bluetooth, necesarios para soportar el uso de manos libres. En este caso el dispositivo puede ser usado como unidad de audio inalámbrico de entrada/salida. El perfil soporta comunicación segura y no segura.

Perfil de Acceso a LAN. (LAP)

El perfil de acceso a LAN define cómo pueden acceder a los servicios de una LAN mediante el protocolo PPP, los dispositivos que utilizan la tecnología inalámbrica Bluetooth. De esta manera pueden crearse puntos de acceso inalámbricos.

Perfil de Transferencia de Archivos.

Este ofrece la capacidad de transferir un archivo de un dispositivo Bluetooth a otro, inclusive permite navegar por el contenido de las carpetas del dispositivo.

Perfil de Transferencia de Objetos

Este ofrece la capacidad de transferir un objeto de un dispositivo Bluetooth a otro.

Perfil de Sincronización

Este perfil provee sincronización dispositivo a dispositivo de programas de gestión de información personal, como agendas telefónicas, calendario, mensajes y notas de información.

WiFi [12]

- Wi-Fi 6 es la iteración más reciente del protocolo de red Wi-Fi y supone una mejora sustancial respecto a su predecesor.
- Wi-Fi 6 puede ser más rápido debido a tecnologías como la priorización del tráfico, la división OFDMA y el proceso de formación de haces.
- Este nuevo protocolo también es más seguro y utiliza nuevas tecnologías de cifrado como SAE.
- Los dispositivos con tecnología Wi-Fi 6E pueden aprovechar la nueva banda de frecuencia de 6 GHz para una conectividad mejorada.
- A medida que Wi-Fi 6 se convierte en el nuevo estándar, la actualización de hardware compatible se volverá cada vez más beneficiosa.

Wi-Fi 6 promete grandes cambios a todas las redes inalámbricas del mundo.

La última década ha expuesto nuestras vidas en línea cada vez más y el internet inalámbrico ha ayudado a hacer posible esa transición. La última versión de esta tecnología es Wi-Fi 6, también conocida como 802.11ax.

El término "Wi-Fi" fue creado por Wi-fi Alliance, una organización sin fines de lucro, y hace referencia a un grupo de protocolos de redes inalámbricas que se basan en el estándar de redes IEEE 802.11. El Wi-Fi existe desde finales de los años 90, pero ha mejorado notablemente en la última década.

Generación/ Norma IEEE	Frecuencia	Velocidad Máxima del enlace	Año
Wi-Fi 6 (802.11ax)	2.4/5 GHz	600-9608 Mbit/s	2019
Wi-Fi 5 (802.11ac)	5 GHz	433-6933 Mbit/s	2014
Wi-Fi 4 (802.11n)	2.4 GHz	72-600 Mbit/s	2009

Wi-Fi 6 es una mejora importante con respecto a las generaciones anteriores, aunque a primera vista, las diferencias pueden no ser obvias para el usuario promedio. Estos cambios pueden no alterar de forma drástica el modo en que usamos routers o redes inalámbricos, sino que consisten en muchas mejoras que, combinadas, dan como resultado una actualización sustancial.

El primer gran cambio es que Wi-Fi 6 permite velocidades de conexión potencialmente más rápidas.

Velocidades más Rápidas

Un Wi-Fi más rápido implica mejores velocidades de carga y descarga (o rendimiento) debido al mayor ancho debanda que ofrece Wi-Fi 6. Esto cobra más importancia a medida que el tamaño de los archivos aumenta junto conla gran demanda de datos para la transmisión de videos de alta calidad y para los juegos en línea que requieren una continua comunicación. Transmitir a Twitch* mientras se juega un juego multijugador requiere una gran cantidad de ancho de banda, y una conexión fiable y estable.

¿Cuánto más rápido es Wi-Fi 6?

9,6 Gbps es el máximo rendimiento de Wi-Fi 6 en múltiples canales. Por el contrario, Wi-Fi 5 ofrece un máximo de 3,5 Gbps. Sin embargo, se trata de máximos teóricos: en situaciones reales, las redes locales pueden no alcanzar esta velocidad superior. Dicho esto, dado que el máximo es compartido en múltiples dispositivos, los dispositivos con Wi-Fi 6 pueden disfrutar de velocidades significativamente más rápidas incluso si no alcanzan el potencial máximo.

Las velocidades pueden ser más rápidas en comparación con el Wi-Fi 5. Esto suponiendo que usted utiliza un router Wi-Fi con un solo dispositivo. Wi-Fi 6 logra estas velocidades de transferencia de datos más elevadas gracias a diversas

técnicas, que empiezan por una codificación de datos más eficiente y un uso inteligente del espectro inalámbrico gracias a procesadores más potentes.

Wi-Fi 6 puede dar lugar a hasta un 75 % menos de latencia. Esto se logra gestionando grandes cantidades de tráfico de red de forma más eficiente. Para los jugadores de videojuegos, esto supone descargas de videojuegos más rápidas, mejores velocidades de carga para transmitir videojuegos y actividades multitarea multimedia más fiables.

Wi-Fi 6 iguala a las señales por cable e inalámbricas. Esto libera a más usuarios de las limitaciones detener que conectar el hardware a su módem. Muchos gamers o creadores de contenido siguen conectándose directamente a routers o enchufes de red mediante cables Ethernet en lugar de aprovechar la flexibilidad que ofrece una red inalámbrica. Wi-Fi 6 ayuda a reducir aún más el espacio entre conectividad por cable y conectividad inalámbrica.

La mayoría de los hogares de hoy en día tienen muchos más dispositivos con Wi-Fi que hace cinco años. Desde teléfonos inteligentes y tabletas hasta televisores y dispositivos de IoT como termostatos y timbres, casi todo se puede conectar a un enrutador inalámbrico. Wi-Fi 6 se comunica mejor con múltiples dispositivos que necesitan datos simultáneamente, y prioriza de forma más eficiente el tráfico a través de esos dispositivos. Esto hace que el Wi-Fi 6 sea más rápido.

La división de frecuencia ortogonal de acceso múltiple (OFDMA) es una de las formas en que se logra esto. OFDMA funciona al subdividir los canales en subportadoras y permitir la transmisión a múltiples puntos finales(dispositivos) al mismo tiempo. Un enrutador Wi-Fi 6 puede enviar diferentes señales en la misma ventana de transmisión. Esto da lugar a que una sola transmisión del enrutador pueda comunicarse con múltiples dispositivos, en lugar de que cada dispositivo tenga que esperar su turno mientras el enrutador envía los datos a través de la red.

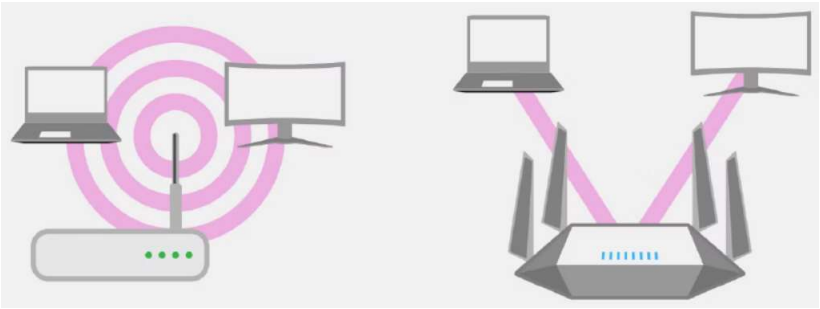
Con una red Wi-Fi tradicional, puede que los dispositivos tengan que esperar a que el cliente envíe o reciba datos en una red muy transitada. OFDMA permite que más dispositivos se sirvan de datos en la misma ventana de transmisión, lo que permite una comunicación más eficiente con múltiples dispositivos simultáneos.

Los conjuntos de servicios básicos (OBSS) son otra característica de Wi-Fi 6 que puede ayudar a mejorar la congestión de la red. Con las versiones anteriores de Wi-Fi, los dispositivos que trataban de conectarse a una red utilizaban un proceso de "escuchar antes de hablar", lo que significaba que tenían que "escuchar" cualquier ruido en un canal antes de empezar a transmitir.

