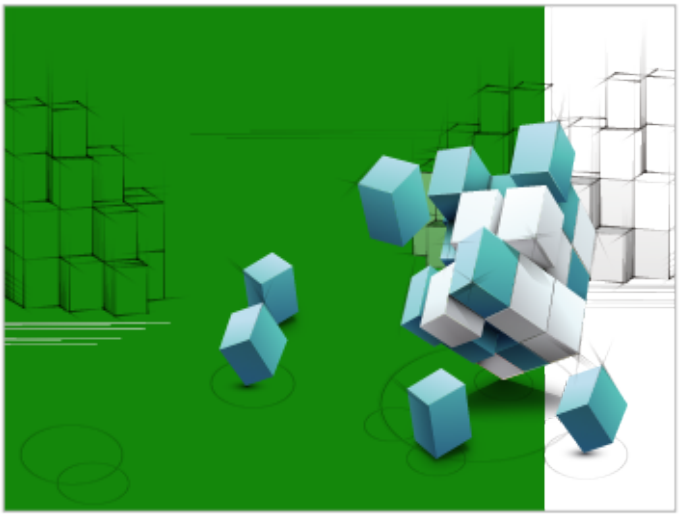
Programación básica



Filho Enrique Borjas García.
Sergio Rogelio Tinoco Martínez.
Luis Gerardo Zavala Figueroa.

SEP

conalep Michoacán

Gobierno del Estado MICHOACAN

Semestre 3

conalep
Michoacán

PRESENTA:

Programación básica

Autores:

Filho Enrique Borjas García.
Sergio Rogelio Tinoco Martínez.
Luis Gerardo Zavala Figueroa.

Título original de la obra:

Registro: **CONALEP-PROGRAM-1H**

Programa: Profesor escritor. Desarrollo de la competencia de la producción de información literaria y lectura.

Borjas García, F. E.; et al. (2014) **Programación básica.** México: CONALEP/CIE

xi, 272 p.; carta

Registro: **CONALEP-PROGRAM-1H** Documentos en línea

Editores:
Ing. Eduardo Ochoa Hernández
Lic. Filho Enrique Borjas García

Editorial: CONALEP Michoacán
Col. Chapultepec norte, Gral. Nicolás Bravo No. 144, Morelia, Michoacán.
http://www.cie.umich.mx/conalepweb2013/

Registro: **CONALEP-PROGRAM-1H**
ISBN: En trámite
Impreso en_____
Impreso en México –Printed in Mexico

Directorio

Dr. Salvador Jara Guerrero
Gobernador Constitucional del Estado de Michoacán

Dr. Armando Sepúlveda López
Secretario de Educación

Dr. Isaías Elizarraraz Alcaraz
Subsecretario de Educación Media Superior y Superior

Ing. Fernando Castillo Ávila
Director de Educación Media Superior

M.A. Candita Victoria Gil Jiménez
Directora General del Sistema CONALEP

Lic. Daniel Trujillo Mesina
Titular de la Oficina de Servicios Federales en Apoyo a la Educación en Michoacán

Dr. Gerardo Tinoco Ruiz
Rector de la Universidad Michoacana de San Nicolás de Hidalgo

Lic. José Arturo Villaseñor Gómez
Director General del CONALEP Michoacán

Lic. José Azahir Gutiérrez Hernández
Director Académico

L.E. Rogelio René Hernández Téllez
Director de Planeación, Programación y Presupuesto

Lic. Faradeh Velasco Rauda
Directora de Promoción y Vinculación

Ing. Mónica Leticia Zamudio Godínez
Directora de Informática

Lic. Víctor Manuel Gómez Delgado
Director de Servicios Administrativos

Ing. Genaro González Sánchez
Secretario General del SUTACONALEPMICH

Tec. Juan Pineda Calderón
Secretario General del SUTCONALEP

Prefacio

Estimado estudiante:

Las palabras que sombrean estas páginas, no son simple ciencia dentro del diálogo como depósitos de datos e información, ni son cuestión de vocabulario o listado de definiciones, son la experiencia generosa de la comunidad CONALEP Michoacán, esa realidad oculta pero necesaria que respaldó las tareas de investigación y composición literaria del discurso que integra este libro. Nos referimos a los profesores, administrativos y sindicatos que hoy convergen en el umbral de la existencia para apoyar a un grupo de profesores escritores que han creado en el sereno libre, arquitecturas de conocimientos como un viaje de aprendizaje que exigirá del estudiante, lo mejor de sí mismo ante la presencia luminosa del texto, ese que pretende enseñarle a caminar con la frente en alto.

Las ideas asociadas en este texto, equivalen a la imaginación lograda en el acto de escribir desde otros textos, al decodificarlas el estudiante, se le exige más vocabulario para enriquecer su habla y hacer ver a sus ojos más allá de la estrechez de la información que inunda a la sociedad moderna. El libro no presenta la superficie de la existencia como cruda observación, procura que su dificultad incite a perforar la realidad hasta reflexiones que renueven los modos inciertos de dar significado al mundo. La ciencia, la literatura y la tecnología no las percibimos como mundos incomunicables, los valores son explícitos caminos que las vinculan en torno al currículo del técnico bachiller. Tienen estos textos organización de premisas, técnicas, justificaciones, normas, criterios y como Usted se dará cuenta, también mostrará nuestros límites para seguir haciendo puentes entre las incesantes creaciones de nuevas fronteras de la investigación científica y técnica. Se pretende que estos libros sean contenido y no un libro de prácticas escolares, sean la herramienta de complementación para enriquecer los discursos de la enseñanza-aprendizaje.

Los profesores de CONALEP enfrentan a diario las carencias visibles de medios tecnológicos, materiales y documentales, sería fácil usar las palabras para señalar hasta el cansancio nuestras apremias, pero se ha decidido mejor producir libros como testimonios vivos y luminosos que renueven el rol social de la academia colegiada sensible a la condición social, susceptible de ir perfeccionándose con la acumulación de esta experiencia literaria, para servir de mejor manera al enriquecimiento de las competencias necesarias para realizar el sueño de éxito de tantos jóvenes Michoacanos.

Lic. José Arturo Villaseñor Gómez
Director General del CONALEP Michoacán

Mensaje a la comunidad académica

Con la colaboración docente, administrativa y sindical se realizó el esfuerzo de producir literatura de contenido en apoyo a la formación curricular en CONALEP Michoacán. El libro, esa experiencia de conocimiento se ha democratizado, ya no es un secreto o privilegio de unos cuantos, el texto virtual en la Web resolvió lo que la imprenta de Gutenberg no logró hacer, la auto publicación, la biblioteca virtual móvil, el libro electrónico y el texto digital; esto nos replantea migrar a una pedagogía interactiva con la experiencia del conocimiento. Desde luego que el libro clásico como dice Humberto Eco, nadie puede acabar con su poder en esta sociedad. Promover crear y leer literatura es enriquecer el vocabulario, el desarrollo intelectual, la agudeza de la creatividad y pintar la realidad con lo que nacemos libres: la imaginación.

El docente escritor, dirige el aprendizaje en función de la experiencia de reconstruir el conocimiento contemplado en el currículo. Se realiza el acto de pensar al escribir e investigar los modelos de conocimiento, ensayo, libro, tesis, reseña, síntesis, semblanza, resumen, análisis de texto, definición, argumento, razonamiento, hipótesis, patente, marco teórico, revisión, poema, novela, cuento, … entre otros, resuelven la necesidad de conocer, ser y aprender. El docente escritor escribe y publica su propuesta en el formato de libro, con ello, se abre a la crítica social y expone su calidad como marco ético de revaloración moral frente a su comunidad.

La escritura es más que gramática y semántica, es el acto de estructurar el pensamiento en un modelo de conocimiento, es volver a dar voz al profesor como producción de la libertad de cátedra, acto creativo original en el que encarna la soberanía de la sociedad como expresión cultural particular que habla desde su propio tiempo. Leer para crear es el acto sustantivo del novel. Escribir es una cierta reorganización del conocimiento previo en un acto de creación, donde la teoría literaria, los marcos normativos de estilo, la psicolingüística, la epistemología y la comunicación son los pilares de plataforma del aprendizaje centrado en el acto creativo.

Este libro fue escrito para compartir la felicidad de crear la presencia del docente en el texto. CONALEP desarrolla un programa académico para impulsar su capacidad y compromiso social para generar las ideas curriculares para enriquecer la sensibilidad y la imaginación científica, técnica y humanista de su comunidad.

Lic. José Azahir Gutiérrez Hernández
Director Académico

La palabra no solo nos otorga realidad, también tengo la sensación de que tiene vida propia separada de nosotros, y que cuando hablamos o escribimos, especialmente en momentos de intensa emoción, no hacemos más que dejarnos llevar por una sílaba amable o una frase complaciente.

Eric Ormsby. *Fine incisions*

Leer es una tarea de la memoria por medio de la cual las ficciones nos permiten disfrutar de experiencias ajenas y lejanas en el tiempo como si fueran nuestras.

Alberto Manguel. *La ciudad de las palabras*

En toda obra literaria se afirma una realidad independiente de la lengua y del estilo: la escritura considerada como la relación que establece el escritor con la sociedad, el lenguaje literario transformado por su destino social. Esta tercera dimensión de la forma tiene una historia que sigue paso a paso el desgarramiento de la conciencia: de la escritura transparente de los clásicos a la cada vez más perturbadora del siglo XIX, para llegar a la escritura neutra de nuestros días.

Roland Barthes, *El grado cero de la escritura*

Palabra escrita bajo luz

En un mundo cada día con más canales de comunicación, la palabra escrita camina por los muros que denuncian el drama catastrófico sobre el medio ambiente y sobre el control de la vida humana; el combustible de esta desesperanza produce apatía profunda por tener contacto con el mundo de la literatura, esta distorsión moral parece reflejarse entre los que no quieren sentir responsabilidad ni pensar, dejando a otros su indiferencia al ser prisioneros de ligeras razones y tirria justificada en la empresa de sobrevivir.

Especular en un mundo sin libros es exponer al mundo a la ausencia de pensamiento, creatividad y esperanza. Los libros, dedicados a ser arrebatados por el lector, están expuestos a ser poseídos por las bibliotecas vacías y que con el tiempo opacan sus páginas y empolvan la cubierta de lo que alguna vez fue un objeto de inspiración. Resulta difícil transcribir este instante de un peligroso espacio donde ya muy pocas palabras sobreviven dentro de la reflexión y las pocas sobrevivientes han abandonado la unión del sentido de vivir y el sentido del pensar científico. Escribir pasa de ser un placer repentino a ser una necesidad inminente, es el puente entre lo conocido y lo inexplorado. Es un reto de hoy en día inmiscuirse en lo que una vez fue lo cercano y dejar de lado la novedad tecnológica para poder, a través de las barreras que nos ciegan, abrir fronteras literarias. Es un proyecto que conspira a favor de la libertad creativa, de la felicidad lúcida cargada de libros embajadores de nuevas realidades.

Entre un mar de razones dentro del libro escolar en crisis, se percibe la ausencia de esa narrativa del cuerpo del texto, misma que alimenta al lector de una experiencia de conocimiento, su ausencia, es más un mal glosario, de un mal armado viaje literario científico o de ficción. En esos viajes de libros en crisis, nos cansamos de mirar espacios vacíos de talento, emociones y sensibilidad para responder a un entorno adverso; son muchas veces un triunfalismo de autoevaluación y una falsa puerta de una real competencia para actuar en la realidad. Uno no solo vive, escucha la voz interior de un libro, uno es fundado en el manejo del lenguaje que explica, crea, aplica o expande los límites del horizonte de nuestro imaginario actuante en lo real. No vivimos leyendo texto, sino leyendo

el paisaje de una realidad, el libro toma la voz del progreso en una siempre reconstrucción lingüística del sujeto que explica, transforma y comunica desde los desafíos de su generación.

La información cruda que tanto rellena los libros grises, oscuros y papel pintado; requiere ser dotada de conceptos que permitan alimentar al sujeto que toma decisiones, que explora con paso lento, que mira por dentro del lenguaje y aplica la información que cobra sentido en la siempre expansión de las ideas.

Escribir un libro es siempre reconstruir un discurso, sus lectores en este discurso son el puente a un texto profundo que demanda esfuerzo en la reconstrucción de los procesos de razonamiento y el entretejido del discurso que involucra información de fondo, esas fuentes que justifican su análisis y poseen significado privilegiado para la comprensión de una realidad.

El lector puede hacer uso del libro con su propia experiencia y con su autoayuda, al precisar términos y conceptos para prolongar su horizonte de interpretación, el libro se hace cargo de la memoria de un plan de estudios, es un discurso de diferentes capas de argumentos, tras este texto se anuncia un orden de experiencia propuesto para su aprendizaje. El libro está conformado para jóvenes con memoria sin dolor para nuevas palabras, aborda el olvido como una deficiencia de interactividad entre el discurso argumentativo y los referentes conceptuales. Esto es el reto en la producción de los libros CONALEP. La propuesta es una reconstrucción de una semántica más profunda, como el principal reto del estudiante técnico bachiller del siglo XXI.

Libro,…

todos te miran,

nosotros te vemos bajo la piel.

SUMARIO

Primera parte
Introducción a la programación

Segunda parte
Lenguaje C

Tercera parte
Tipos de datos, variables y constantes

Cuarta parte
Funciones

Quinta parte
Arreglos

Sexta parte
Apuntadores

Séptima parte
Estructuras de datos

Primera parte

Introducción a la programación

1. Introducción a la programación

1.1. Antecedentes

Ejercicio 1

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Instrucciones	Software	Programa	Lenguaje C
ALGOL	John McCarthy	John von Neumann	FORTRAN
Numérico	Informática	Lenguaje de programación	

Enunciados:

1.- La _____ es la ciencia del estudio y aplicación del conocimiento por medio de procesos automatizados de la información, con el uso de las computadoras.

2.- El _____ es un programa o conjunto de programas que se ejecutan en la computadora.

3.- Decimos que un _____ está formado por un conjunto de instrucciones que se ejecutan en una computadora, con la finalidad de resolver algún problema. Asimismo, especifica la forma de procesar y operar sobre los datos que son utilizados en el equipo de cómputo.

4.- Un _____ es una herramienta informática que permite crear software para que una computadora pueda realizar todas sus funciones. Está formado por un conjunto de _____ escritas en forma lógica y basadas en una secuencia de operaciones que permiten resolver un problema o procesar información en la computadora.

5.- _____ fue quien propuso la arquitectura de las computadoras que son utilizadas hasta el día de hoy.

6.- Las computadoras de los años 50's, construidas con bulbos y –posteriormente– transistores, utilizaban un lenguaje basado en un código _____, al cual también se le denominó lenguaje máquina.

7.- El lenguaje _____ es considerado como el primer lenguaje de programación de alto nivel.

8.- El lenguaje _____, tiene una importancia muy grande en la historia de los lenguajes de programación debido a que, de él, se derivaron varios de los lenguajes de programación que actualmente se emplean.

9.- _____ diseñó el lenguaje de programación de alto nivel llamado LISP.

10.- El _____, que aparece en 1972, fue implementado inicialmente para crear programas de aplicaciones en el sistema operativo UNIX.

1.2. Conceptos generales

1.2.1. Programación

1.2.1.1. Concepto

Ejercicio 2

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Escriba una de las respuestas, que se muestran en el recuadro siguiente, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Programa	Instrucciones	Fuente	Claro
Ejecutable	Implementación	Ejecución	Validación
Algoritmo	Eficiente	Programador	Legible
Acciones	Instrucciones	Diseño del algoritmo	
Proceso de desarrollo de software		Lenguaje de programación	

Enunciados:

1.- La definición de programa se utiliza para describir dos conceptos, el primero se refiere a las _____ o código _____ creados por el programador y, el segundo, alude a un software completo y _____.

2.- Un _____ también se puede definir como la secuencia lógica y ordenada de _____ o _____ que deben seguirse en el proceso de la computadora.

3.- Un _____ es quien desarrolla un programa.

4.- Se le llama _____ al proceso de traducir un algoritmo a un programa.

5.- Decimos que un _____ está formado por un conjunto de instrucciones que permiten el desarrollo de un programa, así como especificar la forma de procesar y operar sobre los datos que se manipulan en la computadora.

6.- El _____ consiste en construir un programa o aplicación para que resuelva un problema de la vida diaria.

7.- Dentro del proceso de desarrollo de software, el _____ es el paso donde se analiza el problema a resolver, con la finalidad de desarrollar una solución adecuada. El resultado de este paso es una especificación en papel, la cual será implementada como un programa, en forma posterior.

8.- El proceso de _____ y _____ del software es la etapa final en el desarrollo de un programa, donde se eliminan los errores y se verifica que el programa dé solución al problema para el que fue creado.

9.- Un programa tiene tres objetivos principales: corrección, es decir, que el programa dé solución a los problemas para los que fue creado; debe ser _____, en otras palabras, que debe consumir los recursos estrictamente necesarios; y, por último, debe ser _____ y _____ para tener mayor facilidad en su mantenimiento.

10.- Un _____ es un procedimiento bien definido para la solución de un problema o realización de una tarea.

1.2.1.2. Paradigmas

Ejercicio 3

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis, la letra correspondiente al inciso que indique la relación correcta.

a) Paradigma de la programación.	( )	1. Programación procedural.
b) Programación funcional.	( )	2. Especificación de tipo de datos con propiedades y funcionalidad, empleada en los lenguajes de programación orientada a objetos.
c) Programación estructurada.	( )	3. Es la programación considerada opuesta a la programación orientada a objetos, y se basa en la aplicación de funciones a valores conocidos, también se conoce como lenguaje aplicativo.
d) Serie de procedimientos que actúan sobre los datos.	( )	4. Conjunto de instrucciones o procedimientos que se ejecutan una después de otra.
e) Función.	( )	5. Es la programación basada en la división de tareas.
f) Programación orientada a objetos.	( )	6. Propiedad por la cual es posible construir nuevas clases a partir de las ya existentes.
g) Objeto.	( )	7. Paradigma de programación en el que la estructura y la ejecución de programa están definidas por sucesos que ocurren en el sistema o son determinados por el usuario.
h) Clase.	( )	8. Es un modelo o patrón que define la manera en que se desarrolla un programa.
i) Programación orientada a eventos.	( )	9. Entidad de una determinada clase, con un determinado estado, capaz de interacción.
j) Herencia.	( )	10. Es el paradigma de programación que tiene como elementos la clase, el objeto y la herencia, entre otros.

1.2.1.3. Paradigmas de la computación moderna

Ejercicio 4

Instrucciones: Lea con atención cada uno de los enunciados que enseguida se muestran. De las opciones enlistadas, subraye solamente el inciso cuya definición corresponda correctamente con el enunciado.

1.- Paradigma de programación en el cual tanto la estructura como la ejecución de los programas van determinadas por los sucesos que ocurran en el sistema, o los generados por el usuario.

a) Programación estructurada
b) Programación por eventos
c) Programación orientada a objetos
d) Programación funcional

2.- Es un lenguaje de programación orientado a eventos creado por la compañía Microsoft.

a) Lenguaje C
b) Pascal
c) Visual Basic
d) Java

3.- Si el texto del programa tiene una estructura sintáctica, se dice que su código es…

a) Estructurado
b) Funcional
c) Por eventos
d) Orientado a objetos

4.- Es el elemento fundamental de la programación orientada a objetos.

a) Estructura
b) Evento
c) Clase
d) Herencia

5.- Es un estímulo ocurrido durante la ejecución de un programa.

a) Objeto
b) Clase
c) Evento
d) Instrucción

6.- En la programación orientada a objetos es la declaración de una variable concreta del tipo de una clase.

a) Objeto
b) Clase
c) Evento
d) Instrucción

7.- Este tipo de programación puede prescindir completamente de los ciclos y las variables.

a) Programación estructurada
b) Programación orientada a objetos
c) Programación funcional
d) Programación por eventos

8.- Es el lenguaje que se utiliza para crear programas en la computadora.

a) Algorítmico
b) De programación
c) Metodológico
d) Estructurado

9.- En la programación orientada a objetos, es una propiedad que define una nueva clase a partir de clases ya existentes.

a) Herencia
b) Clase
c) Evento
d) Instrucción

10.- Se dice que fue el primer paradigma de programación universalmente aceptado; debido a que es fácil de leer, depurar y mantener.

a) Programación estructurada
b) Programación orientada a objetos
c) Programación funcional
d) Programación por eventos

1.2.2. Lenguajes de Programación
1.2.2.1. Conceptos

Ejercicio 5

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "F", dentro de los paréntesis en la columna de la derecha, si la afirmación es falsa o, escriba la letra "V", si la afirmación es verdadera.

1. El lenguaje de bajo nivel también es llamado lenguaje máquina. ( )
2. El lenguaje C es un lenguaje funcional. ( )
3. El lenguaje Java es un lenguaje orientado a aspectos. ( )
4. El lenguaje Delphi es derivado del lenguaje Pascal. ( )
5. El lenguaje COBOL es un lenguaje orientado a negocios. ( )
6. El lenguaje FORTRAN es utilizado en la programación de inteligencia artificial. ( )
7. El lenguaje ALGOL es el predecesor de lenguajes como Pascal. ( )
8. El lenguaje Pascal es un lenguaje orientado a objetos. ( )
9. El lenguaje SQL se utiliza para administrar servidores web. ( )
10. PROLOG es utilizado en la programación de inteligencia artificial. ( )

1.2.2.2. Jerarquía

Ejercicio 6

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis, la letra correspondiente al inciso que indique la relación correcta.

a) Mnemotécnicos.	( )	1. Lenguaje también llamado ensamblador.
b) Lenguaje máquina.	( )	2. Es un tipo de lenguaje de bajo nivel.
c) Lenguaje ensamblador.	( )	3. Son las órdenes que escribe el programador cuando desarrolla un programa.
d) Lenguaje de mnemotécnicos.	( )	4. Es un tipo de lenguaje de alto nivel.
e) Lenguaje de alto nivel.	( )	5. Es aquel tipo de lenguaje que interactúa directamente con el hardware de la computadora.
f) Lenguaje C.	( )	6. Este tipo de software convierte el programa de lenguaje ensamblador a lenguaje máquina.
g) Traductor.	( )	7. Es el tipo de lenguaje utilizado por la máquina.
h) Programa.	( )	8. Tipo de lenguaje con las instrucciones más cercanas al lenguaje humano, y más distantes del lenguaje binario.
i) Código fuente.	( )	9. Conjunto de palabras que representan instrucciones a ejecutar en la computadora, empleado en el lenguaje ensamblador.
j) Lenguaje numérico (representado en bits).	( )	10. Es el nombre que recibe una secuencia de instrucciones escrita en algún lenguaje de alto nivel.

1.2.2.3. Elementos

Ejercicio 7

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Estática	Traductores	Sistemas de tipos	Análisis sintáctico
Símbolos	Estructura léxica	Sintaxis	Símbolos especiales
Análisis léxico	Análisis semántico		

Enunciados:

1.- La _____ es la estructura del programa y el conjunto de reglas que impone para su ejecución correcta.

2.- A los elementos de la _____ de los lenguajes de programación se les llama *token*.

3.- El _____ es el análisis de la gramática del lenguaje.

4.- El _____ comprueba el significado de las instrucciones.

5.- El _____ elimina del programa toda la información que no es necesaria, como las líneas, los espacios en blanco, los caracteres de tabulación, etc.

6.- El análisis sintáctico comprueba que los _____ del lenguaje del código del programa se escriban correctamente.

7.- Los _____ de los lenguajes de programación, transforman un lenguaje fuente a un lenguaje máquina.

8.- Los _____ forman parte de la estructura léxica de un lenguaje de programación.

9.- Una propiedad _____ define la forma en que el lenguaje de programación clasifica, manipula e interpreta, en *tipos*, los diferentes valores y expresiones.

10.- Los _____ incluyen las reglas sobre el significado y/o el análisis del flujo de datos que se puede verificar en tiempo de compilación.

1.2.2.4. Implementación

Ejercicio 8

Instrucciones: Lea con atención cada uno de los enunciados que enseguida se muestran. De las opciones enlistadas, subraye solamente el inciso cuya definición corresponda correctamente con el enunciado.

1.- Es el traductor de un lenguaje de alto nivel que no genera en forma permanente la transcripción respectiva al lenguaje máquina.

a) Intérprete b) Compilador c) Código objeto d) Código fuente

2.- Es el código de instrucciones creado por el programador.

a) Compilador b) Intérprete c) Código fuente d) Código objeto

3.- Es la versión traducida que produce el compilador.

a) Código fuente b) Código objeto c) Intérprete d) Compilador

4.- Traduce el lenguaje de alto nivel a lenguaje ensamblador, y posteriormente a lenguaje máquina; finalmente verifica que no existan errores para ejecutar el programa posteriormente.

a) Compilador b) Enlazador c) Intérprete d) Código fuente

5.- Está formado por una secuencia de instrucciones.

a) Código fuente b) Lenguaje c) Programador d) Compilador

6.- Es un lenguaje de programación de alto nivel que utiliza un traductor de tipo intérprete.

a) Lenguaje C b) Basic c) FORTRAN d) COBOL

7.- Es un lenguaje de programación de alto nivel que utiliza un compilador.

a) Lenguaje C b) Basic c) LISP d) COBOL

8.- Es la categoría que define a los compiladores e intérpretes.

a) Código fuente b) Traductores c) Fase de análisis d) Fase de síntesis

9.- Es la primera fase en el proceso de traducción del programa.

a) Fase léxica b) Fase de síntesis c) Fase de análisis d) Fase semántica

10.- Es la segunda fase en el proceso de traducción de un programa.

a) Fase léxica b) Fase de síntesis c) Fase de análisis d) Fase semántica

1.3. Ciclo de vida del software

1.3.1. Conceptos

Ejercicio 9

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis, la letra correspondiente al inciso que indique la relación correcta.

a) Software.	( )	1. Este modelo de desarrollo de software está representado por acciones del usuario.
b) Ingeniería de software.	( )	2. Es la disciplina que comprende el software, el hardware y la interacción del sistema con el usuario.
c) Ingeniería en sistemas.	( )	3. Es el desarrollo de un programa especialmente diseñado para un cliente o empresa en particular, para actividades que él mismo requiere.
d) Software genérico.	( )	4. Conjunto de programas, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación.
e) Usabilidad.	( )	5. Es un conjunto específico de actividades o tareas para producir un software.
f) Modelo en cascada.	( )	6. Es un atributo de aquel software que hace un uso adecuado de los recursos, con mejor tiempo de respuesta y mantenimiento.
g) Software personalizado.	( )	7. Disciplina que comprende todos los aspectos para producir un software.
h) Eficiencia.	( )	8. Programa o software desarrollado para el uso general de un grupo de clientes o empresas. Comúnmente cuenta con más recursos o herramientas de los que el cliente que los utiliza requiere.
i) Modelo de flujo de trabajo.	( )	9. Está basado en etapas o tareas bien definidas y separadas, donde al término de una de ellas, se continúa con la siguiente.
j) Proceso de desarrollo de software.	( )	10. El software debe ser fácil de utilizar, para lo cual debe presentar una interfaz sencilla de interpretar por el usuario.

1.3.2. Modelo clásico, lineal secuencial o modelo en cascada

Ejercicio 10

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Etapas	Métodos	Análisis de requerimientos	Diseño
Implementación	Software	Clásico	Prueba
Mantenimiento	Análisis previo	Lenguaje de programación	

Enunciados:

1.- El ciclo de vida _____ también es conocido como ciclo de vida en cascada.

2.- El ciclo de vida en cascada está dividido en _____, cada una de ellas está formada por procedimientos bien definidos.

3.- El ciclo de vida del _____ está formado por _____ y técnicas de la ingeniería de software.

4.- En la etapa de análisis de requerimientos se selecciona el _____ que se ha de utilizar para crear el programa.

5.- En la etapa de _____ o codificación, se pasa el diseño elaborado en etapas anteriores, a instrucciones del lenguaje de programación que se va a utilizar para crear el software.

6.- En la etapa de _____ se hacen cambios y mejoras al software.

7.- En la etapa de _____ se definen las necesidades de información que se van a resolver con el software.

8.- Es la tercera etapa del ciclo de desarrollo de software en cascada, llamada etapa de _____, ya que en ella se da la solución al problema y se determinan las funciones que deben integrarse en el programa.

9.- En la etapa de _____ se realizan varios procedimientos para demostrar el funcionamiento correcto del software, y corroborar que está realizando las funciones para las cuales fue creado.

10.- En la etapa de _____ se definen los sistemas de cómputo en los que se implantará el software a desarrollar, el entorno de hardware y el entorno de red, entre otros.

1.3.2.1. Modelo en cascada y etapa de diseño

Ejercicio 11. Etapas del modelo en cascada

Instrucciones: Ordene las etapas del modelo en cascada del ciclo de vida del software, que están en la columna de la izquierda, de acuerdo a la cronología en que se realizan dentro de un proceso de desarrollo de software. Indique la cronología anotando, dentro de los paréntesis, el número 1 para el proceso que se realiza en primer lugar, el número 2 para el segundo, el 3 para el tercero, y así en forma sucesiva.

Diseño	( )
Análisis de requerimientos	( )
Prueba	( )
Mantenimiento	( )
Programación	( )
Análisis previo	( )

Ejercicio 11.1 Pasos de la etapa de diseño

Instrucciones: Realice el mismo procedimiento que en el apartado anterior, a fin de ordenar los pasos de la etapa de diseño que se realizan dentro de un proceso de desarrollo de software.

Representaciones de interface	( )
Estructura de datos	( )
Algoritmo	( )

1.3.2.2. Análisis de Requerimientos

Ejercicio 12

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Síntesis	Revisión	Especificación	Evaluación
Partición	Función	Crear modelos	Mala comunicación
Especificación de requisitos del software		Flujo de la información	
Reconocimiento del problema			

Enunciados

1.- La _____ y la _____ son fundamentales en el análisis de requerimientos, ya que permiten conocer el flujo de la información, estructurar los datos, elaborar las funciones que debe realizar el software, con base en los requerimientos del cliente.

2.- La _____ es la última fase del análisis de requerimientos, en la cual se realiza un análisis de los pasos anteriores y se puede dar a conocer al cliente la solución sobre la cual se diseñará el software.

3.- La _____ es un principio del análisis de requerimientos, que tiene el objetivo de dividir un gran problema en otros más pequeños y comprensibles, para posteriormente enlazarlos, a fin de solucionar el problema principal.

4.- En el _____ el analista debe evaluar las consideraciones que se hicieron durante la planeación del software. Además, debe cuestionar al cliente para tener plenamente identificado el problema que se presenta y la solución que se propone, con la finalidad de que el producto (software) sea la solución más adecuada para el usuario.

5.- En la _____ se representa el software que se propone diseñar, de una manera comprensible para el cliente, con la finalidad de mostrar la forma en que se da solución al problema.

6.- La _____ es uno de los principales problemas al que se enfrenta un equipo de desarrollo al realizar un análisis de requerimientos, provocando un mal entendimiento del problema.

7.- Para ayudar a comprender mejor el proceso y el flujo de datos se pueden _____ , los cuales, además, son utilizados posteriormente en la etapa de diseño.

8.- El resultado final de la etapa de análisis es la _____.

9.- El _____ es un principio del ámbito de la información del análisis de requisitos, en él se tiene la representación y control de los datos, por lo que permite ir esquematizando la información.

10.- La _____ del analista, en el análisis de requerimientos, consiste en ofrecer una solución adecuada y eficiente en la utilización de los recursos informáticos, a fin de completar un producto final (software), que solucione un problema a un cliente.

1.3.2.3. Diseño: Medios de expresión

Ejercicio 13

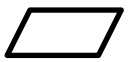
Instrucciones: Para cada una de las afirmaciones siguientes escriba dentro de los paréntesis la letra "F" si la afirmación es falsa, o escriba la letra "V" si la afirmación es verdadera.

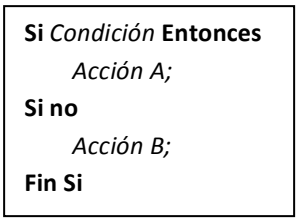
1. La eficiencia es uno de los requisitos en el diseño de programas, sin importar que estos resulten incomprensibles a otros programadores. **( )**
2. La estructura de un programa en módulos facilita su mantenimiento. **( )**
3. Un algoritmo es un conjunto ordenado de instrucciones que permite resolver un problema y codificarlo en un lenguaje de programación. **( )**
4. Un programa es un conjunto de instrucciones especificadas y definidas por un lenguaje de programación, que indica cómo debe trabajar la computadora. **( )**
5. La codificación es transformar un programa en un algoritmo. **( )**
6. La notación algorítmica es necesaria para el diseño de un algoritmo. **( )**
7. El pseudocódigo es una notación algorítmica gráfica. **( )**
8. El diagrama N-S (Nassi-Schneiderman) es un algoritmo gráfico con texto de acciones de las tareas a realizar. **( )**
9. Un algoritmo de sistemas formales utiliza notaciones lógicas y matemáticas. **( )**
10. Un diagrama de flujo es una notación algorítmica textual. **( )**

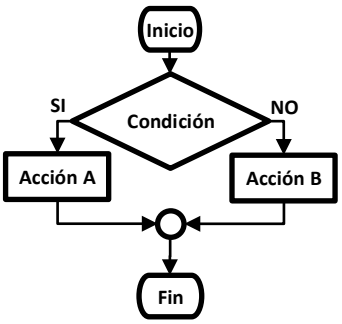
1.3.2.4. Diseño: Diagramas de flujo

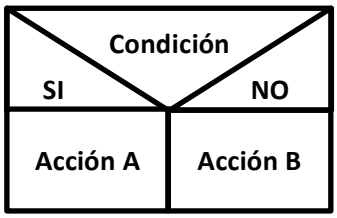
Ejercicio 14

Instrucciones: Relacione correctamente cada uno de los conceptos enlistados en la columna de la derecha con una sola de las imágenes en la columna de la izquierda. Dibuje una línea para indicar la relación correcta.









1. Ejemplo de sistema formal.
2. Línea de flujo.
3. Operación (acción de datos).
4. Ejemplo de diagrama de flujo.
5. Operación lógica (toma de decisión).
6. Ejemplo de diagrama Nassi-Schneiderman.
7. Entrada/salida de datos.
8. Inicio y fin del algoritmo.
9. Conector (cuando se secciona el diagrama).

$$f(x) = \sum_{n=0}^{\infty} a_n \cos(nx) + \sum_{n=0}^{\infty} b_n \sin(nx)$$

10. Ejemplo de pseudocódigo.

1.3.2.4. Diseño de software

Ejercicio 15

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Diseño del sistema.	( )	1.- Etapa en la que se traducen los algoritmos en código del lenguaje de programación a utilizar.
b) Etapa de prueba.	( )	2.- Etapa en la que el producto final de software es utilizado por el cliente.
c) Mantenimiento.	( )	3.- Es una desventaja, ya que el cliente recibe el programa para su uso, hasta el final de las etapas del modelo lineal.
d) Alcance del ciclo de vida del software.	( )	4.- Etapa del desarrollo del software en la que se verifica el buen funcionamiento del mismo.
e) Actualización.	( )	5.- En la práctica es una de las desventajas del ciclo secuencial porque, en la aplicación real del mismo, se introducen modificaciones dentro de cada una de las etapas del proceso de desarrollo de software.
f) Implantación.	( )	6.- Es una de las desventajas del modelo en cascada, ya que solo se hacen pruebas después de la etapa de implementación.
g) Implementación.	( )	7.- En la fase de análisis es la forma en que el sistema cumplirá con los requerimientos. También se le denomina diseño lógico.
h) El riesgo.	( )	8.- Son los cambios que se realizan para corregir o mejorar el programa.
i) No seguir estrictamente el ciclo secuencial.	( )	9.- Consiste en determinar hasta dónde se desea llegar con el proyecto de software, es decir, en qué momento detener el proceso de desarrollo del mismo.

j) El programa se finaliza hasta la última etapa del ciclo.	( ) 10.-Consiste en la mejora del software hacia las nuevas tendencias y avances informáticos.

1.3.3. Otros modelos

Ejercicio 16

Instrucciones: Lea con atención cada uno de los enunciados que enseguida se muestran. De las opciones enlistadas, subraye solamente el inciso cuya definición corresponda correctamente con el enunciado.