Si se producía algún ruido en el canal, incluso si este empezaba en una red distante, tenían que esperar hasta que el canal estuviera despejado y luego transmitir para evitar posibles interferencias. OBSS permite que el punto de acceso utilice un "color" para identificar de forma exclusiva la red. Si se detecta otro tráfico en el canal, pero no es del mismo color que la red local, los dispositivos pueden ignorarlo y continuar la transmisión. Esto puede ayudar a aumentar la fiabilidad y a mejorar la latencia.

Al trabajar juntos, OFDMA y OBSS permiten una comunicación más eficaz en redes saturadas. Ya que cada vez más dispositivos nuestros utilizan Wi-Fi, esto ayudará a preservar la velocidad y estabilidad de nuestras conexiones.

El proceso de formación de haces es otra tecnología mejorada por Wi-Fi 6 para lograr velocidades más altas. Este método de transmisión de datos de sonido que parece tan futurista es en realidad, bastante simple. En lugar de transmitir datos hacia todas las direcciones, el enrutador detecta dónde se encuentra el dispositivo que los solicita y transmite un flujo de datos más localizado hacia esa dirección.



Los routers Wi-Fi estándar (izquierda) transmiten una señal inalámbrica en todas direcciones. El proceso de formación de haces (derecha) permite apuntar más directamente de dispositivos específicos, lo que permite una mayor velocidad de conexión.

El proceso de formación de haces no es algo nuevo para Wi-Fi 6, pero su eficacia ha sido mejorada en esta generación.

La velocidad es quizás lo más importante para el usuario promedio, especialmente para los jugadores, pero una red inalámbrica puede ofrecer muchas cosas más. Wi-Fi 6 también promete mejoras en materia de seguridad.

WPA3

El Acceso protegido Wi-Fi (WPA) es un protocolo de seguridad normal de Wi-Fi que utiliza contraseñas para el cifrado. El WPA entra en acción cada vez que se requiere

una contraseña para entrar en una red Wi-Fi. El WPA 2 fue el estándar por mucho tiempo, pero Wi-Fi 6 ha cambiado eso.

Una de las grandes mejoras es la implementación de una mayor seguridad de las contraseñas a través del sistema de intercambio de claves de Dragonfly, también llamado SAE o Autenticación simultánea de Iguales. Este método de autenticación ayuda a que las contraseñas sean más difíciles de descifrar, ya que utiliza un método más sofisticado de establecer el acuerdo de intercambio con la red Wi-Fi. Esta capa de seguridad adicional junto con un cifrado más fuerte, significa que el Wi-Fi tendrá opciones de seguridad más sólidas que nunca.

Esta capa extra de seguridad es un gran ejemplo de cómo Wi-Fi 6 cambia las cosas para mejor, sin impactar de forma negativa en la experiencia del usuario.

Vida útil de la batería y TWT

Target Wake Time (TWT) es otro desarrollo con enfoque hacia el futuro e incorporado a Wi-Fi 6, y tiene la capacidad de aumentar potencialmente la vida útil de la batería de algunos dispositivos.

Esta tecnología permite una comunicación más eficiente entre el router y el dispositivo en lo relativo a cuando este debe estar en modo de suspensión o encendido. Al comunicarse eficazmente con la radio Wi-Fi del dispositivo y activarla solo cuando necesita estar encendido, su dispositivo gastará menos tiempo y energía en la búsqueda de una señal inalámbrica. Esto puede mejorar la vida útil de la batería.

Wi-Fi 6E

Además de Wi-Fi 6, recientemente ha aparecido otra nueva tecnología: Wi-Fi 6E.

Los dispositivos Wi-Fi estaban limitados a frecuencias de 2,4 GHz y 5 GHz, pero eso ha cambiado recientemente. Los dispositivos habilitados por Wi-Fi 6E pueden utilizar la banda de frecuencia de 6 GHz, que ofrece un ancho debanda de 1200 MHz, lo que lo hace ideal para ofrecer grandes cantidades de datos en distancias más cortas. Esto puede ayudar a aliviar la congestión de tráfico y la interferencia en dispositivos compatibles. Piense en el Wi-Fi 6Ecomo un carril nuevo y ancho que se añade a la autopista Wi-Fi de dos carriles anterior, con todas las ventajas del Wi-Fi 6 incluidas.

No todas las unidades compatibles con Wi-Fi 6 son compatibles con Wi-Fi 6E, así que es necesario verificar que el hardware que está considerando es compatible con Wi-Fi 6E al actualizar.

I²C [13]

El acrónimo I2C o I²C significa *Inter Integrated Circuit*.

El bus I²C cuenta con las siguientes características:

Es un bus serie síncrono. Tiene una línea de datos y una línea por medio de la cual se establece la sincronía.

El sentido del enlace es semibidireccional, ya que la misma línea de datos transporta la información en ambos sentidos, aunque no al mismo tiempo. El bus no requiere la bidireccionalidad.

Da soporte a tecnología multipunto, ya que varios circuitos emisores pueden ser conectados, sin embargo solo uno de ellos está activo en algún momento determinado.

Un circuito puede ser emisor en un momento y en el siguiente momento puede ser receptor.

Admite topologías multimaestro. En un momento determinado un circuito gobierna el bus, en ese momento dicho circuito es el maestro, y en algún otro momento otro circuito podrá ser maestro si gana el bus, suministrando la señal de sincronía.

Posee un mecanismo de arbitraje para los casos en los que más de un maestro quiera gobernar el bus.

Cuando un maestro pierde el bus, se convierte en esclavo.

Tiene un mecanismo para adaptar la velocidad en los casos de que algún esclavo no pueda seguir la velocidad impuesta por el maestro.

La velocidad de las transferencias la marca el maestro, y puede ser variable dentro de ciertos límites. Las razones de transferencia normalmente son bajas.

Posee un esquema de direccionamiento que en cualquier momento se conoce que esclavo se dirige al maestro. Además, la dirección asignada a un circuito podrá ser modificada dinámicamente. Este mecanismo de direccionamiento deberá aceptar modificaciones en la cantidad de elementos direccionables.

El formato de las tramas es flexible con el propósito de adaptarse a la funcionalidad de circuitos añejos, actuales y futuros. Esto es, evitar la obsolescencia.

La interfaz acepta circuitos sin distinción de la tecnología empleada en su fabricación.

El *Comité I2C* asigna las *direcciones* I2C a cada tipo de circuito. De esta manera cada función implementada, independientemente del fabricante, posee la misma dirección; los circuitos que realicen funciones equivalentes deberán poseer la misma *dirección oficial* I2C independientemente del fabricante.

El protocolo del Bus I2C

El bus I2C sólo define dos señales, además del común:

SDA. Es la línea de datos serie (***S****erial* ***DA****ta*), semibidireccional. Eléctricamente se trata de una señal a colector o drenador abierto. Es gobernada por el emisor, sea éste un maestro o un esclavo.

SCL. Es la señal de sincronía (reloj serie, o ***S****erial* ***CL****ock*). Eléctricamente se trata de una señal a colector o drenador abierto. En un esclavo se trata de una entrada, mientras que en un maestro es una salida. Un maestro, además de generar la señal de sincronía suele tener la capacidad de evaluar su estado; el motivo es implementar un mecanismo de adaptación de velocidades. Esta señal es gobernada única y exclusivamente por el maestro; un esclavo sólo puede *retenerla* o *pisarla* para forzar al maestro a ralentizar su funcionamiento.

Los estados del bus I2C

El bus I2C puede encontrarse en distintos estados, que la norma define como los siguientes:

Libre. Este estado se caracteriza por encontrase las líneas SDA y SCL a 1, sin que se esté realizando ningún tipo de transferencia.

Inicio. Se produce una condición de *inicio* cuando un **maestro** inicia una transacción. Consiste en un cambio de alta a baja en la línea SDA mientras SCL permanece a alta. Esto puede apreciarse en el cronograma mostrado en la figura 6.15. A partir de que se dé una condición de inicio se considerará que el bus está ocupado y ningún otro maestro deberá intentar generar su condición de inicio.