1.- En este modelo los requerimientos del cliente pueden cambiar en cualquier momento, ya que estos pueden ir sufriendo modificaciones durante el desarrollo del software.

a) Modelo evolutivo
b) Modelo en espiral
c) Modelo iterativo
d) Modelo por prototipos

2.- Este modelo se utiliza cuando el cliente recibe únicamente el producto final.

a) Modelo lineal
b) Modelo en espiral
c) Modelo iterativo
d) Modelo por prototipos

3.- Este modelo permite crear especificaciones iniciales y hacer un producto parcial; incrementar las especificaciones y realizar un producto intermedio; seguir incrementando las especificaciones y realizar los productos intermedios que se deseen, hasta llegar al producto final.

a) Modelo lineal
b) Modelo en espiral
c) Modelo iterativo
d) Modelo por prototipos

4.- Es la repetición de varios ciclos en cascada, al término de cada uno de los cuales se entrega al cliente una versión del programa que mejora la versión anterior. También es el propio cliente quien evalúa el producto de software de cada repetición, hasta que se cumplan las necesidades que él crea suficientes.

a) Modelo lineal
b) Modelo en espiral
c) Modelo incremental
d) Modelo por prototipos

5.- Este método utiliza diversos prototipos que son mostrados al cliente, a fin de evaluar si satisfacen sus necesidades y son rediseñados de acuerdo a las observaciones que este realice. El método se emplea cuando las especificaciones no son completas.

a) Modelo lineal
b) Modelo en espiral
c) Modelo iterativo
d) Modelo por prototipos

6.- La planificación y el análisis de riesgos forman parte de este modelo.

a) Modelo lineal
b) Modelo en espiral
c) Modelo iterativo
d) Modelo por prototipos

7.- Este modelo de ciclo de vida del software utiliza atributos y métodos.

a) Modelo orientado a objetos
b) Modelo en espiral
c) Modelo iterativo
d) Modelo lineal

8.- Es el modelo de desarrollo de software formado por varios ciclos del modelo en cascada.

a) Modelo lineal
b) Modelo en espiral
c) Modelo iterativo
d) Modelo por prototipos

9.- Busca analizar el código para obtener el diseño mediante el cual fue implementado.

a) Prototipo
b) Modelo
c) Ingeniería inversa
d) Reingeniería

10.- Utiliza los resultados obtenidos por la ingeniería inversa para corregir o prevenir los posibles errores que pudiera tener el software.

a) Prototipo
b) Modelo
c) Ingeniería inversa
d) Reingeniería

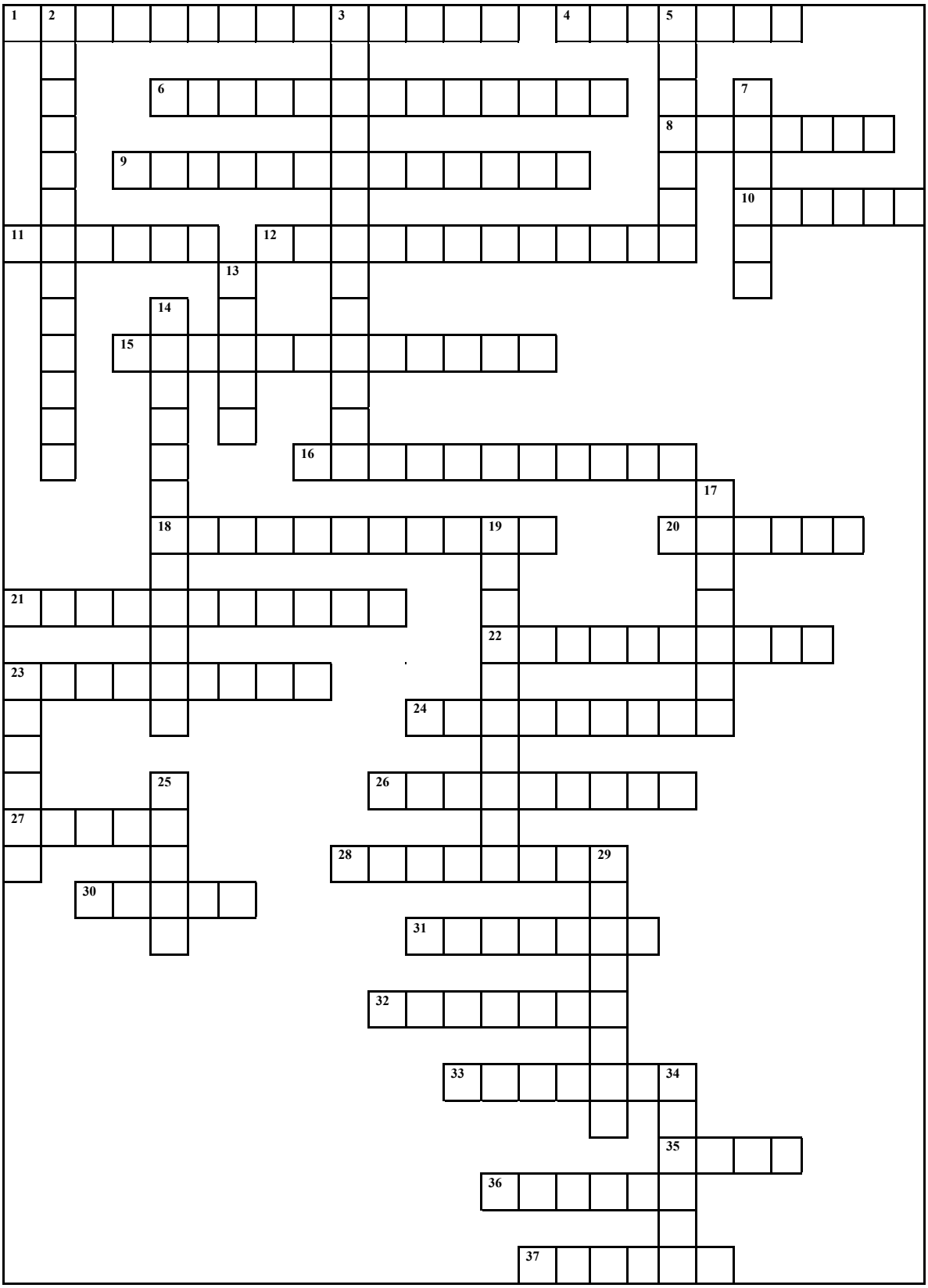
Crucigrama

HORIZONTALES

1.- Es el proceso de pasar un algoritmo a código de algún lenguaje de programación.
4.- El análisis de riegos es una de las etapas del modelo…
6.- Conjunto de palabras que representan instrucciones a ejecutar en la computadora, empleados en el lenguaje ensamblador.
8.- Última palabra del nombre del paradigma de programación que utiliza un traductor de tipo intérprete.
9.- Nombre que recibe aquel software que es desarrollado para un cliente o empresa en particular.
10.- Segunda palabra del nombre de la primera etapa del modelo en cascada.
11.- En el modelo en cascada, etapa donde se plantea el algoritmo del software.
12.- Utiliza los resultados de la ingeniería inversa para corregir o prevenir errores que pudiera tener el software.
15.- Notación de algoritmo en forma de texto.
16.- Lenguaje de bajo nivel.
18.- Transforman un lenguaje de alto nivel en lenguaje máquina.
20.- Etapa del modelo en cascada donde se corrigen los errores que pudiera tener el software, antes de entregarlo al cliente.
21.- Nombre que recibe la persona que desarrolla un programa.
22.- Traductor que verifica que no existan errores en el código, para posteriormente ejecutar el programa.
23.- Modelo o patrón que define los tipos de lenguajes de programación.
24.- Modelo de desarrollo de software formado por varios ciclos del modelo en cascada.
26.- Paradigma de programación que puede prescindir de ciclos y variables.
27.- Lenguaje de programación del cual deriva el lenguaje Pascal.
28.- Última palabra del nombre de los algoritmos de notaciones matemáticas.
30.- Segunda palabra del nombre de un lenguaje de programación orientado a eventos.
31.- Lenguaje de la computadora.
32.- Modelo de desarrollo de software llamado en línea, secuencial o en…
33.- Tipo de algoritmo del diagrama N-S.
35.- Lenguaje de programación orientado a objetos.
36.- Código creado por el programador.
37.- Lenguaje de programación utilizado en inteligencia artificial.

VERTICALES

2.- Etapa del modelo lineal en la cual se realizan cambios para corregir o mejorar el programa, después de haberlo entregado al cliente.
3.- Consiste en mejorar el software, hacia nuevas tendencias y/o avances informáticos.
5.- Tipo de ingeniería que analiza el código para obtener el diseño mediante el cual fue implementado.
7.- Lenguaje de programación derivado del lenguaje Pascal.
13.- Tipo de diagrama que ayuda a visualizar gráficamente el algoritmo.
14.- Paradigma de los lenguajes de programación C y Pascal.
17.- Trabajo (literal).
19.- Atributo de los modelos de software que hacen un uso adecuado de los recursos.
23.- Lenguaje de programación estructurado.
25.- Elemento de la programación orientada a objetos.
29.- Incluye las reglas sobre el significado y/o análisis del flujo de datos que se verifican en la compilación de un programa.
34.- Código resultante de la compilación de un programa escrito en un lenguaje de alto nivel.



Apéndice

Ejercicio 1

1.- Informática. 2.- Software. 3.- Programa. 4.- Lenguaje de programación / Instrucciones. 5.- John von Neumann. 6.- Numérico. 7.- FORTRAN. 8.- ALGOL. 9.- John McCarthy. 10.- Lenguaje C.

Ejercicio 2

1.- Instrucciones / Fuente / Ejecutable. 2.- Programa / Acciones / Instrucciones. 3.- Programador. 4.- Implementación. 5.- Lenguaje de programación. 6.- Proceso de desarrollo de software. 7.- Diseño del algoritmo. 8.- Ejecución / Validación. 9.- Eficiente / Claro / Legible. 10.- Algoritmo.

Ejercicio 3

1.- d. 2.- h. 3.- b. 4.- e. 5.- c. 6.- j. 7.- i. 8.- a. 9.- g. 10.- f.

Ejercicio 4

1.- b) Programación por eventos. 2.- c) Visual basic. 3.- a) Estructurado. 4.- c) Clase. 5.- c) Evento. 6.- a) Objeto. 7.- c) Programación funcional. 8.- b) De programación. 9.- a) Herencia. 10.- a) Programación estructurada.

Ejercicio 5

1.- V. 2.- F. 3.- F. 4.- V. 5.- V. 6.- F. 7.- V. 8.- F. 9.- F. 10.- V.

Ejercicio 6

1. – d. 2. – c. 3. – i. 4. – f. 5. – b. 6. – g. 7. – j. 8. – e. 9.- a. 10.- h.

Ejercicio 7

1. Sintaxis. 2.- Estructura léxica. 3.- Análisis sintáctico. 4.- Análisis semántico. 5.- Análisis léxico. 6.- Símbolos. 7.- Traductores. 8.- Símbolos especiales. 9.- Estática. 10.- Sistemas de tipos.

Ejercicio 8

1.- a) Intérprete. 2.- c) Código fuente. 3.- b) Código objeto. 4.- a) Compilador. 5.- a) Código fuente. 6.- b) Basic. 7.- a) Lenguaje C. 8.- b) Traductores. 9.- c) Fase de análisis. 10.- b) Fase de síntesis.

Ejercicio 9

1. – i. 2. – c. 3. – g. 4. – a. 5. – j. 6. – h. 7. – b. 8. – d. 9. – f. 10. – e.

Ejercicio 10

1.- Clásico. 2.- Etapas. 3.- Software / Métodos. 4.- Lenguaje de programación. 5.- Implementación. 6.- Mantenimiento. 7.-Análisis previo. 8.- Diseño. 9.- Prueba. 10.- Análisis de requerimientos.

Ejercicio 11

Etapas: 3, 2, 5, 6, 4 y 1

Pasos: 3, 1 y 2

Ejercicio 12

1.- Revisión / Evaluación. 2.- Síntesis. 3.- Partición. 4.- Reconocimiento del problema. 5.- Especificación. 6.- Mala comunicación. 7.- Crear modelos. 8.- Especificación de requisitos del software. 9.- Flujo de la información. 10.- Función.

Ejercicio 13

1.- F. 2.- V. 3.-F. 4.- V. 5.- F. 6.- V. 7.- F. 8.- V. 9.- V. 10.- F.

Ejercicio 14

Inicio y fin del algoritmo.

Operación (acción de datos).

Operación lógica (toma de decisión).

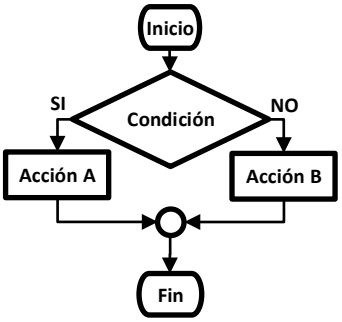
Entrada/salida de datos.

Línea de flujo.

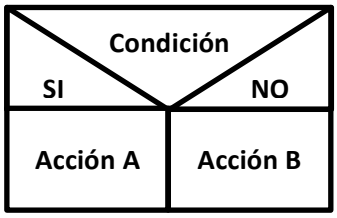
Conector (cuando se secciona el diagrama).

Si *Condición* **Entonces**
Acción A;
Si no
Acción B;
Fin Si

Ejemplo de pseudocódigo.



Ejemplo de diagrama de flujo.



Ejemplo de diagrama Nassi-Schneiderman.

$$f(x) = \sum_{n=0}^{\infty} a_n \cos(nx) + \sum_{n=0}^{\infty} b_n \sin(nx)$$

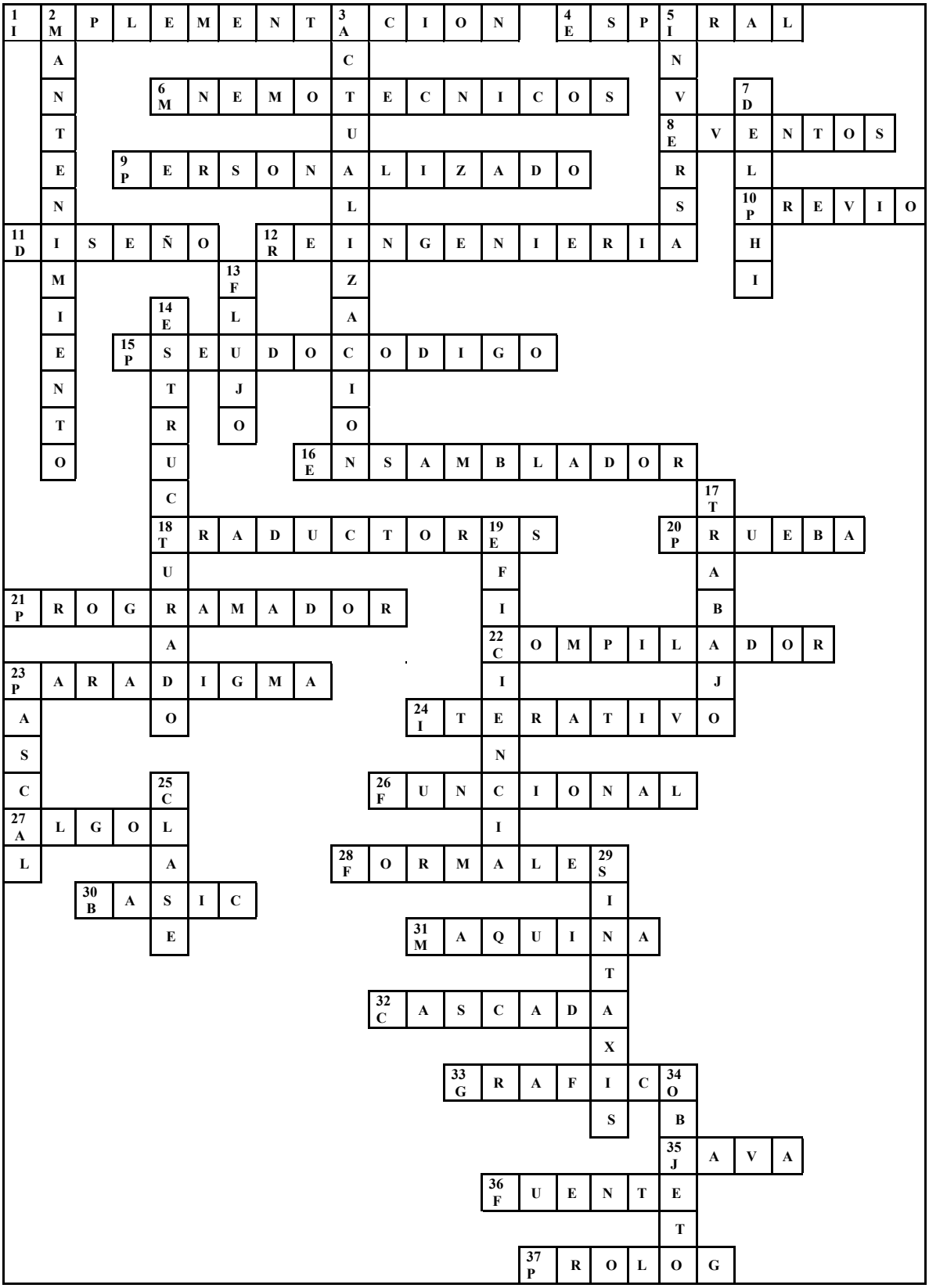
Ejemplo de sistema formal.

Ejercicio 15

1.- g. 2.- f. 3.- j. 4.- b. 5.- i. 6.- h. 7.- a. 8.- c. 9.- d. 10.- e.

Ejercicio 16

1.- a) Modelo evolutivo. 2.- a) Modelo lineal. 3.- d) Modelo por prototipos. 4.- c) Modelo incremental. 5.- d) Modelo por prototipos. 6.- b) Modelo en espiral. 7.- a) Modelo orientado a objetos. 8.- c) Modelo iterativo. 9.- c) Ingeniería inversa. 10.- d) Reingeniería.


1 I
2 M
P L E M E N T
3 A
C I O N
4 E
S P
5 I
R A L
A
C
N
N
6 M
N E M O T
E C N I C O S
V
7 D
T
U
8 E
V E N T O S
E
9 P
E R S O N A L I Z A D O
R
L
N
L
S
10 P
R E V I O
11 D
I S E Ñ O
12 R
E I N G E N I E R I A
H
M
13 F
Z
I
I
14 E
L
A
E
15 P
S E U D O C O D I G O
N
T
J
I
T
R
O
O
O
U
16 E
N S A M B L A D O R
C
17 T
18 T
R A D U C T O R
19 E
S
20 P
R U E B A
U
F
A
21 P
R O G R A M A D O R
I
B
A
22 C
O M P I L A D O R
23 P
A R A D I G M A
I
J
A
O
24 I
T E R A T I V O
S
N
C
25 C
26 F
U N C I O N A L
27 A
L G O L
I
L
A
28 F
O R M A L E
29 S
I
30 B
A S I C
E
31 M
A Q U I N A
T
32 C
A S C A D A
X
33 G
R A F I C
34 O
S
B
35 J
A V A
36 F
U E N T E
T
37 P
R O L O G

Segunda parte

Lenguaje C

2. Lenguaje C

2.1. Antecedentes

Ejercicio 1

Instrucciones: Para cada una de las afirmaciones siguientes escriba dentro de los paréntesis la letra "F" si la afirmación es falsa, o escriba la letra "V" si la afirmación es verdadera.

1. Dennis Ritchie desarrolló el lenguaje de programación C. **( )**
2. No todos los compiladores soportan ANSI C. **( )**
3. La portabilidad es una gran ventaja del lenguaje C. **( )**
4. Podemos decir que el lenguaje C forma parte de C++. **( )**
5. El Lenguaje C permite crear programas orientados a objetos. **( )**
6. C es el lenguaje predominante en el entorno UNIX, desde sus inicios. **( )**
7. Java también se basa en la sintaxis de C. **( )**
8. El lenguaje C++ no permite el paradigma orientado a objetos. **( )**
9. El lenguaje C fue creado en los laboratorios Edison. **( )**
10. El encapsulamiento es una desventaja de la programación en el lenguaje C. **( )**

Ejercicio 2

Instrucciones: Lea con atención cada uno de los enunciados que enseguida se muestran. De las opciones enlistadas, subraye solamente el inciso cuya definición corresponda correctamente con el enunciado.