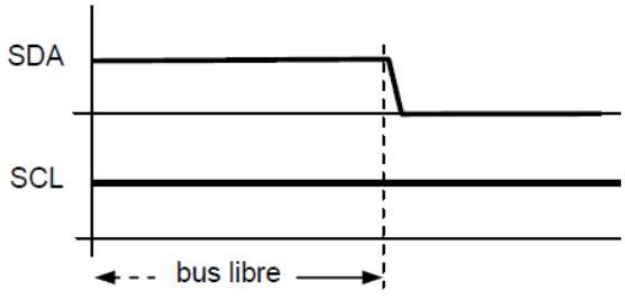


Fig. 6.15 Condición de inicio

Cambio. Se produce una condición de *cambio* cuando, estando a baja la línea SCL, la línea SDA puede cambiar de estado. En la transferencia de datos por un bus I2C éste es el único instante en el que el sistema emisor (que podrá ser tanto un maestro como un esclavo) podrá poner en la línea SDA cada bit del carácter a transmitir.

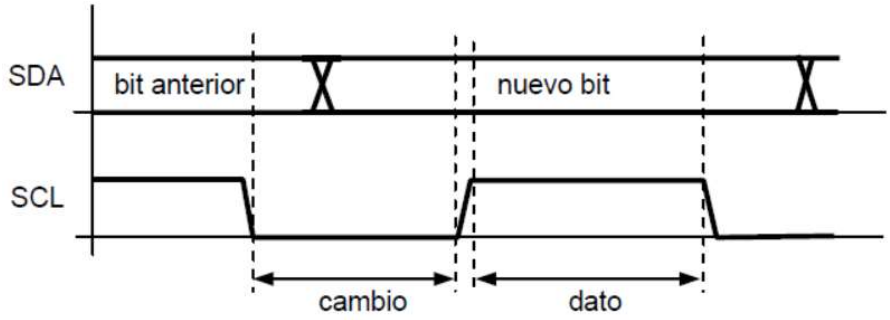


Fig. 6.16 Condiciones o estados de cambio y de dato

Dato. Este estado es el que, una vez iniciada una transacción, queda definido por la fase alta de la señal de sincronía SCL. En este estado se considera que el dato emitido es válido, y no se admite que pueda cambiar. Una vez que se inicia una transferencia, el único instante en que la línea de datos puede cambiar es en el estado de *cambio*.

Parada. Se produce una condición de *fin* o *parada* cuando, estando la línea SCL a alta, se produce un cambio de baja a alta en la línea SDA. Obsérvese que esto es una violación de la condición de *dato*, y es precisamente por esto por lo que se utiliza para que un maestro pueda indicar al esclavo que se finaliza la transferencia. Tras la condición de parada se entra automáticamente en el estado de *bus libre*.

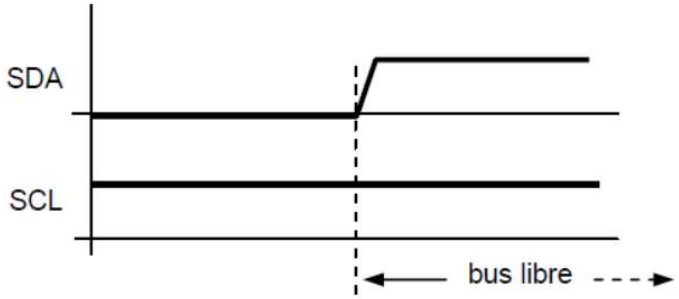


Fig. 6.17 Condición de parada

Formato de una transacción

Una *transacción* son todos los eventos desarrollados entre una condición de inicio y una de parada. Una vez producido el inicio de una transacción comienza propiamente

ésta, y en ella se lleva a cabo la transferencia de uno o varios caracteres, en uno u otro sentido.

El carácter de ocho bits es la unidad de transferencia. El orden de emisión de los bits de un carácter es empezando por el más significativo, siguiendo en orden decreciente de pesos y terminando por el menos significativo, esto es un criterio contrario al que siguen las UART.

Un carácter puede tener diversos significados en función de quién lo emita y en qué instante lo haga.

Tras cada carácter se debe producir un *reconocimiento* por parte del receptor; el motivo es dotar al protocolo de un mecanismo de seguridad.

El primer carácter transferido lo emite siempre el maestro; sus siete bits más significativos indican la dirección del esclavo al que se dirige, y el bit de menor peso indica el sentido de la transferencia de los subsiguientes caracteres (0=escritura, 1=lectura, siempre desde el punto de vista del maestro).

Dentro de una transacción es posible cambiar el sentido del flujo, pero para ello hace falta generar una nueva condición de inicio (*reinicio*) y direccionar de nuevo el mismo esclavo.

Para el direccionamiento de un esclavo se tienen siete bits, lo que daría un máximo de 128 dispositivos I2C distintos. No obstante, en la realidad se dispone de muchas menos direcciones puesto que un cierto número de ellas no se emplean como tales sino como identificadores de función especial; además, ciertos circuitos reservan para sí varias direcciones debido a que es frecuente que en un diseño se tengan que emplear varios del mismo tipo. En estos casos los circuitos tienen una dirección compuesta por dos partes; una fija establecida por el comité I2C y otra variable (bits inferiores) fijada por el diseñador de cada sistema dado. En el encapsulado de estos circuitos existen ciertas patillas mediante las cuales el diseñador puede determinar la parte *programable* de la dirección I2C. Por ejemplo, los circuitos EEPROM tienen asignada una dirección 1010XXX; 1010 es la parte fija de los siete bits, y XXX es la parte programable que corresponde al estado aplicado a tres patillas *ad hoc* de su encapsulado.

La limitación del número de direcciones trajo aparejado el que pocos años después del lanzamiento del bus I2C, y debido a su éxito, se fuesen agotando las direcciones disponibles, lo que implicaba la imposibilidad de ofrecer nuevos circuitos I2C. Sin embargo, se previó esta contingencia dado que se reservaron para futuras ampliaciones las direcciones que comenzaban por 1111. De esta manera, ahora

existen dos tipos direcciones: las normales de siete bits y las ampliadas de diez bits. En este último caso el direccionamiento de un esclavo implica la transferencia de dos caracteres; el primero deberá comenzar por 11110XX, siendo XX los dos bits de mayor peso de la dirección de diez bits, y el segundo carácter corresponderá a los ocho bits inferiores de la dirección del esclavo. El sentido de la transacción se sigue indicando por el bit menos significativo del primer carácter de dirección (11110-XX-L/E).

Una transacción obedecerá a alguna de las siguientes estructuras:

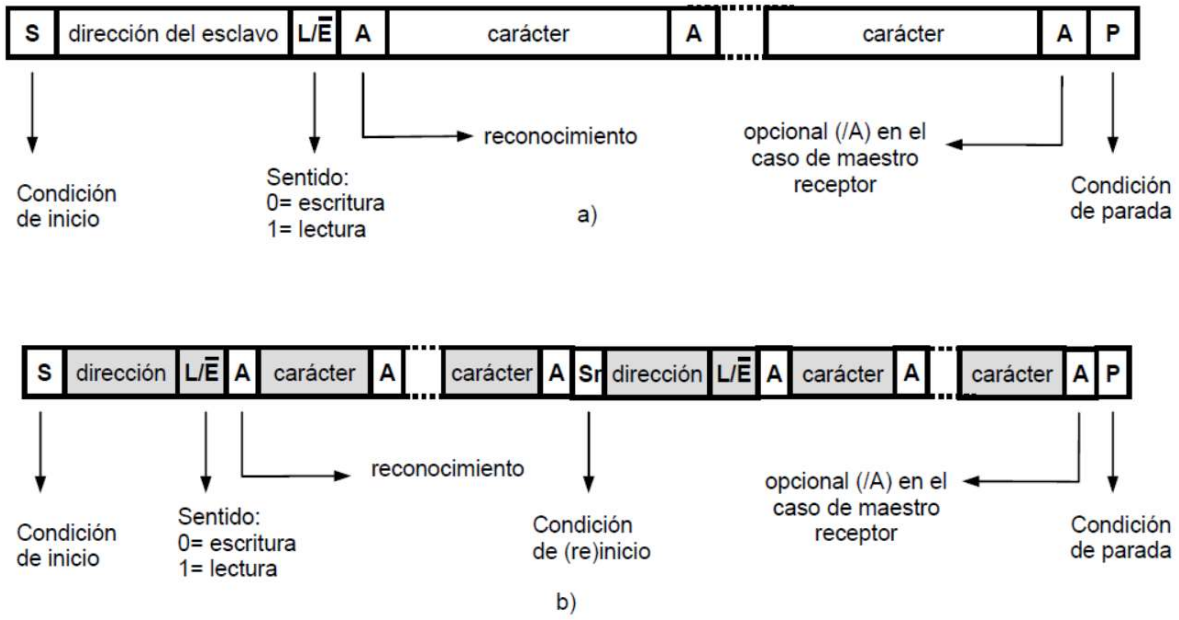


Fig. 6.18 Formatos de una transacción: a) sin cambio de sentido b) con redireccionamiento y posible cambio de sentido

En la figura 6.18 se ha ilustrado el direccionamiento con direcciones de siete bits; si se utilizase un direccionamiento de diez bits entonces la parte del direccionamiento quedaría como se muestra en la figura 6.19.

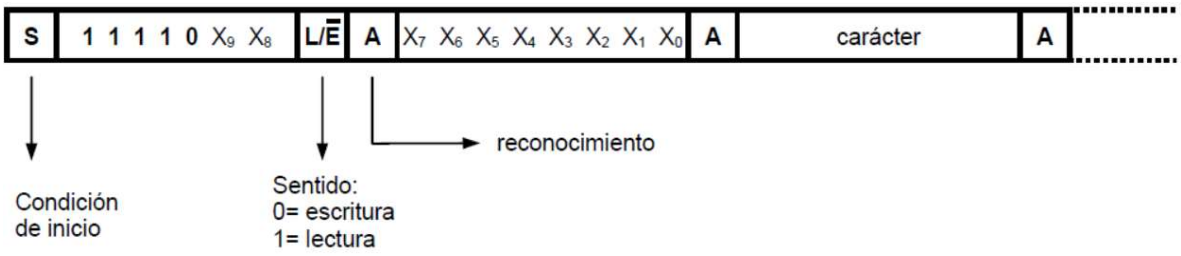


Fig. 6.20 Direccionamiento ampliado, de diez bits

En la figura 6.20 se indica por X9…X0 los diez bits de la dirección del esclavo al que se dirige el maestro.

El gobierno de la señal de sincronía

La señal SCL sólo la puede generar el maestro. Si el maestro actúa como emisor entonces la señal de sincronía puede entenderse como una validación de los bits que va volcando en la línea de datos SDA, sean estos bits los de un carácter de dirección o los de un carácter propiamente de datos. Si actúa como receptor entonces SCL sirve no sólo para indicar los instante en los que el maestro toma los bits que le envía el esclavo, sino también para –indirectamente– marcar la velocidad de envío del siguiente bit (el esclavo no debe poner el siguiente bit hasta que se dé una condición de *cambio*, y ésta no se da hasta que finaliza el estado actual de *dato* (esto es, hasta que SCL pasa de alta a baja).

La unidad de transferencia es el carácter de ocho bits, y que a cada carácter le debe seguir un bit de reconocimiento por parte del receptor. Será siempre el maestro quien validará el bit de reconocimiento, sea éste ofrecido por él mismo (maestro receptor) o por el esclavo (esclavo receptor).

Finalmente, sólo hay una situación en la que un esclavo puede *interferir* la señal SCL gobernada por el maestro. Se trata del mecanismo denominado *sincronización*, por medio del cual un esclavo puede ralentizar la velocidad de transferencia marcada por el maestro. En este caso el esclavo no gobierna el reloj pero interfiere con cierto criterio su estado para que el maestro advierta que se le está pidiendo que vaya más lento.

Necesidad del reconocimiento

El establecer la necesidad de que el receptor reconozca cada carácter transmitido obedece al deseo de establecer un mecanismo de seguridad en las transferencias, pero también a algo más.

El reconocimiento es **obligatorio** para todo esclavo receptor. Si uno no lo hace entonces el maestro emisor considerará que se ha producido un fallo en el bus y deberá establecer los mecanismos oportunos de respuesta a este fallo. Dependiendo de en qué circunstancias se produzca esta ausencia de reconocimiento entonces se podrá obrar de manera distinta. Por ejemplo:

- Si la ausencia de reconocimiento del esclavo receptor se produce en la fase de direccionamiento esto podrá significar varias cosas:

  Que el circuito se encuentra ausente (removido del bus, o apagado).

Que el circuito está ocupado realizando algún tipo de tarea interna que le impide responder al direccionamiento del maestro. Esto podrá tener sentido en algunos dispositivos I2C, pero no en otros. Por ejemplo, en las memorias EEPROM con bus I2C si se les escribe un bloque de datos se tarda un cierto tiempo en llevar a cabo tal proceso, durante el cual no dará el reconocimiento si es vuelta a direccionar para realizar un nuevo acceso.

Que el circuito está averiado o ha sucedido alguna anomalía en la línea.

- Pero si esta ausencia se produce en medio de una transferencia de datos entonces esto normalmente será síntoma de una avería funcional (a menos que dé la casualidad de que en ese instante ha sido removido o apagado).

El tipo de medida a tomar por parte del maestro dependerá de cada circunstancia, y podrá ir desde volver a intentar direccionar el esclavo un poco más adelante hasta generar algún tipo de alarma.

El reconocimiento consiste en poner la línea de datos SDA a baja durante el impulso SCL que sigue al del último bit de un carácter; y esto lo debe realizar el receptor.

En el instante que toca el reconocimiento el emisor deberá aislar del bus su línea SDA; en los bits de los caracteres (estados de datos) es él quien la gobierna –no se olvide que actúa como emisor– pero no sucede tal cosa en el reconocimiento. Aquí quien ha de gobernar ahora SDA es el receptor, para dar el reconocimiento. Por ello, si el emisor no aísla del bus su SDA local, y diese la casualidad de que el bit menos significativo, y último del carácter transmitido, fuese un 0, entonces podría suceder que si el receptor no diese su reconocimiento el maestro no advertiría tal eventualidad puesto que al sondear el estado de la línea SDA detectaría que está a baja (por el último bit de dato transmitido) e interpretaría erróneamente que ha habido el reconocimiento. En la figura 5.21 se aprecia lo expresado.

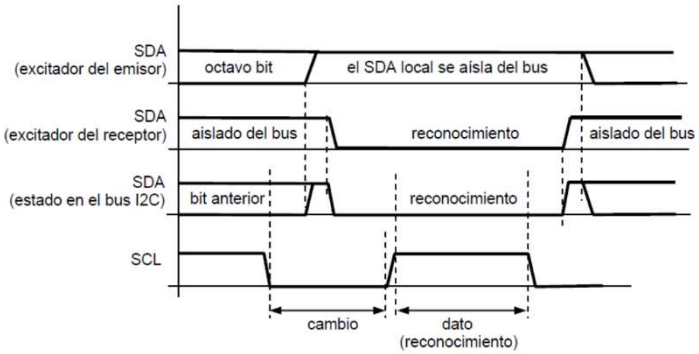


Fig. 5.21 El reconocimiento en el bus I2C

En la figura se ha mostrado el estado de los excitadores SDA del emisor y del receptor I2C. Recuerde que la capa física de la interfaz I2C necesita la existencia de un excitador y de una electrónica receptora tanto para SDA como para SCL. Cuando alguien actúa como receptor, debe aislar del bus su excitador SDA (ofrecer un 1 lógico), y también SCL si es un esclavo, para que el excitador del emisor sea quien gobierne la línea SDA del bus. Pero cuando llega la hora de dar el reconocimiento entonces es el excitador del emisor quien debe aislarse para que, ahora sí, sea el del receptor quien aplique un nivel a baja en SDA. De esta manera, si se observa la figura 6.21 puede verse que la señal en la línea SDA del bus I2C es el producto lógico de las señales aplicadas por los excitadores de todos y cada uno de los circuitos conectados al bus. Ésta es una propiedad de los excitadores con salida a colector o drenador abierto.