1.- El lenguaje de programación C fue creado originalmente para el sistema operativo:

a) Linux b) MS – DOS c) Unix d) Ensamblador

2.- El lenguaje del que deriva C es:

a) El lenguaje FORTRAN b) El lenguaje B c) El lenguaje Pascal d) El lenguaje COBOL

3.- El lenguaje B fue creado por:

a) Dennis Ritchie b) Ken Thompson c) Brian Kernighan d) Bjarne Stroustrup

4.- Creador del lenguaje C++:

a) Dennis Ritchie b) Brian Kernighan c) Ken Thompson d) Bjarne Stroustrup

5.- Variante del lenguaje C, creada por la compañía Microsoft:

a) C b) C++ c) ANSI C d) C#

6.- Versión estándar del lenguaje C:

a) C b) C++ c) ANSI C d) C#

7.- Desarrollador del lenguaje BCPL:

a) Martin Richards b) Ken Thompson c) Brian Kernighan d) Bjarne Stroustrup

8.- Lenguaje basado en C:

a) Pascal b) Java c) Delphi d) Prolog

9.- Implementación del lenguaje C, creada para incluir el desarrollo de la interfaz de usuario:

a) C b) C++ c) Visual C d) ANSI C

10.- Versión de C que tiene sus raíces en C, C++ y Java:

a) C b) C# c) C++ d) Visual C e) ANSI C

Ejercicio 3

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Ahorro de memoria	( )	1. Es un predecesor del lenguaje C, de bajo nivel.
b) Bajo nivel	( )	2. Una de las grandes ventajas del lenguaje ensamblador, que le permite trabajar directamente con los recursos de la máquina.
c) Lenguaje C++	( )	3. Ventajas que permiten agrupar el código en funciones.
d) Consumo excesivo de tiempo y esfuerzo en el desarrollo	( )	4. Lenguaje que proporciona la capacidad para la programación orientada a objetos.
e) Portabilidad	( )	5. Enlace dinámico y sobrecarga de funciones y operadores.
f) Son características del lenguaje C++	( )	6. A través de él, se permite el acceso a direcciones de memoria.
g) Popularidad	( )	7. Esta ventaja permite utilizar varias veces una función, ahorrando y disminuyendo código en el programa.
h) Estructura y modularidad	( )	8. Característica del lenguaje C que permite ejecutar un programa en una PC o en un equipo Macintosh, realizando algunas modificaciones en el código fuente.
i) Apuntador	( )	9. Desventaja durante el diseño y mantenimiento de funciones y clases.
j) Lenguaje BCPL	( )	10. Una de las ventajas del lenguaje C, ya que existe una gran variedad de compiladores, librerías y herramientas de apoyo a la programación.

2.2. Estructura general de un programa

Ejercicio 4

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Variables globales	Palabras reservadas	else	Identificador
Instrucción	Operadores	Comentarios	Llaves
Variable	Constante		

1.- Las _____ son las variables que se declaran fuera de la función *main()*.

2.- Un _____ es utilizado para referirnos a un valor constante, a una variable o a una estructura de datos.

3.- _____ es una palabra reservada en lenguaje C.

4.- Una _____ es una orden completa para la computadora y termina con punto y coma.

5.- El cuerpo principal de una función está constituido por todas sus líneas de código fuente, y se delimita con símbolos de _____.

6.- Las _____ son aquellas que no se pueden utilizar como identificador, dentro de un programa en C.

7.- Una _____ es un identificador o nombre que hace referencia a un valor que no debe cambiar durante la ejecución del programa.

8.- Los _____ son componentes sintácticos o *tókens* del vocabulario del lenguaje C.

9.- Los _____ sirven para documentar cómo está diseñado el programa.

10.- La _____ es un identificador o nombre de un valor que puede cambiar durante la ejecución de un programa.

2.3. Metodología para el desarrollo de un programa

Ejercicio 5

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis, la letra correspondiente al inciso que indique la relación correcta.

a) Archivos de biblioteca	( )	1. Cuando utilizamos estos archivos los encerramos particularmente entre < y >. Además, se encuentran en el compilador o en el sistema operativo.
b) Prototipo de una función	( )	2. Permite definir símbolos. Su uso principal es sustituir un texto por otro.
c) Bibliotecas de funciones	( )	3. Permiten al compilador leer como entrada una serie de definiciones disponibles en un archivo.
d) Directiva **#include**	( )	4. Son las instrucciones básicas o conjunto de extensiones, que van al inicio de un programa.
e) **stdio.h**	( )	5. Son los archivos de definición de funciones utilizadas en un programa en lenguaje C.
f) Librería estándar	( )	6. Son las órdenes que el preprocesador debe compilar, antes de que se compile el código fuente. Se encuentran en la cabecera de un programa y les antecede el símbolo #.
g) Instrucciones de inclusión	( )	7. Archivo de cabecera que incluye las funciones de entrada y salida.
h) Directiva **#define**	( )	8. Declaración de una función.
i) Directivas del preprocesador	( )	9. En ella se encuentran las funciones y tipos de datos especiales del compilador, las operaciones de entrada y salida así como las matemáticas y otras.
j) Archivos de cabecera de la librería estándar	( )	10. Permite incluir en un archivo de código fuente otros archivos de texto.

Ejercicio 6

Instrucciones: Enliste correctamente las partes que correspondan a la cabecera y al cuerpo de un programa, tomando en consideración la lista de enunciados que se presenta del lado derecho.

Cabecera	Funciones propias del programa Programa principal (*main*) Directivas del preprocesador Prototipos de función
Cuerpo del programa	Declaración de variables y constantes locales Instrucciones o acciones Estructuras de control Estructuras secuenciales Declaración de variables globales

2.4. Ciclo de desarrollo

Ejercicio 7

Instrucciones: Ordene correctamente, de manera cronológica, las etapas para el desarrollo de un programa escrito en lenguaje C. Además, enliste los tipos de errores que se pueden presentar durante el desarrollo del programa. Tome en consideración la lista de enunciados que se presenta del lado derecho.

Etapas	Lógicos Ejecución Edición Carga Diseño Sintaxis
Errores	Compilación Análisis Enlace Avisos Depuración Preproceso

Ejercicio 8

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) .cpp, .cc, .c y .C	( )	1. En esta etapa se crea el código fuente de un programa escrito en lenguaje C.
b) Ejecutable	( )	2. Etapa en la que se interpretan y ejecutan todas las directivas del lenguaje C.
c) Edición	( )	3. Si el programa presenta una falla al ejecutarse, en esta etapa se corrige, volviendo a la etapa de edición posteriormente.
d) .exe	( )	4. Extensiones posibles del nombre del archivo que se genera en la etapa de edición.
e) Código objeto	( )	5. Programa final utilizado por el usuario.
f) Enlazador	( )	6. Realiza el análisis sintáctico, léxico y semántico de un código fuente; generando, además, un código en lenguaje ensamblador.
g) Preproceso	( )	7. Código resultante de la compilación.
h) Carga	( )	8. Extensión del nombre del archivo de un programa ejecutable, para el sistema operativo Windows.
i) Compilador	( )	9. Agrupa los archivos objeto y edita los enlaces, con la finalidad de generar un archivo ejecutable.
j) Depuración y prueba	( )	10. En esta etapa se toma el programa ejecutable y se transfiere a la memoria, antes de ejecutarse.

CRUCIGRAMA

HORIZONTALES

1.- La estructura _____ se forma escribiendo las instrucciones una detrás de la otra.

4.- Función principal de un programa escrito en lenguaje C.

6.- Una estructura de _____ se basa en expresiones condicionales para determinar el flujo de la secuencia de instrucciones o acciones.

7.- Componentes sintácticos del lenguaje C.

9.- Se encuentra dentro de un archivo de cabecera.

10.- Código resultante de la compilación.

12.- El fichero _____.h contiene las definiciones y los prototipos de las funciones de la librería estándar, para la manipulación de las cadenas de caracteres.

13.- Una de las palabras reservadas del lenguaje C.

14.- Las _____ del preprocesador son las órdenes que se interpretan antes que el código fuente sea compilado.

16.- Etapa donde se pasa el programa ejecutable a la memoria.

17.- Una de las directivas del preprocesador.

18.- Contiene una serie de instrucciones que realizan una tarea específica y puede ser llamada las veces que sea necesario.

VERTICALES

2.- Programa final entregado al usuario.

3.- El fichero _____.h contiene las definiciones y los prototipos de las funciones de la librería estándar, para el manejo de la entrada y la salida.

4.- Cadena de caracteres identificada por un nombre con la directiva **#define**, en la cabecera de un programa.

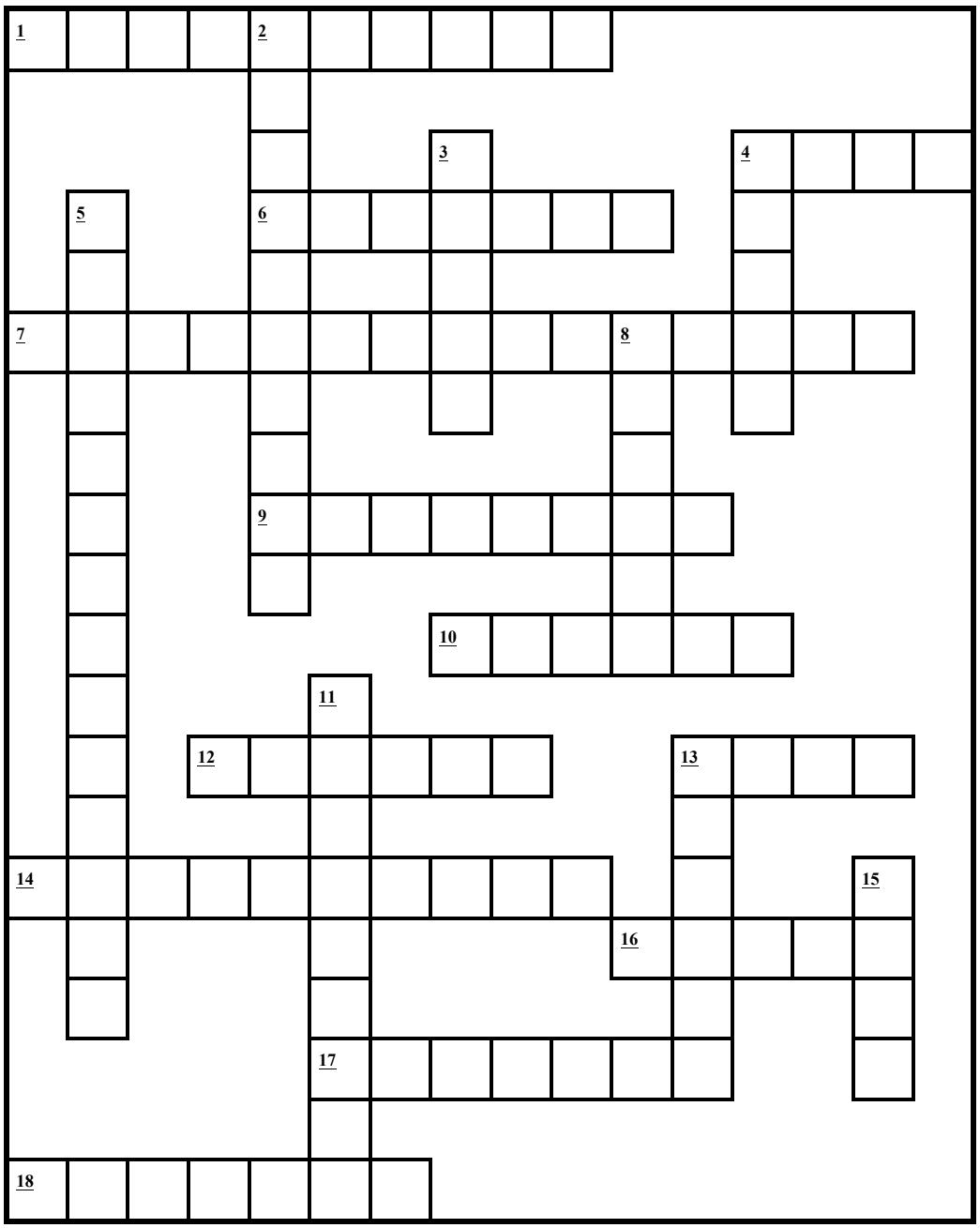
5.- Dividir un programa complejo en partes más pequeñas y manejables.

8.- Directiva del preprocesador para definir constantes simbólicas o macros.

11.- El _____ de una función es su definición sin cuerpo y terminada en punto y coma.

13.- Tipo de error creado por funciones que no son definidas en ningún archivo objeto y en ninguna librería.

15.- El fichero _____.h contiene las definiciones y los prototipos de las funciones de la librería estándar, relacionadas con las matemáticas.



APÉNDICE

Ejercicio 1

1.- (V); 2.- (V); 3.- (F); 4.- (V); 5.- (F); 6.- (V); 7.- (V); 8.- (F); 9.- (F); 10.- (F)

Ejercicio 2

1.- c); 2.- b); 3.- b); 4.- d); 5.- d); 6.- c); 7.- a); 8.- b); 9.- c); 10.- b)

Ejercicio 3

1.- (j); 2.- (b); 3.- (h); 4.- (c); 5.- (f); 6.- (i); 7.- (a); 8.- (e); 9.- (d); 10.- (g)

Ejercicio 4

1.- Variables globales; 2.- Identificador; 3.- else; 4.- Instrucción; 5.- Llaves; 6.- Palabras reservadas; 7.- Constante; 8.- Operadores; 9.- Comentarios; 10.- Variable

Ejercicio 5

1.- (j); 2.- (h); 3.- (g); 4.- (c); 5.- (a); 6.- (i); 7.- (e); 8.- (b); 9.- (f); 10.- (d)

Ejercicio 6

CABECERA

En cualquier orden:
1.- Directivas del preprocesador
2.- Prototipos de función
3.- Declaración de variables globales

CUERPO DEL PROGRAMA

En cualquier orden:
1.- Programa principal (*main*)
2.- Declaración de variables y constantes locales
3.- Instrucciones o acciones
4.- Estructura secuencial
5.- Estructura de control
6.- Funciones propias del programa

Ejercicio 7

Etapas

1.- Análisis
2.- Diseño
3.- Edición
4.- Preproceso
5.- Compilación
6.- Enlace
7.- Carga
8.- Ejecución
9.- Depuración

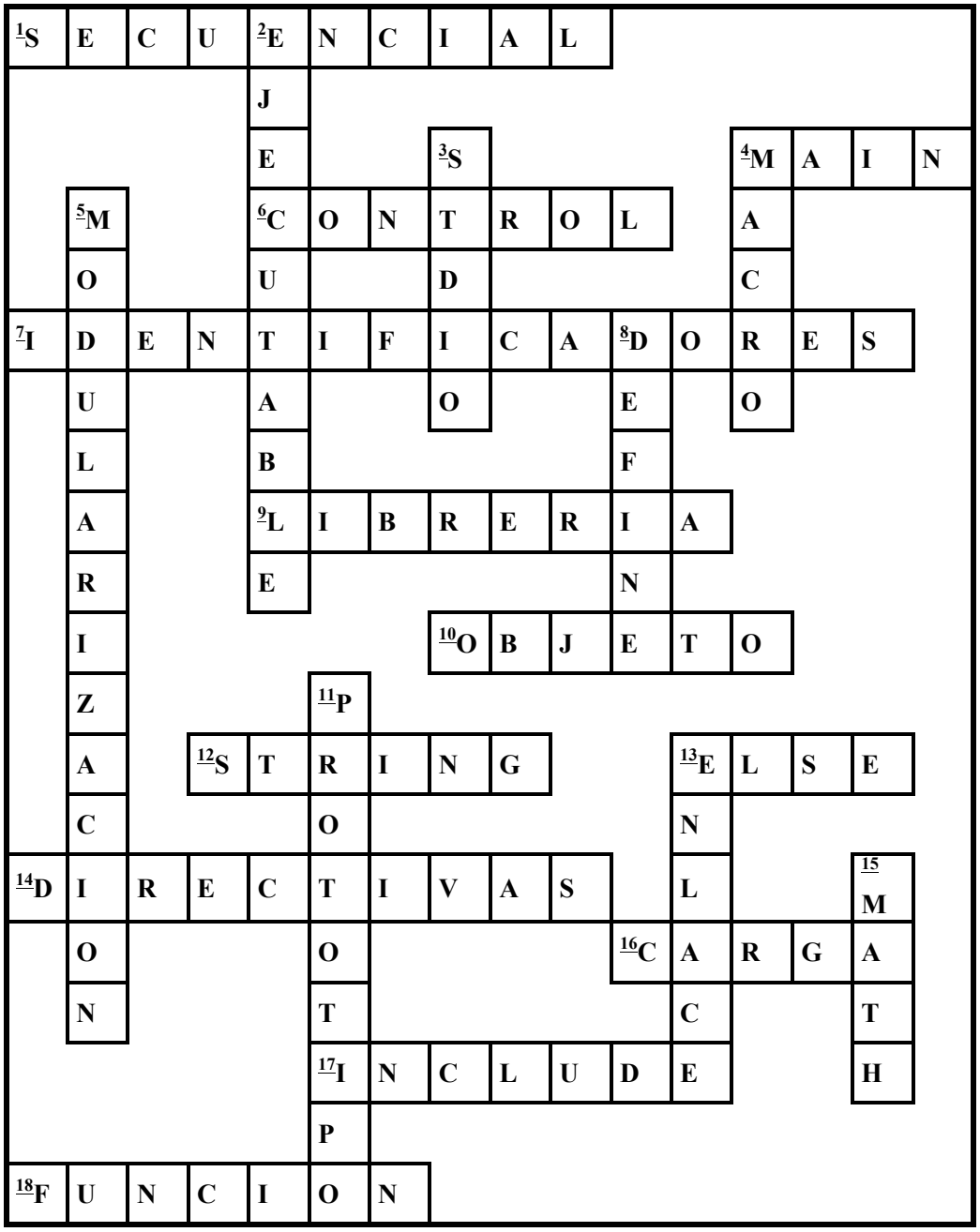
Errores

En cualquier orden:
a) Sintaxis
b) Avisos
c) Lógicos

Ejercicio 8

1.- (c); 2.- (g); 3.- (j); 4.- (a) ; 5.- (b); 6.- (i); 7.- (e); 8.- (d); 9.- (f); 10.- (h)

CRUCIGRAMA


1S E C U 2E N C I A L
J
E 3S 4M A I N
5M 6C O N T R O L A
O U D C
7I D E N T I F I C A 8D O R E S
U A O E O
L B F
A 9L I B R E R I A
R E N
I 10O B J E T O
Z 11P
A 12S T R I N G 13E L S E
C O N
14D I R E C T I V A S L 15M
O O 16C A R G A
N T C T
17I N C L U D E H
P
18F U N C I O N

Tercera parte

Tipos de datos, variables y constantes

3. Componentes del lenguaje C

3.1. Tipos de datos, variables y constantes

Ejercicio 1

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Constantes	Tipo de dato	Char	Locales
Variables	Short	Float	Declaración
Nombre	Apuntador	Reservadas	

Enunciados:

1.- Las _____ son datos definidos, cuyo valor no cambia durante la ejecución del programa.

2.- El modificador _____ es un modificador de la declaración de tipo, que reduce el espacio de almacenamiento en memoria de los enteros.

3.- Para declarar una variable se le debe asignar un _____ y un _____.

4.- La declaración ***int *punt1;*** hace referencia a una variable de tipo _____.

5.- Las _____ son los elementos más utilizados para almacenar datos, su valor puede reasignarse durante la ejecución del programa.

6.- El tipo de variable _____, hace referencia a un valor de tipo carácter.

7.- Las palabras clave o _____ no pueden utilizarse como nombres de variables.

8.- El tipo de variable _____ es numérico, con valor decimal.

9.- La _____ de variables y constantes se debe realizar antes de su utilización en el programa.