Aunque se ha dicho que el reconocimiento es obligatorio por parte del receptor, **existe una excepción a esta regla: cuando el receptor es un maestro entonces es posible no dar el reconocimiento al último carácter**, justo antes de generar la condición de parada. Para entender el motivo de esto es necesario recordar que es el maestro quien inicia una transacción, la gobierna y la finaliza. Por tanto, aunque un maestro actúe como receptor sabrá perfectamente cuándo no quedan más datos por recibir. Dicho de otra manera, un esclavo emisor si tiene algo que enviar no lo hará por propia iniciativa sino porque el maestro receptor se lo solicite, y en este caso el maestro receptor sabrá perfectamente cuántos datos desea tomar. Supóngase ahora un caso concreto en el que se desea leer un bloque de siete datos de una memoria EEPROM. Podrá pensarse que esta circunstancia no planteará mayor problema dado que bastará, al recibirse el séptimo carácter, reconocerlo y generar entonces una condición de parada para poder iniciar una nueva transacción con otro circuito I2C. Pues bien, si el maestro receptor reconociese el carácter antes de generar la condición de parada podría suceder que tal condición de parada no se llegase a producir efectivamente en el bus I2C. El motivo es el siguiente: cuando el maestro receptor da su reconocimiento al carácter enviado por el esclavo emisor, éste supondrá que a continuación el maestro le solicitará el primer bit del siguiente carácter, el octavo, dado que no tiene por qué saber cuántos datos le solicitará el maestro. Por ello el esclavo, cuando SCL pase a baja y se produzca un estado de cambio, colocará en la línea SDA el bit más significativo del siguiente dato, y esperará el impulso SCL del maestro. Supongamos que este bit da la casualidad de que es un 0 lógico. ¿Qué sucederá entonces? Pues que el maestro receptor, que desea finalizar la transacción y para ello intenta generar una condición de parada, se encontrará con que no puede hacerlo porque el esclavo emisor no ha liberado la línea SDA del bus.

Esto puede apreciarse en la figura 6.22. Es por este motivo por el que es opcional el último reconocimiento de un maestro receptor; como el esclavo emisor **obligatoriamente** tiene que liberar la línea SDA para permitir que el maestro receptor reconozca, pues entonces éste puede, si así lo desea, aprovechar este lapso para tener la garantía de poder generar una condición de parada.

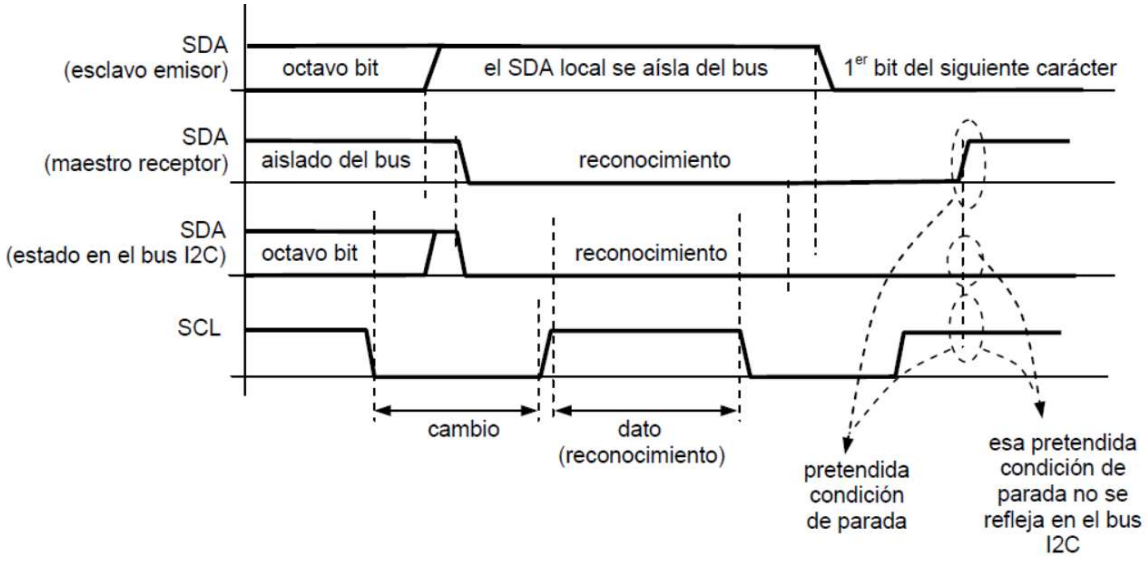


Fig. 6.22 El problema de una parada tras un reconocimiento

Cuando el receptor es un maestro también puede intencionadamente no producirse el reconocimiento en medio de una transacción. Esto se haría para abortarla ante situaciones que hacen aconsejable tal cosa. Supóngase que en medio de una transacción, al leerse un carácter cualquiera de una trama, sucede un evento que hace que el maestro opte por abortar la transacción e iniciar otra con otro dispositivo I2C. Ya se ha visto que si se reconoce ese carácter podría suceder que el maestro no pudiera generar la condición de parada y por tanto no podría iniciar una nueva transacción con el nuevo circuito I2C con el que ahora urge comunicarse. Al no reconocer, se aprovecha este instante para forzar una condición de parada que aborte la transacción en curso, que podrá reanudarse más adelante.

Es necesario advertir que la manera ortodoxa de que un maestro receptor realice con garantía la condición de parada es no aprovechar el instante del reconocimiento sino el siguiente tiempo de bit. En este caso el maestro receptor no reconoce pero aplica el correspondiente impulso de sincronía. Cuando un esclavo emisor detecta que no se ha reconocido el dato enviado, la norma I2C obliga a que justo a continuación aísle del bus su SDA. De esta manera ya el maestro receptor está en condiciones de generar sin problemas su condición de parada. Este segundo método es el que especifica la norma, pero el que se ha comentado en primer lugar resulta igualmente

efectivo y más rápido de llevar a cabo (la única precaución que hay que tener es conocer cómo se implementa la interfaz I2C en los esclavos utilizados; si una condición de parada se detecta de manera cableada entonces no habrá problema, pero si se hace por programa entonces se deberá verificar que se comprueba en cualquier instante y no sólo tras cada reconocimiento). Lo comentado puede observarse en la figura 6.23.