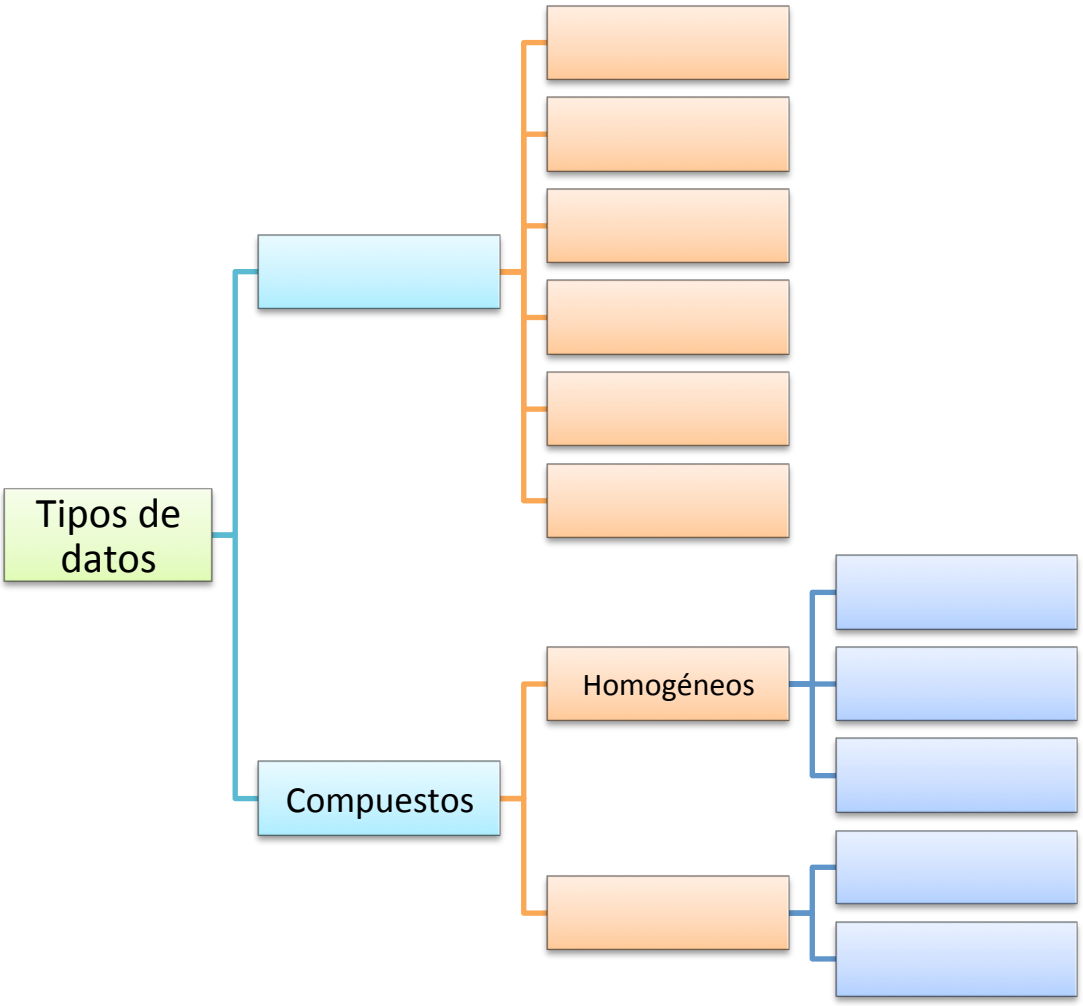
10.- Las variables _____ están ubicadas dentro del programa principal *main()* o alguna otra función.

Ejercicio 2

Instrucciones: Complete de forma correcta el diagrama siguiente referido a los tipos de datos. Para tal fin, escriba una de las respuestas que se proporcionan a continuación, dentro de cada uno de los recuadros vacíos en el diagrama.

Respuestas:

Entero	Arreglos	Booleano	Simples	Uniones
Cadenas	Apuntador	Flotante	Arreglos >1 dimensión	
Heterogéneos	Estructuras	Carácter	Doble precisión	



3.2. Identificadores y palabras reservadas

Operadores, precedencias y evaluaciones en cortocircuito

Ejercicio 3

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis, la letra correspondiente al inciso que indique la relación correcta.

a) Operador de asignación	( ) 1. Realizan operaciones matemáticas como suma, resta, multiplicación, división, etc.
b) for	( ) 2. Conjunto de variables, constantes y operadores.
c) Expresión	( ) 3. Palabra reservada para declarar un tipo de variable o constante.
d) Evaluación en cortocircuito	( ) 4. Nombres de las funciones, las variables o las constantes del programa. Su primer carácter puede ser una letra o un guión bajo pero no puede contener caracteres especiales.
e) Operadores lógicos	( ) 5. Modificador de los tipos de variables y constantes enteras.
f) int	( ) 6. Palabra reservada utilizada en instrucciones de repetición.
g) Identificadores	( ) 7. Palabra reservada del lenguaje C, empleada para salir de una instrucción de repetición.

h) long	( ) 8. Asigna un valor a una variable.
i) break	( ) 9. Tipo de optimización realizada por un compilador o un intérprete, en el cual se evalúa el primer operando de una operación booleana y, si el resultado de esta evaluación determina la solución de la expresión completa, ya no se evalúa el segundo operador.
j) Operadores aritméticos	( ) 10. Permiten la comparación entre variables y constantes

3.3. Operadores, precedencias y evaluaciones en cortocircuito

Conversiones

Ejercicio 4

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Escriba una de las respuestas, que se enlistan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Precedencia	De incremento	Variables estáticas	Bits
Lógicos	Asignación	Conversiones	
Relacionales	Casting	Bloque de instrucciones	

1.- La _____ es el orden de prioridad que tienen los operadores en la evaluación, dentro de la ejecución de un programa.

2.- Las _____ pueden cambiar el tipo de valor almacenado en una variable.

3.- En una operación de _____ podemos cambiar el tipo de valor de una variable.

4.- El _____ es la conversión clara y precisa de un tipo de variable a otro.

5.- Las _____ tienen memoria asignada durante la ejecución del programa, su valor permanece aunque la función donde esté definida acabe y se vuelva a ejecutar.

6.- Los operadores _____ van aumentando el valor de la variable.

7.- Los operadores _____ calculan valores de falso o verdadero.

8.- Los operadores de manejo de _____ permiten manipular las variables bit a bit.

9.- Un _____ es el conjunto de instrucciones delimitado por símbolos de llaves.

10.- Los operadores _____ utilizan los símbolos <, >, <= y >=; entre otros.

3.4. Conversiones

Ejercicio 5

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis, la letra correspondiente al inciso que indique la relación correcta.

a) Instrucciones	( ) 1. Se utilizan para marcar el inicio y fin de una función en lenguaje C.
b) Carácter de escape	( ) 2. Palabra reservada que es un especificador de tipo.
c) "%f"	( ) 3. Función utilizada para terminar un programa.

d) Instrucción	( ) 4. Palabra reservada empleada en una instrucción de control condicional.
e) Llaves "{" y "}"	( ) 5. A la diagonal invertida ('\'), se le conoce como:
f) printf( )	( ) 6. Línea de comando terminada por punto y coma.
g) exit( )	( ) 7. Conforman el cuerpo del programa y también se les conoce como sentencias.
h) if	( ) 8. Carácter de salto de línea.
i) '\n'	( ) 9. Función de impresión en pantalla.
j) void	( ) 10. Especifica un valor de punto flotante en la función printf( ).

3.5. Interacción con el usuario

Ejercicio 6

Instrucciones: Relacione correctamente cada uno de los conceptos enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba sobre la línea en blanco, la respuesta que indique la relación correcta.

printf() 1.- _____ Descriptor del archivo para escribir mensajes de error.

"%s"	2.- _____ Descriptor del archivo de salida.
"%d"	3.- _____ Descriptor del archivo de entrada.
stderr	4.- _____ Función de entrada de datos que lee un carácter a la vez.
stdin	5.- _____ Función utilizada para imprimir la salida en pantalla.
"%c"	6.- _____ Formato para impresión de un carácter.
putc()	7.- _____ Formato para impresión de un número entero.
getch()	8.- _____ Formato para impresión de una cadena de caracteres.
stdout	9.- _____ Formato para impresión de un número de punto flotante.
"%f"	10.- _____ Función, diferente a printf(), que imprime un carácter en la salida estándar.

3.6. Estructuras de control

Ejercicio 7

Instrucciones: Relacione correctamente cada uno de los conceptos enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba sobre la línea en blanco, la respuesta que indique la relación correcta.

break	1.- _____ Ejecuta un bloque de instrucciones mientras la expresión sea correcta.
return	2.- _____ Ciclo que tiene una instrucción de inicialización, otra que verifica si la expresión es verdadera y otra más de incremento/decremento.
if	

for

if – else

goto

continue

do – while

while

switch

Ciclos anidados

3.- _____ Coloca la condición de repetición al final del ciclo, primero ejecuta las instrucciones en el cuerpo del ciclo y, posteriormente, verifica si se debe repetir.

4.- _____ Colocar un ciclo dentro de otro.

5.- _____ Control de flujo que ejecuta un bloque de instrucciones si se cumple la condición de la expresión.

6.- _____ Control de flujo que ejecuta un bloque de instrucciones si se cumple la condición, si no, ejecuta un bloque alternativo de instrucciones.

7.- _____ Control de flujo que selecciona (según la expresión) uno de entre varios bloques de instrucciones.

8.- _____ Se utiliza para salir de la instrucción switch.

9.- _____ Instrucción que salta al final del ciclo y fuerza la comprobación de la condición de repetición.

10.- _____ Salto incondicional al nombre de etiqueta que se le indica.

11.- _____ Retorno de función.

Ejercicio 8

Instrucciones: Complete correctamente la sintaxis de las plantillas de instrucción mostradas en la columna de la izquierda. Para tal fin, escriba en cada una de las líneas en blanco, uno de los enunciados proporcionados en la columna de la derecha.

_____ (_____)
{
_____;
 	
}
	Instrucciones o sentencias
	
	while
	
	expresión

_____ {	Instrucciones o sentencias

_____; }_____ (_____)	while do expresión

_____ (_____; _____; _____)
{
_____;
}
	Instrucciones o sentencias
	Condición
	for
	inicialización
	actualización

_____ (_____)
{
_____;
}
	Instrucciones o sentencias
	Expresión
	if

_____ (_____)
{
_____;
}
_____
{
_____;
}
	expresión
	Instrucciones o sentencias (condición falsa)
	else
	Instrucciones o sentencias (condición verdadera)
	if

_____ (_____) { _____: _____;	Instrucciones o sentencias 2 break case 1

_____; _____: _____; _____; _____: _____; _____; }	break switch case 2 break Instrucciones o sentencias 1 Expresión default Instrucciones o sentencias 3

CRUCIGRAMA

HORIZONTALES

1.- Control de flujo que selecciona (según la expresión) uno de entre varios bloques de instrucciones.

5.- Se utiliza para salir de la instrucción switch.

7.- Si el bloque de instrucciones correspondiente a la instrucción if no se ejecuta, se ejecuta el bloque de instrucciones que corresponde a esta instrucción.

8.- Instrucción que se ejecuta cuando ninguna opción satisface la expresión condicional en la instrucción switch.

10.- Salto incondicional al nombre de etiqueta que se le indica.

11.- Instrucción que salta al final del ciclo y fuerza la comprobación de la condición de repetición.

14.- Repetición de un bloque de instrucciones.

15.- Retorno de función.

VERTICALES

2.- Instrucción que ejecuta un bloque de instrucciones mientras la expresión sea correcta.

3.- Función que permite imprimir mensajes en pantalla.

4.- Opción válida de la instrucción de control switch.

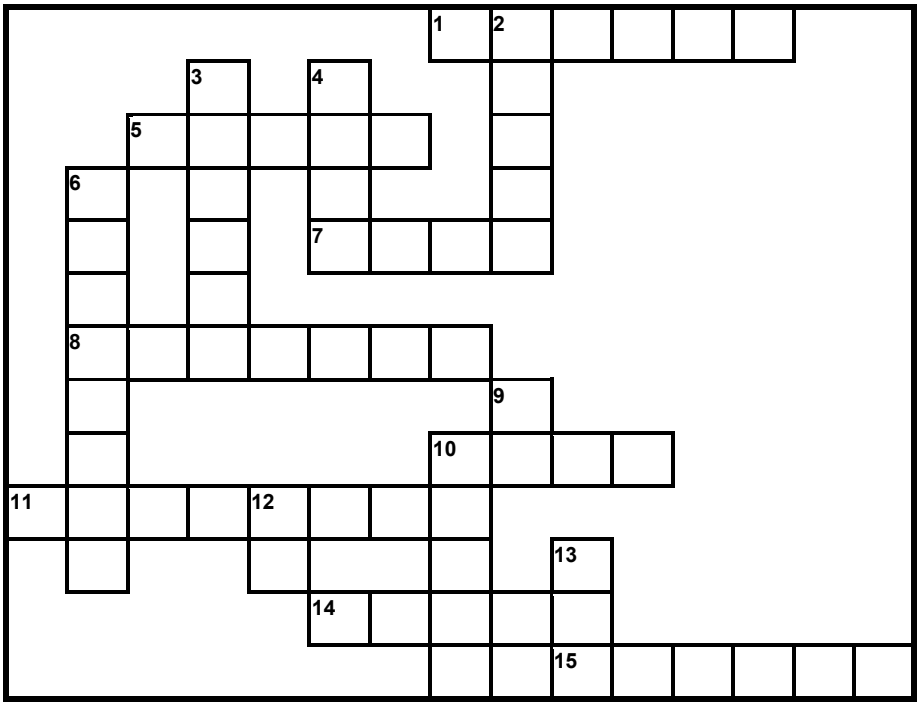
6.- Ciclos colocados dentro de otro ciclo.

9.- Se utiliza junto con while.

10.- Función de entrada de datos.

12.- Control de flujo que ejecuta un bloque de instrucciones si se cumple la condición de la expresión.

13.- Ciclo que tiene una instrucción de inicialización, otra que verifica si la expresión es verdadera y otra más de incremento/decremento.



3.7. Ejercicios prácticos

Para simplificar la revisión de este documento, se aclara que los ejercicios están alternados, es decir, en el apéndice de este apartado se presenta la solución de un ejercicio y se propone que el siguiente sea resuelto por el lector.

El significado de la nomenclatura utilizada, por ejemplo "Ejercicio 2.4.3", tiene la explicación siguiente: el primer número corresponde al apartado; el segundo corresponde al tema específico; y, el tercero, es el número de ejercicio, es decir, apartado 2, tema 4 y ejercicio o programa número 3.

Por otra parte, hay dos tipos de prácticas, las prácticas *normales* y las de *paso a paso*. Para las prácticas normales se recomienda utilizar los ejercicios propuestos para que sean resueltos por el lector y, para las prácticas paso a paso, guiarse con los ejercicios resueltos, solo se debe seguir el procedimiento siguiente:

1. Encender el equipo de cómputo.
2. Abrir sesión del ambiente gráfico Windows.
3. Ejecutar la aplicación que se tenga instalada para editar los programas escritos en lenguaje C.
4. Transcribir el programa en cuestión.
5. Guardar el archivo que contiene el programa transcrito.
6. Compilar el programa.
7. Iniciar un proceso de depuración del programa (corrección de errores) hasta cerciorarse de que el programa cumpla con el objetivo establecido.
8. Imprimir el resultado de la ejecución o ejecuciones del programa (paso opcional).

Ejercicio 3.7.1: Elaborar un programa en lenguaje C que calcule la edad de una persona, conociendo su año de nacimiento.

a) **Análisis:** El resultado que se pide es la EDAD de la persona en años, entonces, como ya se conoce el año actual, lo único que se requiere es solicitar un dato adicional. Esto es ¿en qué año nació la persona? Con eso se tiene la información necesaria para hacer el cálculo. Además, considere que el año se debe manejar a 4 dígitos, tanto el año actual como el año en el que nació la persona, esto para el funcionamiento correcto del programa.

b) **Codificación:** Para elaborar el programa se sugiere emplear las instrucciones para lectura de datos por teclado y de salida por pantalla. Respecto a la declaración de variables, solamente es necesaria una y del tipo entero; porque el año actual se puede manejar directamente como valor tecleado dentro del código fuente, al momento de hacer el cálculo, o bien, opcionalmente se puede declarar una segunda variable e inicializarla directamente con el dato que ya se conoce.

c) **Comprobación**: Por último, el lector puede probar el programa con sus propios datos, es decir, introduciendo su año de nacimiento y, así, verificar que efectivamente el programa indique su EDAD correcta. Luego se puede probar con la edad de algún compañero o

compañera. Finalmente, ¿qué resultado indica el programa si, como año de nacimiento, se pone 1974?

Ejercicio 3.7.2: Elaborar un programa en lenguaje C, que calcule la antigüedad de una persona, conociendo en qué año ingresó a la empresa.

a) **Análisis:** El resultado que se pide es la ANTIGÜEDAD del empleado en años, entonces, como ya se conoce el año actual, lo único que se requiere es solicitar un dato adicional más. Esto es, en qué año comenzó a trabajar en la empresa el empleado; y con eso se tiene la información necesaria para hacer el cálculo. Además, considere que el año se debe manejar a 4 dígitos, tanto el año actual como el año en el que comenzó a laborar la persona, esto para el funcionamiento correcto del programa.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla. Respecto a la declaración de variables, solamente es necesaria una, del tipo entero, para el año en el que ingresó el empleado a la empresa o institución, porque el año actual se puede manejar directamente como valor tecleado dentro del código fuente, al momento de hacer el cálculo, o bien, opcionalmente se puede declarar una segunda variable e inicializarla directamente con el dato que ya se conoce.
c) **Comprobación**: Por último, se puede probar el programa con los datos de un profesor o el padre del lector, es decir, introduciendo los datos proporcionados por el profesor y, así, verificar que efectivamente la antigüedad sea la correcta. Luego se puede probar con los datos de alguno de los padres. Finalmente, ¿Qué resultado indica el programa si, como año de ingreso, se pone 1974?

Ejercicio 3.7.3: Calcular el precio de un terreno, suponiendo que se conoce el precio por metro cuadrado y las medidas del terreno. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** El resultado que se pide es el precio de un terreno, para esto se requiere hacer varias preguntas al usuario del programa. Esto es, las medidas del terreno y el precio del metro cuadrado. Aquí lo importante es la declaración adecuada de variables, para que se almacenen correctamente los datos y se logre el cálculo acertado del programa.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una vez por cada pregunta. Respecto a la declaración de variables, aquí se van a requerir al menos cinco; con la ventaja de que todas pueden ser del mismo tipo. Se realizará un mal manejo de memoria, ya que unas variables van a guardar cifras pequeñas (como las medidas del terreno) y, otras, valores muy grandes (como el precio del terreno). Respecto a los cálculos, puede hacerse una operación, sin embargo, se recomienda calcular primero el

área del terreno y, posteriormente, obtener el valor del terreno con el precio por metro cuadrado.

c) **Comprobación**: Finalmente, se puede ejecutar el programa con datos ficticios o reales. En el caso de los reales, se pueden ver anuncios en el periódico en la sección de venta de terrenos y, con esos datos, probar el programa. Ahora bien, para comprobar las operaciones, se pueden manejar cantidades pequeñas o múltiplos de 10 y eso permitirá saber si las operaciones son las correctas. Por ejemplo, si se dice que el terreno mide 10 x 10 metros, eso da como resultado $100m^2$; y si se considera que el precio por metro cuadrado es de $1.00 peso, el resultado final deberá ser $100.00 pesos. Intente con esta última información y verifique la exactitud del programa.