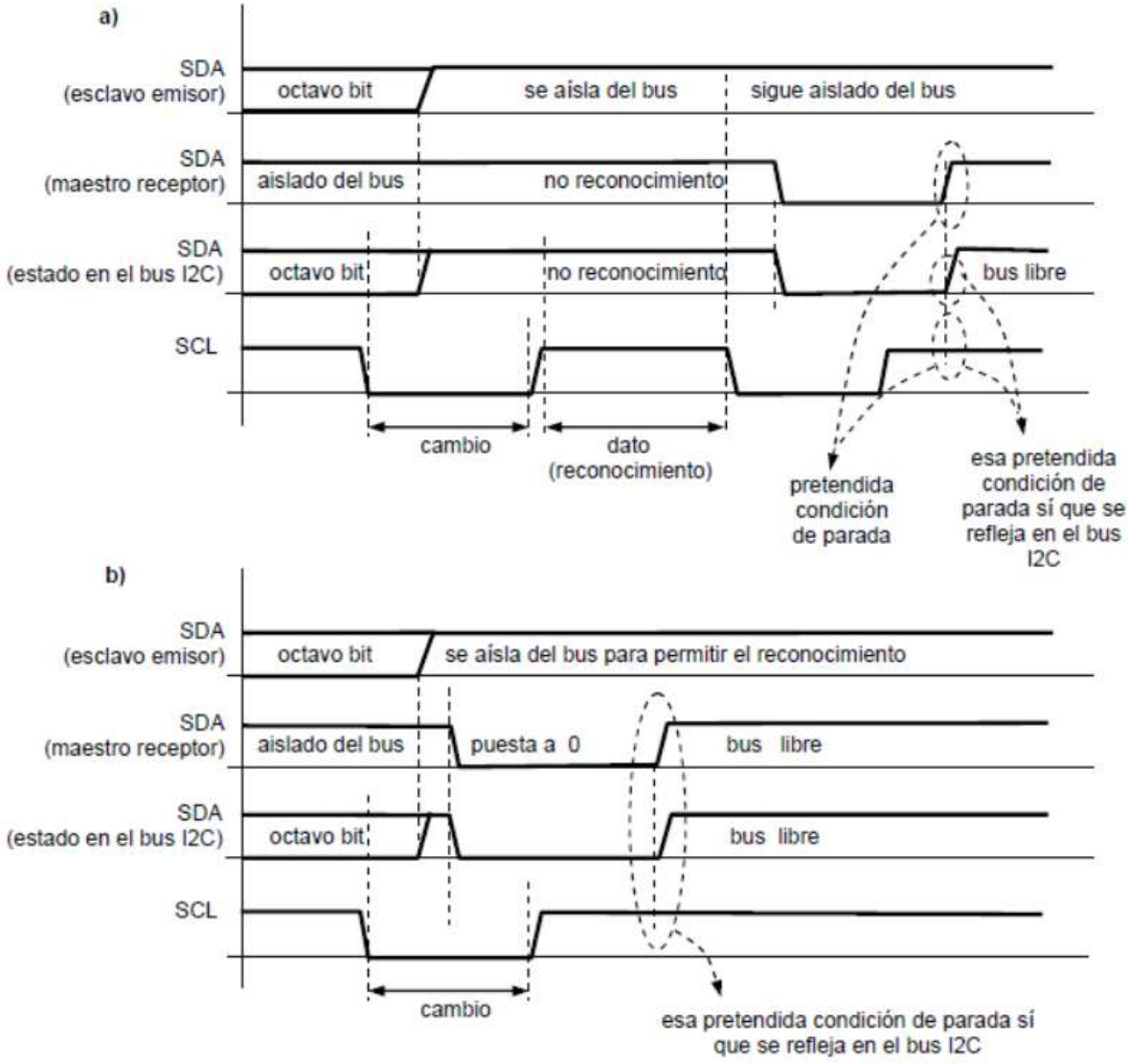


Fig. 6.23 Terminación sin reconocimiento: a) modo normal b) modo abreviado

6.3.Envío y recepción de datos a través de un puerto de comunicación. [14]

Se tiene un PLC Siemens S7-1200 para el cual se prueban los programas siguientes empleando SCL de Siemens.

Activar_Desactivar una Salida con una Entrada

```
IF “Entrada0”=1 THEN
   “Salida0”:=1;
END_IF;
```

Control de una Sali2da mediante Set/Reset

```
IF “Entrada1”=true THEN
   “Salida1:=true;
END_IF;
```

Asociación serie (AND) de dos variables de entrada

```
//Activación de una salida mediante dos entradas en serie
IF “Entrada3” AND “Entrada4”=true THEN
“Salida3”:=true;
ELSE “salida3”:=FALSE;
END_IF;
```

```
//Calculo del producto matricial con Arduino
const int ren=3, col=3;
int A[ren][col]={0,1,2,3,4,5,6,7,8},
     B[ren][col]={8,7,6,5,4,3,2,1,0},
     C[ren][col];
int rg[col], co[ren];
int j;
for(j=0;j<col;j++)
rg[j]=A[r][j];
```

```
}

void columna(int c) {
int i;
for(i=0;i<ren;i++)
co[i]=B[i][C];
}
void muestra_Matriz(void){
int i, j;
for(i=0;i<ren;i++){
for(j=0;j<col;j++){
Serial.print(C[i][j]);
Serial.print('\t');
}
Serial.println(".");
}
}

void multiplica(void){
int i,j,m,s;
for(i=0;i<ren;i++){
renglón(i);
for(j=0;j<col;j++){
s=0;
columna(j);
for(m=0;m<ren;m++)
```

```
s=s+rg[m]*co[m];
C[i][j]=s;}
}
void setup(){
  Serial.begin(9600);
}

void loop(){
int i,j;
for(i=0;i<ren;i++){
for(j=0;j<col;j++){
C[i][j]=A[i][j];}}
Serial.println(“Mostrando Matriz A”);
muestra_Matriz();

for(i=0;i<ren;i++){
for(j=0;j<col;j++){
C[i][j]=B[i][j];}}
Serial.println(“Mostrando Matriz B”);
muestra_Matriz();

multiplica();
Serial.println(“El producto C=A*B”);
muestra_Matriz();
delay(2000);
```

Referencias

[1] Diza Online, https://dizaonline.com/blog/tecnologia/importancia-de-la-computacion/

[2] Universidad Galileo, https://www.galileo.edu/noticias/la-importancia-de-la-programacion-en-la-actualidad-un-pilar-en-la-revolucion-industrial-digital/#:~:text=La%20programaci%C3%B3n%20es%20la%20base,de%20las%20cosas%20(IoT).

[3] H. Shildt, *C The Complete Reference*, Osborne/McGraw-Hill, 4th Ed., USA, 2000.

[4] G. Leblanc, *Turbo C para IBM-PC y Compatibles*, Editorial Gustavo Gili, 1988.

[5] O. Cairó, *Fundamentos de programación. Piensa en C*, Ed. Pearson, 2006.

[6] F. Menchaca, *Fundamentos de programación en lenguaje C,* 1ª Ed., Instituto Politécnico Nacional,1999.

[7] Universitat Politecnica de Valencia, https://www.disca.upv.es/aperles/asignatures/antic/sii/comunicaciones_serie_RS_232.pdf.

[8] Universidad Nacional del Sur-Bahia Blanca, Clase 10: RS232, RS485 y USB Sistemas Embebidos, Prof: Lic. José H. Moyano, Departamento de Ciencias e Ingeniería de la Computación, 2019.

[9] Intel, ¿Qué es la tecnología Bluetooth®?, visitada 10/12/2024

[10] Intel, ¿Cómo funciona la tecnología Bluetooth®?, visitada 10/12/2024.

[11] Intel, Tecnología Bluetooth, visitada 10/12/2024.

[12] Intel, ¿Que es Wi-Fi 6?, visitada 10/12/2024.

[13] A. Moreno, *EL BUS I2C,* Universidad de Córdoba (España). Depto. Arquitectura de Computadores, E. y T. Electrónica, 2004.

[14] E. Camacho, *Programación Básica del PLC Siemens S7-1200*, 2023.

INSTITUTO TECNOLÓGICO DE SAN LUIS POTOSÍ

VERIFICACIÓN DE INSTRUMENTACIÓN DIDÁCTICA COLEGIADA

Para el llenado de este formato considere la información de la cédula 3.3.2 de CACEI, el programa de la asignatura y la información que se solicita en el SII2. Esta información debe capturarse en el sistema por un solo docente y podrá ser exportada por los demás. Se pueden realizar ajustes a esta instrumentación cuando los docentes así lo consideren, pero al menos se debe hacer una revisión anual. Es importante indicar que estrategias de evaluación miden la evaluación del logro del atributo o atributos a los que contribuye la materia.
Recuerde que la información que se defina en este formato es la misma que se debe definir en los formatos mencionados en el párrafo anterior y por lo tanto servirá para el llenado de los mismos. Es además la evidencia del trabajo colegiado ante el Sistema de Gestión Integral.