Ejercicio 3.7.4: Calcular el precio de una casa, suponiendo que se puede conocer el precio del metro cuadrado de construcción y las medidas de la casa. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** El resultado que se pide es el precio de una casa, por lo que se requiere hacer varias preguntas al usuario del programa. Esto es, las medidas de la casa y el precio del metro cuadrado de construcción. Aquí lo importante es la declaración adecuada de variables, para que se almacenen correctamente los datos y se logre el cálculo correcto del programa.

b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables, aquí se van a requerir al menos cinco; con la ventaja de que todas pueden ser del mismo tipo. Se realizará un mal manejo de memoria, ya que unas van a guardar cifras pequeñas (como las medidas del terreno que ocupa la casa) y, otras, valores muy grandes (como el precio de metro cuadrado de construcción). Así que valore esta consideración, para mejor declarar al menos dos tipos de variables. Respecto a los cálculos, puede hacerse una sola operación, sin embargo, se recomienda calcular primero el área de la casa y, posteriormente, obtener el valor de la vivienda con el precio por metro cuadrado de construcción.

c) **Comprobación**: Finalmente, se puede ejecutar el programa con datos ficticios o reales. En el caso de los reales, se pueden ver anuncios en el periódico en la sección de venta de casas y llamar para solicitar las medidas, con esos datos se puede probar el programa. De otra forma, usando datos ficticios, se pueden comprobar las operaciones al manejar cantidades pequeñas o múltiplos de 10, eso permitirá saber fácilmente si las operaciones son las correctas. Por ejemplo, si se dice que la casa mide 10 x 10 metros, eso da como resultado 100 m^2; y si se considera que el precio por metro cuadrado de construcción es de un $1.00 peso, el resultado final deberá ser $100.00 pesos. Intente con esta última información y verifique la exactitud del programa.

Ejercicio 3.7.5: Generalmente muchos artículos electrónicos tales como computadoras, laptops, televisores LCD, entre otras cosas; se cotizan en dólares. Elabore un programa en lenguaje C que convierta de dólares a pesos, a partir de conocer la cantidad en dólares, para que facilite al usuario conocer el precio de los artículos electrónicos, en moneda nacional.

a) **Análisis:** En concreto, el propósito es convertir dólares a pesos, para ello necesitamos conocer la cotización del dólar, aparte del precio del producto. Además de la declaración correcta de las variables, para que la información se almacene en memoria apropiadamente, se requieren al menos dos tipos diferentes de variables, uno que maneje cantidades cerradas y, otro, con decimales.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables y tomando en cuenta la última consideración del análisis, será necesario contar con dos variables que almacenen cantidades con decimales, para que no haya pérdida de centavos en el cálculo y, lógicamente, errores en el resultado.
c) **Comprobación**: Para probar el programa, al momento de la ejecución, suponga que un artículo cuesta $100 dólares, y que el dólar está en $10 pesos. Anote el resultado. Ejecute el programa nuevamente y ahora suponga que el artículo cuesta $1,000 dólares, escribiendo la misma cotización de $10 dólares. Anote el segundo resultado. El segundo resultado deberá ser igual al primero, pero con un cero adicional. Intente con esta información y verifique la exactitud del programa.

Ejercicio 3.7.6: Actualmente hay navegadores GPS que permiten circular en ciudades extrañas y llegar a los sitios de interés sin mayores contratiempos. Colabore con este tipo de aplicaciones para que se les incorpore una opción para convertir millas a kilómetros y que, de esa forma, se evite el que sean multados los extranjeros o conductores, cuyo auto maneje la velocidad en millas por hora. Elabore el programa en lenguaje C, partiendo de que 1 milla equivale a 1.6 km.

a) **Análisis:** En resumen, se pide convertir millas a kilómetros, a diferencia del ejercicio anterior, aquí ya se sabe la equivalencia entre millas y kilómetros. Esto simplifica la solicitud de datos, ya que únicamente se requiere solicitar la velocidad en millas a la que viaja la persona.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables, hay que tomar en cuenta la conversión de tipos que hace el lenguaje C, esto porque la equivalencia cuenta con decimales y, si no se cuida este detalle, al momento de declarar la variable que almacene ese dato, se pueden provocar errores en el resultado.
c) **Comprobación**: Para probar el programa, al momento de la ejecución suponga que la velocidad es de 100 millas por hora, por lo que el resultado deberá ser 160 km/h. Ejecute

el programa nuevamente y ahora suponga que la velocidad es de 10 millas por hora, ¿Cuál será el resultado con estos datos?

Ejercicio 3.8.1: Muchas tiendas ofrecen promociones donde aplican descuentos sobre descuentos, generando una guerra de precios y de *rebajas*. Aunque los teléfonos celulares traen calculadora, muchas veces a la gente se le complica realizar estos cálculos. Entonces, elabore un programa en lenguaje C que calcule el precio final de un producto, conociendo el precio inicial y el porcentaje de descuento que, con algunos arreglos mínimos, posteriormente podría ser incorporado a dichos celulares. Únicamente se preguntaría el precio inicial y el porcentaje de descuento, para luego informar el precio final y la cantidad *ahorrada*.

a) **Análisis:** Revisando el ejercicio se puede notar, directamente, que se requiere la solicitud de dos cosas: el precio del producto y su porcentaje de descuento respectivo. Si bien es cierto que el cálculo se puede hacer en una sola operación, hay una diferencia con respecto a los ejercicios previos, donde solamente se pedía un resultado; aquí se están pidiendo dos, uno es cuántos pesos se está ahorrando la persona y, el otro dato, es el precio final de la prenda; es decir, una vez aplicado su descuento correspondiente.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias, una por cada pregunta. Respecto a la declaración de variables, hay que tomar en cuenta la conversión de tipos que hace el lenguaje C y, esto, porque muchas de las veces el precio se maneja en pesos y centavos; si no se cuida ese detalle al momento de declarar la variable que almacenará esos datos, se pueden provocar errores en el resultado. Otra consideración es que, al menos, será necesario realizar dos operaciones, una donde se obtenga el descuento y otra para el precio final del artículo.
c) **Comprobación**: Al ejecutar el programa suponga que un artículo cuesta $100 pesos y cuenta con un descuento del 10%, ¿Cuáles son los resultados con esos datos? ¿Los resultados de su programa son $10 pesos de descuento y el precio final es de $90 pesos? Si es así, pruebe el programa nuevamente con datos que vienen en la publicidad de las empresas.

Ejercicio 3.8.2: Elabore un programa en lenguaje C que calcule el sueldo final de un trabajador y, que como salida, muestre en cuántos pesos se incrementó el salario y cuál sería su sueldo final. Tomando como base el sueldo inicial y el porcentaje de incremento.

a) **Análisis:** Revisando el ejercicio se puede notar cierta similitud al ejercicio anterior. Si bien es cierto que el cálculo se puede hacer en una sola operación, nuevamente se pide presentar dos resultados; uno es cuántos pesos está ganando la persona de más y, el otro dato, es el sueldo final; es decir, con su correspondiente incremento salarial.

b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables, hay que tomar en cuenta nuevamente la conversión de tipos que hace el lenguaje C y, esto, porque al obtener el incremento salarial es necesaria una división; si no se cuida este detalle al momento de declarar la variable que almacenará esos datos, se pueden provocar errores en el resultado. Otra consideración es que, al menos, será necesario realizar dos operaciones, una donde se obtenga el incremento salarial y, otra, para el sueldo final del trabajador.

c) **Comprobación**: Al ejecutar el programa suponga que una persona gana $100 pesos y que su incremento salarial es del 10%, ¿Cuáles son los resultados con esos datos? ¿$10 pesos de incremento y el sueldo final es de $110 pesos? Si es así, ahora pruebe el programa nuevamente con datos que le pueda proporcionar un profesor o alguno de sus padres.

Ejercicio 3.8.3: Calcular el número máximo de fotos que le caben a una cámara digital, utilizando la máxima resolución permitida. Tomar en cuenta que la capacidad de las memorias es variable. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** Al leer el ejercicio, observamos que implícitamente viene una equivalencia, en otras palabras, podemos preguntar la capacidad en gigabytes de la cámara y también podemos solicitar la máxima resolución por foto que permite la cámara, pero aquí viene la pregunta ¿A cuántos bytes equivale un megapixel? Sin esta información, no se estaría en condiciones de resolver el ejercicio. Por lo que el primer paso es investigar ese dato y, posteriormente, realizar un par de operaciones aritméticas para obtener el resultado solicitado.

b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables, es necesario manejar al menos dos tipos distintos de datos, es decir, uno para almacenar primero los datos del tipo entero y, como esta solución requiere de un par de divisiones, entonces el otro tipo de variable tendría que ser del tipo *float*, para que se puedan conservar los decimales.

c) **Comprobación**: Al ejecutar el programa pida a un compañero que tenga teléfono celular con cámara, los datos de la resolución y la capacidad de la memoria, e infórmele el resultado para que él o ella le digan si es correcto. También lo que se puede hacer es conseguir información de las cámaras en publicidad donde las anuncian y que, generalmente, incluyen dichos datos.

Ejercicio 3.8.4: Calcular el número máximo de canciones en formato MP3 que le caben a un reproductor de música portátil. Tomar en cuenta que la capacidad de las memorias es variable y

considerar un tamaño promedio de las canciones. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** Este ejercicio tiene cierta similitud al ejercicio 3.8.3, pero con una diferencia importante porque aquí sí sabemos que 1 GB equivale a 1,024 megabytes. Hay una ligera complicación dado que el tamaño de las canciones es variable, por estar en relación con la duración, y sabemos que no todas las canciones duran lo mismo. Sin embargo, se nos pide considerar un tamaño promedio, lo que simplifica la solución, es decir, que podemos suponer que todas las canciones tienen 5 MB de tamaño, por lo que únicamente será necesario solicitar la capacidad del teléfono celular y, después de la operación correspondiente, podemos dar solución a este ejercicio.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables, es necesario manejar al menos dos tipos distintos de datos, es decir, uno para almacenar primero los datos del tipo entero y, como esta solución requiere del uso de divisiones, entonces el otro tipo de variable tendría que ser del tipo *float*, para que se puedan conservar los decimales.
c) **Comprobación**: Al ejecutar el programa pregunte a un compañero que tenga teléfono celular la capacidad de memoria del mismo y suponga que el tamaño por canción es de 5 MB. Sorpréndalo informándole cuántas canciones puede guardar en su teléfono, aproximadamente. También puede usar este programa si se quiere comprar un teléfono celular y que, probando con diferentes capacidades, vea cual le conviene más.

Ejercicio 3.8.5: ¿Cuántos discos DVD se necesitan para hacer un respaldo de una estación de radio, la cual tiene una amplia colección musical en discos compactos? Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** Este ejercicio trae implícita una tarea adicional y consiste en investigar, ya sea cuántos discos CD caben en un solo DVD, o bien, si a un disco CD le caben 700 MB, eso a ¿Cuántos GB equivale? El punto de partida es que sabemos que 1 GB equivale a 1,024 megabytes, resuelta la tarea anterior, estaremos en condiciones de dar solución a este ejercicio.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables, es necesario manejar al menos dos tipos distintos de datos, es decir, uno para almacenar primero los datos del tipo entero y como esta solución requiere del uso de divisiones, entonces el otro tipo de variable tendría que ser del tipo *float* para que se puedan conservar los decimales.
c) **Comprobación**: Al ejecutar este programa tome en cuenta la siguiente proporción, por cada 4 discos de CD se ocupará un DVD, pensando en que cada CD almacene 1 GB. Esto no es cierto porque los CD almacenan 700 MB, pero sirve como aproximación para

determinar si la solución proporcionada por el programa es correcta. Entonces, si se pone que la radio tiene 10 CDs, se ocuparán entre 2 y 3 DVDs para realizar el respaldo solicitado. Pero si se ponen 100 CDs, entonces es de pensar que, máximo, se ocuparán 30 DVDs para el respaldo. Recuerde que, lógicamente, la cifra real será mayor a la que estamos suponiendo.

Ejercicio 3.8.6: Calcular el número máximo de videos que caben en la memoria de un celular. Tomar en cuenta que la capacidad de las memorias es variable y considerar, también, un tamaño promedio de los videos. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** El propósito de este ejercicio es poner en práctica todos los conocimientos aplicados en los ejercicios previos. Así que la intención básica es reafirmar los diferentes elementos que conforman un programa, como son, instrucciones básicas para solicitar datos, declaraciones adecuadas en las variables con sus diferentes tipos y la manera correcta de hacer cálculos con las operaciones respectivas.
b) **Codificación:** Para elaborar el programa, se sugiere emplear las instrucciones para lectura de datos por teclado y salida por pantalla, las veces que sean necesarias; una por cada pregunta. Respecto a la declaración de variables, es necesario manejar al menos dos tipos distintos de datos, es decir, uno para almacenar primero los datos del tipo entero y como esta solución requiere del uso de divisiones, entonces el otro tipo de variable tendría que ser del tipo *float* para que se puedan conservar los decimales.
c) **Comprobación**: Al ejecutar el programa pregunte a un compañero que tenga teléfono celular, la capacidad de la memoria y suponga que el tamaño por video sea de 10 MB. Sorpréndalo informándole cuántos videos puede guardar en su teléfono, aproximadamente. También puede usar este programa si se quiere comprar un teléfono celular y que, probando con diferentes capacidades, vea cual le conviene más.

Ejercicio 3.9.1: Modificar el programa del ejercicio 3.7.2 para que presente un mensaje cuando una persona tenga más de 30 años trabajando, donde informe que "ya se puede jubilar".

a) **Análisis:** Al ser este un ejercicio ya resuelto, se simplifica la adecuación que se pide, porque ya tenemos el resultado, ahora lo que se solicita es incluir un mensaje donde se le avise a la persona que puede estar en condiciones de jubilarse.
b) **Codificación:** Respecto al código, la variante a incluir es el uso de la estructura de control **if()**, la cual permite comparar la antigüedad de la persona con los 30 años de antigüedad requeridos, como mínimo, para la jubilación .
c) **Comprobación**: Al ejecutar el programa pruebe poniendo que un trabajador comenzó en 1980 y verifique el resultado. Posteriormente, ejecute el programa con un trabajador que comenzó en 1990 y revise el resultado, ¿por qué en el primer caso sí aparece el mensaje y

en el otro no? ¿qué pasa si pone el dato para alguien que tenga exactamente 30 años, ¿aparecerá el mensaje o no? ¿por qué?

Ejercicio 3.9.2: Elaborar un programa en lenguaje C que presente un mensaje avisando si hace calor, cuando la temperatura sea mayor a 30 °C. Para ello, debe pedir la temperatura en grados Fahrenheit y hacer la conversión correspondiente, de acuerdo a la siguiente fórmula: °C = [Grados Fahrenheit] - 32 x (5 / 9).

a) **Análisis:** Adicionalmente al resultado de una conversión de grados Fahrenheit a Centígrados, es necesario informar si esa temperatura es cálida o no. En otras palabras, ya se da la operación que permite la conversión, únicamente es necesario comparar ese dato con una temperatura base, en este caso 30 °C y, si es superior, avisar que es un día cálido.
b) **Codificación:** Respecto al código, después de preguntar la temperatura en grados Fahrenheit y hacer el cálculo para efectuar la conversión, será necesario incluir el uso de la estructura de control **if()**, la cual permite comparar la temperatura obtenida contra los 30 °C y avisar si es mayor.
c) **Comprobación**: Probar el programa escribiendo 120 grados Fahrenheit y verificar el resultado, después ejecute el programa con 80 grados Fahrenheit y revise el resultado. ¿Por qué en el primer caso sí aparece el mensaje y en el otro no? ¿en qué casos aparecerá el mensaje y en cuáles no? ¿por qué?

Ejercicio 3.9.3: Calcular qué tiempo le lleva a una impresora en minutos (o en horas si es mayor a 60 minutos) el imprimir un libro con formato PDF, suponiendo que imprime a 10 ppm. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** En este ejercicio hay que solicitar el número de páginas que tiene el libro con formato PDF y hacer la operación para saber el tiempo que le lleva imprimir esas páginas. En este caso ya sabemos la velocidad de la impresora, por lo que no es necesario pedir ese dato; lo que sí se requiere es convertir a horas cuando la duración sea mayor a 60 minutos. Como la impresora puede tardar en imprimir el documento varios minutos o varias horas, al mostrar el resultado será necesario expresarlo en las unidades correspondientes. También, se observa que aquí ya no se pide imprimir mensaje alguno, sino más bien, realizar un cálculo adicional que depende del resultado del primero.
b) **Codificación:** Respecto al código, después de preguntar el tamaño del libro y hacer el cálculo para conocer el tiempo que le lleva imprimir esas páginas, será necesario nuevamente incluir el uso de la estructura de control **if()**, la cual permite comparar si el tiempo es mayor a 60 y, si es mayor, hacer la conversión a horas. Posteriormente, se asignará, en una variable del tipo carácter, la letra 'h' para representar las horas.

c) **Comprobación**: Probar el programa pensando que el libro es de 500 páginas y observar el resultado. Ejecute nuevamente el programa y dé como dato 1,000 páginas; a fin de revisar los resultados y compare ¿qué diferencias nota o notó?

Ejercicio 3.9.4: Calcular el tiempo en minutos u horas que le lleva a un auto ir de Morelia a Pátzcuaro, suponiendo que viaja a la velocidad máxima permitida y sabiendo que velocidad = distancia / tiempo. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** Para este ejercicio, hay que solicitar la velocidad permitida y hacer la operación para conocer el tiempo. En este caso ya sabemos que la velocidad es igual a la distancia entre el tiempo, pero como ya se conoce la distancia (suponga que son 60 Km) y la velocidad, será necesaria una tarea adicional, esto es, alterar la fórmula para que nos permita conocer el tiempo. Como el tiempo de viaje puede consistir en varios minutos o varias horas, al mostrar el resultado será necesario expresarlo en las unidades correspondientes. También, se observa que aquí ya no se pide imprimir mensaje alguno, sino más bien realizar un cálculo adicional, que depende del resultado del primero.
b) **Codificación:** Respecto al código, después de preguntar la velocidad del vehículo y hacer el cálculo para conocer el tiempo que le lleva ir de Morelia a Pátzcuaro, será necesario nuevamente incluir el uso de la estructura de control **if()**, la cual permite comparar si el tiempo es menor a 1 hora. Si es menor, hacer la conversión a minutos y asignar, en una variable del tipo carácter, la letra 'm' para representar los minutos.
c) **Comprobación**: Probar el programa escribiendo como dato que la velocidad es de 60 Km/h y observar el resultado. Ahora ejecutar nuevamente el programa y dar como dato 120 Km/h. Revise los resultados y compare ¿qué diferencias nota o notó?

Ejercicio 3.9.5: Calcular el rendimiento de gasolina de un auto, cuya capacidad del tanque es de 45 litros de gasolina. Avisar si tiene problemas mecánicos cuando su rendimiento sea menor al rendimiento ideal. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** En ejercicios previos se comparaba el valor de una variable con un dato constante o fijo y aquí hay una variante, se debe averiguar, porque se necesita preguntar varias cosas tales como el kilometraje inicial (antes de llenar el tanque) y el kilometraje final (kilometraje recorrido una vez terminado el tanque). También es importante preguntar cuál es el rendimiento ideal, porque varía de un auto a otro cuando son de diferente compañía. Es aquí donde aparece la variante, ya que necesitamos comparar el rendimiento ideal, proporcionado por el usuario, y el otro obtenido al dividir los kilómetros recorridos entre la capacidad del tanque (45 litros). Con base en el resultado de la comparación, avisar al conductor de las condiciones de su auto.

b) **Codificación:** Después de solicitar los datos necesarios y hacer los cálculos respectivos, se requerirá el uso de la estructura de control **if – else,** la cual permite comparar ambos rendimientos y, dependiendo del resultado de la comparación, mostrar los mensajes correspondientes.
c) **Comprobación**: Al ejecutar el programa probar con dos situaciones, es decir, cuando el rendimiento real sea menor al rendimiento ideal y, luego, otro caso donde el rendimiento ideal sea igual al real. Compare ¿qué diferencias nota en ambos casos?

Ejercicio 3.9.7: Una escuela calcula la calificación final de un alumno a partir de 3 calificaciones parciales y lo hace promediando los parciales. Calcular el promedio final e informar al alumno si tiene derecho a BECA, si su promedio es mayor a 9 o, si está reprobado, cuando su promedio sea menor a 7. Desarrolle un programa en lenguaje C que resuelva el planteamiento anterior.

a) **Análisis:** En este ejercicio es necesario solicitar 3 calificaciones y calcular el promedio. Ya con ese dato listo, se puede comparar para saber la situación del estudiante, es decir, si tiene derecho a beca o repite curso.
b) **Codificación:** Después de solicitar los datos necesarios y hacer los cálculos respectivos, se requerirá usar dos estructuras **if(),** lo cual permitirá comparar ambos casos y, dependiendo del resultado de la comparación, mostrar los mensajes correspondientes.
c) **Comprobación**: Al ejecutar el programa probar con las dos situaciones, es decir, cuando el promedio sea mayor a 9 y, luego, otro caso donde el promedio sea menor a 7. Compare ¿qué diferencias nota en ambos casos?

Ejercicio 3.9.8: Calcular el total a pagar en copias fotostáticas dependiendo del número y tamaño de copias. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** Para este ejercicio, después de preguntar la cantidad de copias, es necesario definir el tamaño, porque de eso depende el precio de la copia. En este caso particular se puede suponer de antemano un monto predefinido para los precios, es decir, que la copia tamaño carta valga 30 centavos y la oficio 50. Solicite el tamaño de las copias y, ya con esos datos, haga la operación correspondiente.
b) **Codificación:** En este caso en especial, se puede decir que la variante es que antes de hacer el cálculo respectivo es necesario primero conocer de qué tamaño sacó las copias el usuario y, ya con ese dato, usar la estructura de control simple **if-else**. Como se va preguntar el tamaño, el usuario puede proporcionar una letra en minúscula o mayúscula, así que aquí es adecuado usar los operadores de relación para obtener el total correcto.
c) **Comprobación**: Para notar cómo varía el resultado, pida sacar 100 copias y primero elija 'C', para tamaño carta y luego oficio. Revise qué diferencia nota en el resultado final.

Ejercicio 3.9.9: Un sitio en Internet permite descargar videos y películas. Dependiendo del tipo de conexión es la velocidad de descarga que maneja, es decir, que si la conexión es telefónica la velocidad es de 128 Kbps y, si es de banda ancha, es de 128 Mbps. ¿Cuántos días u horas le llevaría descargar una película, si elige un tipo de conexión? Desarrolle un programa que pida el tipo de conexión, el tamaño de la película y que resuelva la pregunta planteada.

a) **Análisis:** En síntesis, lo que se pide en este ejercicio es solicitar el tamaño de la película a descargar y el tipo de conexión. Con esos datos se puede calcular el tiempo que llevaría la descarga. Aquí también la idea es que si la descarga lleva más de 24 horas, el resultado se pase a días. Para ello será necesario dividir el total de horas entre 24, con la finalidad de averiguar cuántos días llevará la descarga.
b) **Codificación:** Es claro observar que se usará la estructura de control **if-else**, ya sea en escalera o anidada, ya que al menos se requiere de un par de comparaciones; primero el tipo de conexión y, luego, saber si lo que va a durar la descarga rebasa las 24 horas, para realizar la conversión a días.
c) **Comprobación**: Este programa tiene 3 variantes, primero es correr el programa con los dos tipos de conexión y un mismo tamaño para la película; luego, dar un tamaño enorme para la película y la conexión más lenta, para ver si funciona la conversión a días solicitada.

Ejercicio 3.9.10: Elaborar un programa en lenguaje C que solicite el peso y estatura de una persona e informe cuál es su índice de masa corporal (*IMC*). Además, debe informar con mensajes apropiados la situación de la persona, de acuerdo con el cuadro siguiente:

IMC	***Mensaje***
Menos de 0.20	Bajo peso
Entre 0.20 y 0.25	Peso normal
Entre 0.25 y 0.30	Peso moderado
Más de 0.30	Sobrepeso

a) **Análisis:** Para resolver el planteamiento anterior es necesario investigar la fórmula para calcular el IMC, porque podemos pedir la estatura y el peso como se solicita, sin embargo, de manera inicial o intuitiva se desconoce la proporción correcta entre peso-estatura, a fin de averiguar si la persona anda en su peso correcto o no. Ya obtenido el resultado, se puede dar la interpretación correcta de acuerdo al cuadro anterior.

b) **Codificación:** La solución se centra en la estructura de control **if-else**, que se sugiere emplear en cascada, contemplando los rangos marcados en el recuadro que aparece arriba y, de esa manera, únicamente imprimir en pantalla el mensaje apropiado.

c) **Comprobación**: Si queremos probar este programa será necesario combinar los datos de las personas, de manera que como resultado nos den un IMC inferior a 0.20, entre 0.20 y

0.25, de 0.25 a 0.30 y superior a 0.30; para que se pueda verificar la impresión del mensaje apropiado, además, de que solo aparezca uno de ellos.

Ejercicio 3.9.11: Elaborar un programa en lenguaje C que simule una caja registradora. Esto es, que a partir del total a pagar informe al cajero el mensaje apropiado, dependiendo de la cantidad entregada por el cliente, de acuerdo al cuadro siguiente:

Cantidad entregada por el cliente	Mensaje
Es más que el total a pagar	"Su cambio es de: " e indicar cuánto dinero se devuelve.
Es menos que el total a pagar	"Pago insuficiente, faltan: " e indicar cuánto dinero hace falta.
Es igual al total a pagar	"Gracias por su compra, vuelva pronto."

a) **Análisis:** Con un grado de dificultad mayor al ejercicio anterior, el resto de la solución es muy similar. Se solicitan un par de datos, el cobro y el pago que hace el cliente; de aquí se saca la diferencia que existe entre ambos y, ya obtenido el resultado, se imprime en pantalla el mensaje apropiado.
b) **Codificación:** Respecto a la solución con estructuras de control, nuevamente puede ser resuelto con una cascada de **if-else**, contemplando los rangos marcados en el recuadro que aparece arriba y, de esa manera, únicamente imprimir en pantalla el mensaje correcto.
c) **Comprobación**: Si queremos probar este programa será necesario combinar los datos de tal manera que el pago sea mayor al cobro, sean iguales y sea menor el pago al cobro; para que podamos verificar que el programa solo envía un mensaje en cada caso.

Ejercicio 3.9.12: Elaborar un programa en lenguaje C que simule el cajero de un banco despachador de dinero. Esto es, que a partir del saldo informe al cliente el mensaje apropiado, dependiendo de la cantidad solicitada por el cliente, de acuerdo al cuadro siguiente:

Condiciones posibles	Acción
Menú con cantidades en billetes circulantes: 20, 50, 100, 200, 500 y 1000.	"Retiro autorizador por $ ", señalar la cantidad y avisar cuál es su saldo nuevo.
Cantidad tecleada por el cliente en múltiplos de 100	"Retiro autorizador por $ ", señalar la cantidad y avisar cuál es su saldo nuevo.
Fondos insuficientes	"Lo sentimos, no cuenta con fondos suficientes para hacer ese retiro."

⇨ **En todos los casos deberá recalcular el saldo final e informar si el cliente cuenta con fondos suficientes.**

a) **Análisis:** Aquí la solución parece simple, ya que únicamente se debería solicitar la cantidad a retirar y el saldo disponible para avisarle al usuario el saldo final, o bien, si tiene los fondos suficientes para hacer el retiro. Donde viene la complicación es en todas las combinaciones posibles de retiro, por lo que aquí se hace necesario aplicar una estructura de control diferente.
b) **Codificación:** Como la clave está en las estructuras de control a implementar, esta es indiscutiblemente la estructura **switch()**, la cual permite construir menús para que la persona elija una de las alternativas presentadas y, ya sabiendo la elección, se puede determinar la operación correcta que permita hacer el respectivo descuento.
c) **Comprobación**: Será necesario la ejecución del programa varias veces para elegir varias de las opciones y verificar los diferentes resultados, para de esa forma comprobar el funcionamiento óptimo del programa. No olvide verificar el caso donde se elija la opción de introducir una cantidad y se ponga una cantidad mayor al saldo inicial, verificando el tipo de mensaje que aparece.

Ejercicio 3.9.13: Elaborar un programa en lenguaje C que simule el servicio que hace una compañía de teléfonos celulares. Esto es, que a partir del saldo disponible en un teléfono celular, informe al cliente su saldo nuevo o saldo final, después de que el cliente utilice algunos de los servicios disponibles por la compañía, de acuerdo a los servicios siguientes:

Condiciones posibles	Acción
Menú de servicios: MMS de texto, MMS multimedia, MMS para rifa, descargar tono, pasatiempo y llamada.	En caso de que sea cualquier tipo de mensaje, preguntar la tarifa y verificar que haya fondos suficientes, de lo contrario activar una bandera para enviar, al final, un aviso de que el mensaje no pudo ser enviado.
En el caso del tono, preguntar la tarifa.	Aparte de preguntar la tarifa impuesta por la compañía de la cual se descarga el tono, verificar que haya saldo suficiente, de lo contrario activar una bandera para enviar, al final, un aviso de que el mensaje no pudo ser enviado.
Para la llamada, deberá preguntar adicionalmente si, es llamada a teléfono local, teléfono celular de la misma compañía, teléfono celular de otra compañía, si es número frecuente o si está dado de alta como número gratuito	Recordemos que son diferentes tarifas, dependiendo si el teléfono es móvil o fijo y si es local o larga distancia, pero todo se simplifica si se solicita la tarifa de la llamada, lo importante es preguntar dos cosas, la duración de la llamada y si se origina dentro del área de cobertura o fuera de ella, porque al ser fuera, se tiene que hacer un cargo adicional de llamada *roaming*, que es otra cantidad fija que se cobra por minuto.

⇨ **En todos los casos deberá recalcular el saldo final e informar si el cliente cuenta con fondos suficientes para realizar la acción que desea.**

a) **Análisis:** En esencia, la solución a este ejercicio es muy parecida al ejercicio previo, pero tal vez aquí se requiera el manejo de un submenú, dadas las diferentes tarifas que

manejan las compañías y todas las combinaciones posibles que existen para descontar el saldo dependiendo de si la llamada es local, larga distancia, a teléfono fijo o móvil y si la persona que realiza la llamada está dentro o fuera de su área. En este caso, para simplificar, se puede solicitar la tarifa aplicada en la llamada; esto reduce la complejidad en la implementación de la solución. En conclusión, será necesario pedir qué servicio usó la persona, a través de un menú y, luego, la tarifa respectiva. Solo en el caso de la llamada se deberán pedir los minutos, con esa información hacer el cálculo correspondiente y aplicar el descuento respectivo.

b) **Codificación:** Aquí la clave nuevamente se encuentra en la estructura de control a implementar y es la estructura **switch()**, la cual permite construir el menú para que la persona elija una de las alternativas presentadas. Ya sabiendo la elección, se puede determinar la operación correcta que permita hacer el descuento respectivo.

c) **Comprobación**: Será necesaria la ejecución del programa varias veces, para elegir varias de las opciones y verificar los diferentes resultados para, de esa forma, comprobar el funcionamiento óptimo del programa.

Ejercicio 3.9.14. Elaborar un programa en lenguaje C que realice la multiplicación de dos números, sin utilizar la multiplicación directamente.

a) **Análisis:** Hay mucho que analizar para este ejercicio, porque muchas aplicaciones y lenguajes de programación traen el operador matemático para la multiplicación, pero si la restricción es evitar su uso, cabe recordar que la multiplicación no es más que el resultado de sumas sucesivas, por ejemplo: 2 x 3 es igual a decir 2 + 2 + 2. Esto es, si pedimos dos números, el primero es la base de la suma y, el segundo, es el número de veces que se va a realizar la suma. Con esto se obtiene un resultado equivalente a multiplicar.

b) **Codificación:** En este caso y para resolver el planteamiento, se sugiere utilizar la estructura de control repetitiva o cíclica llamada **for()**, ya que esta permite ejecutar una o varias instrucciones un cierto número de veces conocido de antemano, es decir, de 1 a N veces.

c) **Comprobación**: Este programa se puede comprobar multiplicando un numero por sí mismo y verificar si da el resultado correcto, por ejemplo: 5 x 5 y el resultado deberá ser 25.

Ejercicio 3.9.15. Elaborar un programa en lenguaje C que imprima una frase de castigo y a continuación la presente numerada, repitiéndola el número de veces indicado, a fin de que el castigado la pueda imprimir y, de esta forma, pueda cumplir con su castigo.

Ejemplo: “Debo ser puntual” 5 veces.

1. Debo ser puntual.
2. Debo ser puntual.

3. Debo ser puntual.
4. Debo ser puntual.
5. Debo ser puntual.

a) **Análisis:** Aunque es muy descriptivo el ejercicio anterior, es bueno observar que si la frase va a aparecer N veces, podemos darnos cuenta que se trata de un ciclo que permita imprimir varias veces una frase. En cuanto a la solicitud de los datos, se pide específicamente el número de veces que la frase se repetirá, pero no dice que necesariamente la frase se tenga que solicitar. Como aún no se ve el tema para guardar cadenas de caracteres, la solución se simplifica usando la instrucción **printf()** que contenga directamente la frase de ejemplo u otra que el programador decida.
b) **Codificación:** Para este programa, se sugiere utilizar la estructura de control repetitiva o cíclica llamada **for()**, porque es evidente la necesidad de imprimir en pantalla una frase tantas veces como lo solicite el usuario. Este número de veces se puede saber de antemano, por lo que esa estructura de control es la idónea.
c) **Comprobación**: Antes de ejecutar este programa y como parte de la instrucción **printf()**, es conveniente imprimir la variable tipo contador del ciclo, así si la persona pide 20 veces la frase, podrá comprobar con ese número que efectivamente se imprimió 20 veces.

Ejercicio 3.9.16. Modificar el programa del ejercicio 3.9.14 para que pregunte si se desea realizar otra multiplicación y, de ser afirmativa la respuesta, volver a preguntar los números a multiplicar y así sucesivamente, hasta que la persona diga lo contrario.

a) **Análisis:** Si bien es cierto ya se hizo un análisis previo del ejercicio 3.9.14, para la modificación que se pide es importante examinar que se trata de un ciclo, ya que se pide que se vuelva a ejecutar el programa, pero podemos apreciar que esa ejecución está condicionada a la respuesta que el usuario nos dé.
b) **Codificación:** Para la modificación a incorporar, se sugiere utilizar la estructura de control repetitiva o cíclica llamada **do-while()**, porque esta estructura ejecuta varias veces una o más instrucciones, dependiendo del resultado de una condición; en otras palabras, siempre y cuando el resultado de la comparación sea verdadero.
c) **Comprobación**: La manera de probar este programa es dar respuesta a la última pregunta con las letras ‘S’, ‘s’, ‘N’ y ‘n’; a fin de determinar en qué casos continúa la ejecución del programa y en cuáles no.

Ejercicio 3.9.17. Elaborar un programa en lenguaje C que muestre la tabla de multiplicar indicada por el usuario y, al final, pregunte si se desea mostrar una tabla diferente o si se desea salir del programa. Ejemplo: tabla del 3, 3x1=3, 3x2=6, 3x3=9,…, 3x10=30

a) **Análisis:** Dado el planteamiento anterior es notorio observar que entre líneas se habla de dos ciclos, uno que realice la multiplicación de 1 hasta 10 y, otro ciclo, cuya ejecución está condicionada a la respuesta que el usuario nos dé.
b) **Codificación:** La estructura de control repetitiva o cíclica llamada **do-while()** resuelve la parte de ejecutar todo el programa nuevamente, tantas veces como el usuario quiera y, con el ciclo **for(),** se resuelve la parte de presentar la tabla de multiplicar del número solicitado.
c) **Comprobación**: La manera de probar este programa es dar respuesta a la última pregunta con las letras 'S', 's', 'N' y 'n'. ¿Qué ocurre con el programa si se contesta con un carácter diferente a los anteriores? ¿se efectúa nuevamente la ejecución? ¿sí o no y por qué?

Ejercicio 3.9.18. Simular una bomba despachadora de gasolina, de tal manera que calcule el importe total, conociendo el tipo de combustible y la cantidad de litros que desea el automovilista. El proceso se debe repetir hasta que ya no haya más autos. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** En este planteamiento no es evidente el uso de ciclos, pero si la operación de cobrar el combustible se va a realizar para cada auto que cargue gasolina, con eso descubrimos la presencia de un ciclo. Además, como está condicionado el cálculo a si existen más autos por despachar, podemos corroborar que el ciclo es condicionado. En cuanto al cálculo, podemos observar que se trata de pedir el tipo de combustible y la cantidad de litros, con lo que una simple multiplicación dará el resultado, pero no se pierda de vista que existen al menos dos tipos de combustible. Como no se especifica que se pida el precio del combustible, este puede definirse como una constante en el programa, o como variables inicializadas con las tarifas vigentes.
b) **Codificación:** La estructura de control repetitiva o cíclica llamada **do-while()** resuelve la parte de ejecutar todo el programa nuevamente tantas veces como el usuario quiera, pero para hacer el cobro respectivo se tienen que combinar ciclos con estructuras de control condicional, en este caso **if()**.
c) **Comprobación**: Al momento de proporcionar los datos, en una primera ejecución, escoger un tipo de combustible y, en la siguiente, un tipo diferente. Observe la diferencia en los resultados.

Ejercicio 3.9.19. Modificar el programa anterior para que informe al automovilista cuántos litros le serán surtidos, para aquellos clientes que acostumbran pedir con base en una cantidad de dinero. El proceso se debe repetir hasta que ya no haya más autos que atender. Desarrolle un programa que resuelva este planteamiento.

a) **Análisis:** Ahora en este replanteamiento existe una variante y es que, en México, es muy común que la persona pida por cantidad de dinero y no por litros, es decir, que se puede solicitar que se surtan $ 200 pesos. Lo que se pide es que el programa informe cuánto es

esa cantidad en litros. El resto de la solución es idéntica, ya que también es necesario conocer qué tipo de combustible eligió el cliente.

b) **Codificación:** La estructura de control repetitiva o cíclica llamada **do-while()** resuelve la parte de ejecutar todo el programa nuevamente tantas veces como el usuario quiera, pero para hacer el cobro respectivo se tienen que combinar ciclos con estructuras de control condicional, en este caso **if()**, de tal manera que se permita informar la cantidad exacta en litros, una vez que se sabe el tipo de combustible a despachar.

c) **Comprobación**: Al momento de proporcionar los datos, en una primera ejecución, escoger un tipo de combustible y, en la siguiente, un tipo diferente. Observe la diferencia en los resultados.