DATOS GENERALES

Carrera	**Ingeniería Eléctrica**
Nombre de la Asignatura	**Programación**
Clave de la Asignatura (Programa de estudio)	**ELC-1022**
Fecha de Revisión	**09/05/2022**
¿Existe un acuerdo de Academia para una adecuación al programa?	○ **Si** ● **No** **Folio del acuerdo:**
Atributos de Egreso a los que contribuye la materia en orden de importancia de mayor a menor y de acuerdo a la cédula 3.3.2	Atributo 1 Nivel de contribución: Medio Resuelve problemas aplicando CB e Ingeniería Atributo 5 Nivel de contribución: Medio Responsabilidades éticas y profesionales Agregar más atributos si así se considera

ACUERDOS POR UNIDAD

UNIDAD 1
Fundamentos de programación

Tiempo: 10 horas

Actividades de enseñanza (métodos, técnicas y ambientes de aprendizaje)

- Solicita al estudiante investigar en diferentes fuentes de información los lenguajes de programación y su clasificación.
- Solicita que en equipos de trabajo se exponga frente a grupo los resultados de la investigación previamente realizada.
- Propicia el desarrollo de algoritmos a partir de problemas perfectamente delimitados usando pseudocódigo y diagramas de flujo.
- Elabora el diagrama de flujo y realiza la simulación del algoritmo mediante software y pide a los estudiantes sigan el procedimiento.
- Explica la importancia del compilador en los lenguajes de alto nivel.
- Explica los lenguajes orientados al cálculo numérico que apoyan al campo de la ingeniería.

Actividades de aprendizaje (métodos, técnicas e instrumentos) por unidad

- Investigar en diferentes fuentes de información los lenguajes de programación y su clasificación.
- Exponer frente a grupo los resultados de la investigación previamente realizada.
- Desarrollar algoritmos a partir de problemas perfectamente delimitados usando pseudocódigo y diagramas de flujo.
- Simular el algoritmo mediante diagramas de flujo, empleando software.
- Investigar la importancia del compilador en los lenguajes de alto nivel.
- Investiga los lenguajes orientados al cálculo numérico que apoyan al campo de la ingeniería.

Indicadores de alcance (agregue o quite indicadores según corresponda)

A 100 %

Proporciona una introducción a la programación

B %

C %

D %

E %

F **%**

MATRIZ DE EVALUACIÓN (Solamente es necesario llenar esta matriz en forma colegiada si contiene las evidencias de aprendizaje que contribuyen a los Atributos de Egreso)

Evidencia de aprendizaje	Total %	Indicador de alcance						Evaluación formativa de la competencia (instrumento de evaluación)	Nombre abreviado del **AE** del **PE** al que contribuye **Si aplica**
		A	B	C	D	E	F		
Examen	60	60						Examen	
Prácticas	40	40						Rúbrica	
Total	100	100							

Principales prácticas/proyecto (de acuerdo a cédula 3.3.2)

- Manejo del entorno de programación.
- Desarrollar un programa que comprenda la estructura básica del lenguaje.

UNIDAD 2
Elementos del lenguaje de programación

Tiempo: 10 horas

Actividades de enseñanza (métodos, técnicas y ambientes de aprendizaje)

- Dar las instrucciones al estudiante para que se familiarice con el entorno de programación.
- Explicar a los estudiantes los conceptos de las palabras reservadas del lenguaje específico a utilizar.
- Explicar y desarrollar programas que utilicen variables, constantes y los diferentes tipos de datos
- Explicar y desarrollar programas que utilicen los operadores aritméticos, lógicos y relacionales.
- Explicar y desarrollar programas que utilicen la mayoría de los puntos vistos en la estructura básica de un programa con la finalidad de dar solución a problemas cotidianos.

Actividades de aprendizaje (métodos, técnicas e instrumentos) por unidad

- Familiarizarse con el entorno de programación por medio del diseño, edición y compilación de programas sencillos en un lenguaje de programación orientado al cálculo numérico.
- Investigar cuales son las palabras reservadas del lenguaje específico a utilizar.
- Desarrollar programas que utilicen variables, constantes y los diferentes tipos de datos
- Desarrollar programas que utilicen los operadores aritméticos, lógicos y relacionales.
- Desarrollar programas que utilicen la mayoría de los puntos vistos en la estructura básica de un programa con la finalidad de dar solución a problemas cotidianos.

Indicadores de alcance (agregue o quite indicadores según corresponda)

A 100 %

Explica el entorno de programación.

B %

C %

D %

E %

F %

MATRIZ DE EVALUACIÓN (Solamente es necesario llenar esta matriz en forma colegiada si contiene las evidencias de aprendizaje que contribuyen a los Atributos de Egreso)

Evidencia de aprendizaje	Total %	Indicador de alcance						Evaluación formativa de la competencia (instrumento de evaluación)	Nombre abreviado del **AE** del **PE** al que contribuye **Si aplica**
		A	B	C	D	E	F		
Examen	60	60						Examen	
Prácticas	40	40						Rúbrica	

Total	100	100							

Principales prácticas/proyecto (de acuerdo a cédula 3.3.2)

- Desarrollar un programa que muestre el valor almacenado en una constante y en variables de los tipos de datos básicos.
- Escribir un pro un programa que desglose cierta cantidad de segundos introducida por teclado en su equivalente en semanas, días, horas, minutos y segundos.

UNIDAD 3
Arreglos

Tiempo: 10 horas

Actividades de enseñanza (métodos, técnicas y ambientes de aprendizaje)

- Dar a conocer el uso de arreglos en la programación estructurada y su relación con la electrónica en semestres posteriores.
- Explicar y desarrollar programas que utilicen arreglos unidimensionales y multidimensionales con ejemplos didácticos para compresión de la lectura y escritura, y modificación de datos.
- Explicar y desarrollar programas con arreglos que permitan operaciones entre ellos.

Actividades de aprendizaje (métodos, técnicas e instrumentos) por unidad

- Define la importancia del uso de arreglos en la programación estructurada y su relación con la electrónica en semestres posteriores.
- Desarrolla programas que utilicen arreglos unidimensionales y multidimensionales con ejemplos didácticos para compresión de la lectura y escritura, y modificación de datos.
- Desarrolla programas con arreglos que permitan operaciones entre ellos.

Indicadores de alcance (agregue o quite indicadores según corresponda)

A 100 %

Aplica arreglos en la solución de problemas de ingeniería.

B %

C %

D %

E **%**

F **%**

MATRIZ DE EVALUACIÓN (Solamente es necesario llenar esta matriz en forma colegiada si contiene las evidencias de aprendizaje que contribuyen a los Atributos de Egreso)

Evidencia de aprendizaje	Total %	Indicador de alcance						Evaluación formativa de la competencia (instrumento de evaluación)	Nombre abreviado del **AE** del **PE** al que contribuye **Si aplica**
		A	B	C	D	E	F		
Examen	60	60						Examen	
Prácticas	40	40						Rúbrica	
Total	100	100							

Principales prácticas/proyecto (de acuerdo a cédula 3.3.2)

- Desarrollar un programa que muestre un menú para la realización de operaciones con matrices.
- Realizar un programa que demuestre el uso de las operaciones con arreglos en C.

UNIDAD 4
Estructuras de control

Tiempo: 14 horas

Actividades de enseñanza (métodos, técnicas y ambientes de aprendizaje)

- Explicar y desarrollar programas que utilicen estructuras de selección, repetición y múltiple selección.
- Diseñar algoritmos más complejos donde se vincule los conocimientos vistos en la unidad anterior.

Actividades de aprendizaje (métodos, técnicas e instrumentos) por unidad

- Desarrollar programas que utilicen estructuras de selección, repetición y múltiple selección.
- Diseñar algoritmos más complejos donde se vincule los conocimientos vistos en la unidad anterior.

INSTITUTO TECNOLÓGICO DE SAN LUIS POTOSÍ

VERIFICACIÓN DE INSTRUMENTACIÓN DIDÁCTICA COLEGIADA

Indicadores de alcance (agregue o quite indicadores según corresponda)

A 100 %

Desarrolla algoritmos utilizando estructuras de selección y repetición.

B %

C %

D %

E %

F %

MATRIZ DE EVALUACIÓN (Solamente es necesario llenar esta matriz en forma colegiada si contiene las evidencias de aprendizaje que contribuyen a los Atributos de Egreso)

Evidencia de aprendizaje	Total %	Indicador de alcance						Evaluación formativa de la competencia (instrumento de evaluación)	Nombre abreviado del **AE** del **PE** al que contribuye **Si aplica**
		A	B	C	D	E	F		
Examen	60	60						Examen	Resuelve problemas aplicando CB e Ingeniería
Prácticas	40	40						Rúbrica	
Total	100	100							

Principales prácticas/proyecto (de acuerdo a cédula 3.3.2)

- Desarrollar un programa que calcule el área, perímetro y diámetro de un círculo a partir de la declaración de una constante (pi) y la asignación de valor del radio. Realizarlo con diferentes valores.
- Desarrollar un programa que a partir de un rango de años que obtenga los que son bisiestos.

UNIDAD 5
Funciones

Tiempo: 10 horas

Actividades de enseñanza (métodos, técnicas y ambientes de aprendizaje)

- Describe los elementos que conforman una función, estos términos dependiendo del lenguaje a utilizar.
- Desarrollar los programas previamente elaborados y explica cómo crear funciones a partir de ellos.
- Explica la metodología para llamar las funciones en un programa principal, observando las ventajas y desventajas de este tipo de programación.
- Explicar y desarrollar programas que utilicen bibliotecas de funciones de entrada y salida, de manejo de archivos y manipulación de cadenas.

Actividades de aprendizaje (métodos, técnicas e instrumentos) por unidad

- Investigar los elementos que conforman una función o método o subfunción, estos términos dependiendo del lenguaje a utilizar.
- Rediseñar los programas previamente elaborados y crear funciones a partir de ellos.
- Los programas ya diseñados con funciones deberán ser invocados en un programa principal, observando las ventajas y desventajas de este tipo de programación.
- Desarrollar programas que utilicen bibliotecas de funciones de entrada y salida, de manejo de archivos y manipulación de cadenas.

Indicadores de alcance (agregue o quite indicadores según corresponda)

A 100 %

Introduce las funciones para optimizar la ejecución del programa.

B %

C %

D %

E **%**

F **%**

MATRIZ DE EVALUACIÓN (Solamente es necesario llenar esta matriz en forma colegiada si contiene las evidencias de aprendizaje que contribuyen a los Atributos de Egreso)

Evidencia de aprendizaje	Total %	Indicador de alcance						Evaluación formativa de la competencia (instrumento de evaluación)	Nombre abreviado del **AE** del **PE** al que contribuye **Si aplica**
		A	B	C	D	E	F		
Examen	60	60						Examen	
Prácticas	40	40						Rúbrica	
Total	100	50							

Principales prácticas/proyecto (de acuerdo a cédula 3.3.2)

- Desarrollar un programa que utilice una función que calcule el factorial de un número.

UNIDAD 6
Uso de puertos de comunicación

Tiempo: 10 horas

Actividades de enseñanza (métodos, técnicas y ambientes de aprendizaje)

- Dar a conocer los diferentes tipos de puertos de comunicación.
- Explicar la interacción de estos puertos con los sistemas electrónicos.
- Desarrollar y explicar programas que permitan interactuar con sistemas electrónicos externos a la computadora (motores, leds, actuadores, sensores, etc.).

Actividades de aprendizaje (métodos, técnicas e instrumentos) por unidad

- Investigar y exponer frente a grupo los diferentes tipos de puertos de comunicación, dividiéndolos en dos grupos: seriales y paralelos.
- Comprender la interacción de estos puertos con los sistemas electrónicos.

- Desarrollar programas que permitan interactuar con sistemas electrónicos externos a la computadora (motores, leds, actuadores, sensores, etc.).

Indicadores de alcance (agregue o quite indicadores según corresponda)

A 100 %

Conoce los diferentes puertos de comunicación.

B %

C %

D %

E %

F %

MATRIZ DE EVALUACIÓN (Solamente es necesario llenar esta matriz en forma colegiada si contiene las evidencias de aprendizaje que contribuyen a los Atributos de Egreso)

Evidencia de aprendizaje	Total %	Indicador de alcance						Evaluación formativa de la competencia (instrumento de evaluación)	Nombre abreviado del **AE** del **PE** al que contribuye **Si aplica**
		A	B	C	D	E	F		
Examen	60	60						Examen	
Prácticas	40	40						Rúbrica	Responsabilidades éticas y profesionales.
Total	100	100							

Principales prácticas/proyecto (de acuerdo a cédula 3.3.2)

- Desarrollar un programa que realice una animación de led´s.

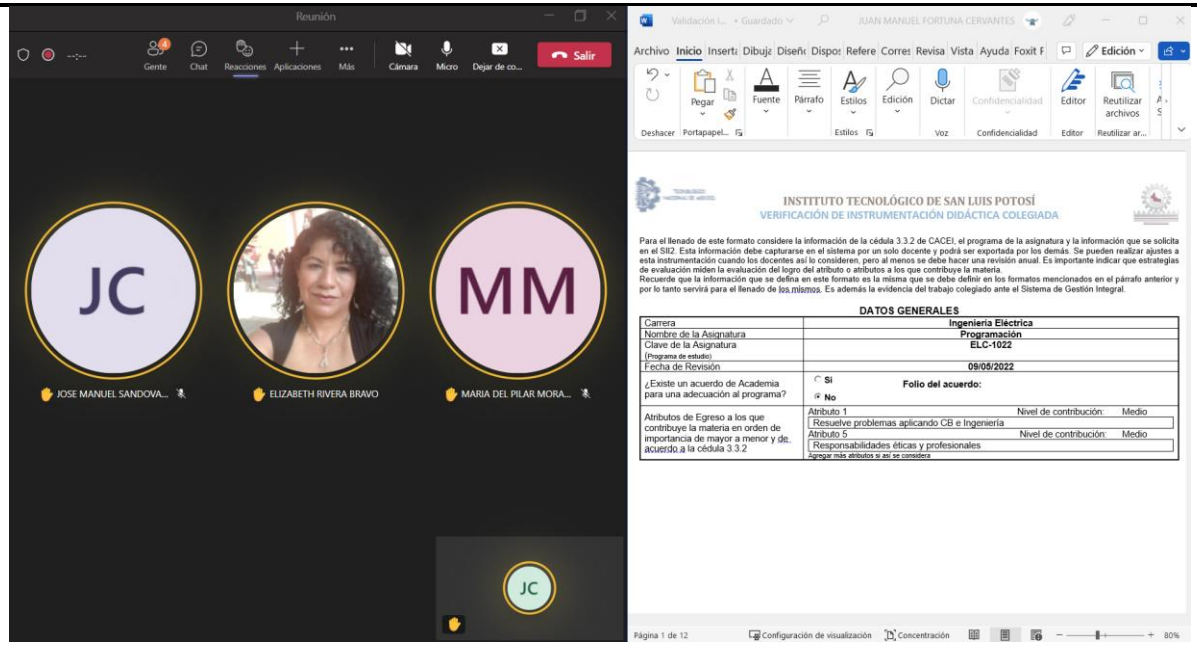
Si requiere más unidades puede copiar y pegar la tabla de unidad

FIRMAS DE LOS DOCENTES

Nombre	Firma
Juan Manuel Fortuna Cervantes	
Elizabeth Rivera Bravo	
José Manuel Sandoval Cancino	
María del Pilar Morales Morelos	

INSTITUTO TECNOLÓGICO DE SAN LUIS POTOSÍ
VERIFICACIÓN DE INSTRUMENTACIÓN DIDÁCTICA COLEGIADA

IMAGEN O LIGA DE LA REUNIÓN



REVISIÓN

Presidente de Academia	Jefe del Departamento
M.E.H. Irene Zapata Silva	Ing. Norma Orocio Castro
Vo.Bo.	**Vo.Bo.**