Ejercicio 3.9.20. Elabore un programa que permita hacer divisiones hasta que el usuario decida terminar los cálculos. Además, el programa debe permitir que el usuario intente adivinar el resultado y lo felicite cuando acierte, o lo corrija cuando falle. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** Aunque se pide una operación muy básica, aquí lo interesante es retar al usuario preguntando el posible resultado y felicitarlo en caso de acertar, o animarlo a seguir intentándolo.

b) **Codificación:** Con la estructura de control **if()** se resuelve la parte de comparar la respuesta del usuario con el resultado de la operación y, si se desea que el programa se vuelva a ejecutar nuevamente a voluntad del usuario, la estructura de control repetitiva o cíclica llamada **do-while()** resuelve esta otra parte.

c) **Comprobación**: Para probar este programa realice una división entre un mismo número, como por ejemplo 10 entre 10, ya que sabemos que cualquier numero dividido entre sí mismo es uno, y así será fácil adivinar el resultado; además de comprobar que sea, a su vez, el resultado correcto. En ambos casos, cuando se atine o se falle al adivinar el resultado, note qué mensaje aparece en cada caso.

Ejercicio 3.9.21. Elabore un programa que calcule el total a pagar en una taquería donde se puede pedir la cuenta, ya sea individualmente o en grupos variables. Los datos a considerar son el número de tacos consumidos y la cantidad de bebidas ingeridas por cada persona, asimismo suponer que los tacos y las bebidas tienen un precio fijo. Adicionalmente, calcular la propina sugerida a la razón de un 15% sobre el total de la cuenta y preguntar al cliente si está de acuerdo en que se sume al total a pagar. De ser afirmativa la respuesta, entonces, sumar ambos cargos a la cuenta e informar del monto final a pagar. Desarrolle un programa que resuelva el planteamiento anterior.

a) **Análisis:** Para concluir esta unidad, se presenta un ejercicio más elaborado, pero que no se aparta en la simpleza de las operaciones, ya que puede ser resuelto con sumas y

multiplicaciones. Con el propósito de aplicar las estructuras repetitivas, se pide que se suponga que fue un grupo de personas a la taquería y que piden una sola cuenta.

b) **Codificación:** Como se pide calcular con base en un porcentaje la propina y se da la opción al cliente de incluirla en el total, esa aprobación se sugiere se trabaje con la estructura de control **if(),** ahora que si se desea que el programa se vuelva a ejecutar nuevamente por cada cliente que es parte de ese grupo, en caso de que quieran cuentas individuales, la estructura de control repetitiva o cíclica llamada **do-while()** resuelve esa otra parte del problema.

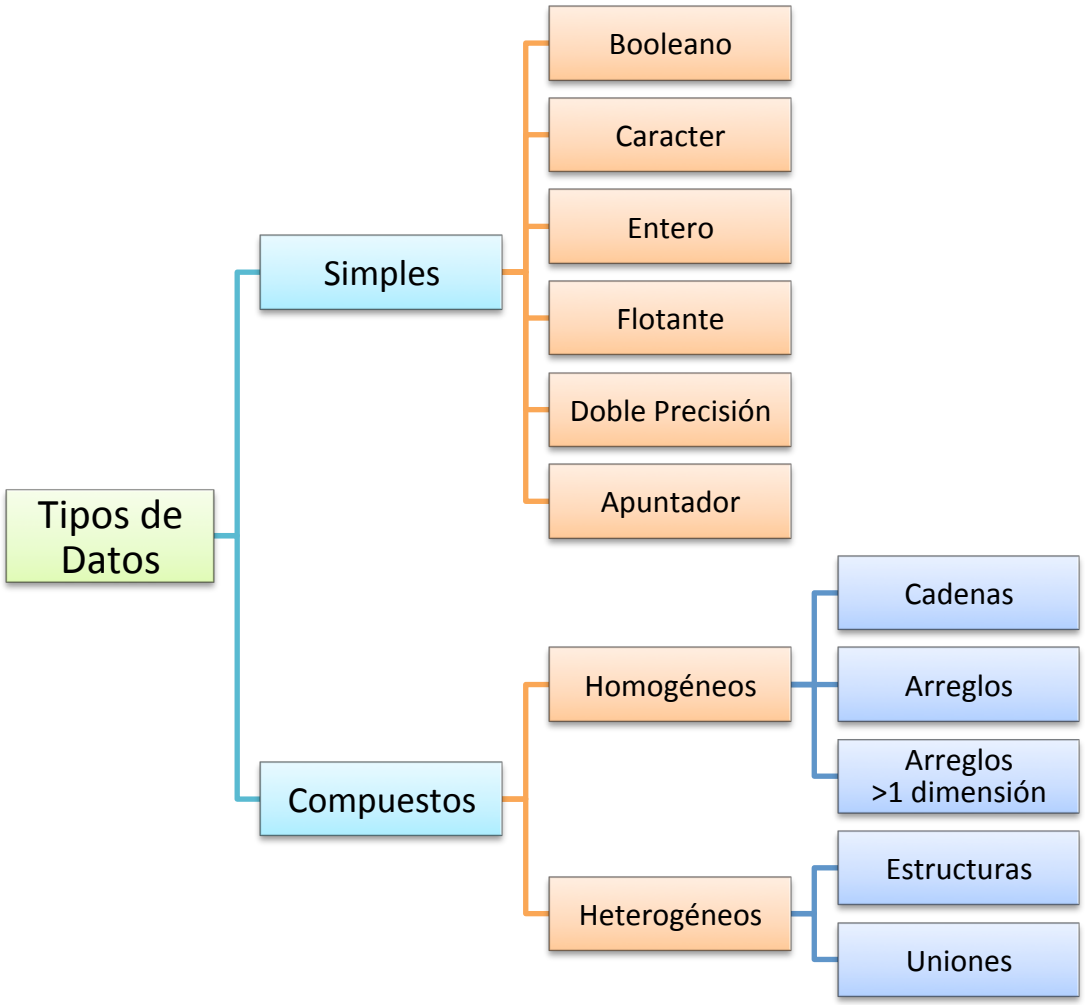
c) **Comprobación**: Si se quiere comprobar este programa hágalo con dos clientes, que consumieron la misma cantidad de tacos y una sola bebida del mismo precio, ambos. Además, aceptan que se incluya la propina sugerida. Tome nota del total a pagar. Posteriormente, ejecute el programa con los mismos datos, pero en esta oportunidad no aceptan dejar propina. ¿Qué diferencia nota en el resultado? Si los resultados son correctos, utilice el programa como simulador para cuando piense ir con amigos a un lugar de este tipo, calculando cuánto dinero necesitan en total, si cada uno de antemano sabe lo que va a comer y lo que va a tomar. Así tendrán su presupuesto y se protegerán de cobros indebidos.

APÉNDICE

Ejercicio 1

1.- Constantes; 2.- Short; 3.-Tipo de dato / Nombre; 4.- Apuntador; 5.- Variables;
6.- Char; 7.- Reservadas; 8.- Float; 9.- Declaración; 10.- Locales

Ejercicio 2



3.2. Identificadores y palabras reservadas

Ejercicio 3

1.- (j); 2.- (c); 3.- (f); 4.- (g); 5.- (h); 6.- (b); 7.- (i); 8.- (a); 9.- (d); 10.- (e)

3.5. Operadores, precedencias y evaluaciones en cortocircuito

Ejercicio 4

1.- Precedencia; 2.- Conversiones; 3.- Asignación; 4.- Casting; 5.- Variables estáticas
6.- De incremento; 7.- Lógicos; 8.- Bits; 9.- Bloque de instrucciones; 10.- Relacionales

3.6. Conversiones

Ejercicio 5

1.- (e); 2.- (j); 3.- (g); 4.- (h); 5.- (b); 6.- (d); 7.- (a); 8.- (i); 9.-(f); 10.-(c)

3.7. Interacción con el usuario

Ejercicio 6

1.- stderr; 2.- stdout; 3.- stdin; 4.- getch(); 5.- printf();
6.- “%c”; 7.- “%d”; 8.- “%s”; 9.- “%f”; 10.- putc()

3.8. Estructuras de control

Ejercicio 7

1.- while; 2.- for; 3.- do – while; 4.- Ciclos anidados; 5.- if
6.- if – else; 7.- switch; 8.- break; 9.- continue; 10.- goto; 11.- return

Ejercicio 8

```
while (expresión)
{
   Instrucciones o sentencias;

}
```

```
do
{
   Instrucciones o sentencias;

}while (expresión)
```

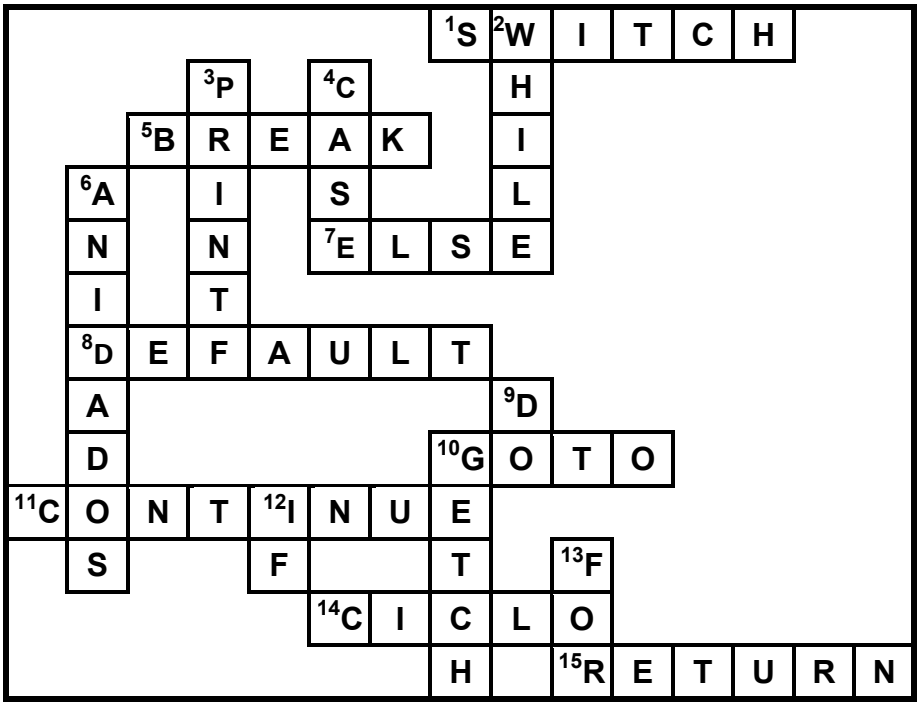
```
for (inicialización; condición; actualización)
{
   Instrucciones o sentencias;
}
```

```
if (expresión)
{
   Instrucciones o sentencias;

}
```

```
if  (expresión)
{
   Instrucciones o sentencias (condición verdadera);
}
else
{
   Instrucciones o sentencias (condición falsa);
}
```

```
switch (expresión)
{
   case 1:
      Instrucciones o sentencias 1;
      break;
   case 2:
      Instrucciones o sentencias 2;
      break;
   default:
      Instrucciones o sentencias 3;
      break;
}
```

CRUCIGRAMA



Lista de ejercicios prácticos RESUELTOS

Nota: Las soluciones a los ejercicios fueron probadas en las versiones para Windows de los compiladores Turbo C++ versión 3.0 y Dev-C++ versión 4.9.9.2.

Listado 3.7.1

```
/* ----------------------------------------------------------------
Programa: Ejercicio3-7-1.c
Objetivo: Aplicación de operaciones aritméticas y operaciones de entrada
y salida de datos. Calculando, además, la edad para una persona.
----------------------------------------------------------------*/
#include <stdio.h>

int main()
{
      int aho_actual = 2014;
      int  aho, edad;


      /* Lectura de datos */
      printf("\n Año en que naciste: ");
      scanf("%d", &aho);


      /* Cálculo de la edad */
      edad = aho_actual - aho;


      /* Impresión de resultado */
      printf("\n Si tu naciste en %d, tienes %d años.", aho, edad);

 }
```

Listado 3.7.3

```
/* ----------------------------------------------------------------------
Programa: Ejercicio3-7-3.c
Objetivo: Aplicación de operaciones aritméticas y operaciones de entrada
y salida de datos. Declaración de varios tipos de variables y cómo
solicitarlas apropiadamente.
----------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
      int frente, fondo, terreno;
      float preciom2, valorT;


      /* Lectura de datos */

      printf("\n Proporcione las medidas del terreno…\n");

      printf("\n Cuánto mide de frente (metros): ");
      scanf("%d", &frente);

      printf("\n Cuánto mide de fondo (metros): ");
      scanf("%d", &fondo);

      printf("\n Precio del metro cuadrado $ ");
      scanf("%f", &preciom2);


      /* Realización de cálculos */

      terreno = frente * fondo;

      valorT = terreno * preciom2;



      /* Impresión de resultado */

      printf("\n El precio total de un terreno de %d x %d metros, a $ %f
      pesos el metro cuadrado, es de: $ %f pesos", frente, fondo,
      preciom2, valorT);

 }
```

Listado 3.7.5

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-7-5.c
Objetivo: Aplicación de operaciones aritméticas y operaciones de entrada
y salida de datos, para convertir dólares en pesos.
------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
      float dolares, cotiza, pesos;


      /* Lectura de datos */

      printf("\n Proporcione la cantidad de dinero (en dólares) $ ");
      scanf("%f", &dolares);

      printf("\n Cotización del dólar $ ");
      scanf("%f", &cotiza);


      /* Cálculo de la conversión */

      pesos = dolares * cotiza;


      /* Impresión de resultados */

      printf("\n %f dólares en pesos son $ %f pesos", dólares, pesos);

 }
```

Listado 3.8.1

```
/* ----------------------------------------------------------------
Programa: Ejercicio3-8-1.c
Objetivo: Aplicación de operaciones aritméticas y operaciones de entrada
y salida de datos
----------------------------------------------------------------*/
#include <stdio.h>

int main()
{
      float porcentaje, descuento, precio_ini, precio_fin;

      printf("Precio inicial? $ ");
      scanf("%f", &precio_ini);

      printf("\n Porcentaje de descuento (por ejemplo: 15)? ");
      scanf("%f",&porcentaje);


      /* Cálculo del descuento */
      descuento = precio_ini * porcentaje / 100;

      /* Cálculo del precio final*/
      precio_fin = precio_ini - descuento;


      /* Impresión de resultados */

      printf("\n Un producto que vale: $ %f pesos, con el %f%% de
      descuento…", precio_ini, porcentaje);

      printf("\n … tiene un descuento de $ %f pesos y, por lo tanto,
      tiene un precio final de $ %f pesos", descuento, precio_fin);

}
```

Listado 3.8.3

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-8-3.c (Verificar equivalencia de megapixeles a
megabytes)
Objetivo: Aplicación de operaciones aritméticas y operaciones de entrada
y salida de datos
------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
      int cmemo, maxresolucion;
      float equivaleEnGb, totalF;

      printf("Introduce la máxima resolución permitida por la cámara (en
      megapixeles): ");
      scanf("%d", &maxresolucion);

      printf("\n Introduce la capacidad total de la cámara (Gigabytes):
      ");
      scanf("%d",&cmemo);

      /* Convertir megapixeles a gigabytes, 1024.0 para que la expresión
      se convierta a float */
      equivaleEnGb = maxresolucion / 1024.0;

      totalF = cmemo / equivaleEnGb;

      printf("\n El total de fotos que se pueden guardar en esta cámara
      es de: %f imágenes", totalF);

}
```

Listado 3.8.5

```
/* ----------------------------------------------------------------
Programa: Ejercicio3-8-5.c
Objetivo: Aplicación de operaciones aritméticas y operaciones de entrada
y salida de datos
----------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int ncds, cCD = 700;
     float tdvds, dvd, equivaleGb, cDVD = 4.7;

     printf("Introduce cuántos discos compactos tiene la estación de
     radio: ");
     scanf("%d", &ncds);

     /* Convertir megabytes a gigabytes, 1024.0 para que la expresión se
     convierta a float */
     equivaleGb = cCD / 1024.0;
     dvd = cDVD / equivaleGb;
     tdvds = ncds / dvd;

     printf("\n El total de discos DVD requeridos para hacer el respaldo
     es de: %f DVDs", tdvds);

}
```

Listado 3.9.1

```
/* ----------------------------------------------------------------------
Programa: Ejercicio3-9-1.c
Objetivo: Uso de estructuras condicionales simples.
----------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
      int aho_actual = 2014;
      int  aho_ing, antiguedad;

      printf("\n En qué año se comenzó a laborar: ");
      scanf("%d", &aho_ing);

      antiguedad = aho_actual - aho_ing;

      printf("\n Actualmente tiene una antigüedad de %d años",
      antiguedad);

      if (antiguedad > 30)
            printf("\n Felicidades, ya se puede jubilar.”);

}
```

Listado 3.9.3

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-9-3.c
Objetivo: Uso de estructuras condicionales simples.
------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int ppm = 10, nplibro;
     float tiempo;
     char utiempo;

     printf("Introduce cuántas páginas tiene el libro? ");
     scanf("%d", &nplibro);

     tiempo = nplibro / ppm;
     utiempo = 'm';

     if (tiempo > 60)
     {
         tiempo = tiempo / 60;
         utiempo = 'h';
     }

     printf("\n El tiempo necesario es de: %f %c", tiempo, utiempo);

}
```

Listado 3.9.5

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-9-5.c
Objetivo: Uso de estructuras condicionales simples.
------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int kminicial, kmfinal, kmrecorrido;

     /* 45.0 para que la expresión se convierta a float */
     float tanque = 45.0;
     float rendimientoR, rendimientoI;


     printf("\n Proporcione el kilometraje inicial: ");
     scanf("%d", &kminicial);

     printf("\n Proporcione el kilometraje final: ");
     scanf("%d", &kmfinal);

     printf("\n Proporcione el rendimiento ideal (km/litro): ");
     scanf("%f", &rendimientoI);

     kmrecorrido = kmfinal - kminicial;
     rendimientoR = kmrecorrido / tanque;

     if (rendimientoR < rendimientoI)
     {
        printf("\n El rendimiento real es de: %f kilómetros por litro,
        cuando debería ser de: %f kilómetros por litro", rendimientoR,
        rendimientoI);

        printf("\n Es necesario que lleve su carro al mecánico o a
        mantenimiento");
     }
     else
        printf("\n El rendimiento real es de: %f kilómetros por litro y,
        al compararlo con el ideal, mecánicamente tu carro anda bien",
        rendimientoR);

}
```

Listado 3.9.8

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-9-8.c
Objetivo: Uso de estructuras condicionales simples.
------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int ncopias;
     char tcopia;
     float carta = 0.3, oficio = 0.5, tpagar;


     printf("\n Cuantas copias fueron? ");
     scanf("%d", &ncopias);

     printf("\n Indique el tamaño de la copia (C=Carta y O=Oficio): ");

     /* Se requiere llamar dos veces a scanf() para quitar, de la entrada
     estándar, la tecla ENTER que se presiona para introducir el número
     de copias solicitado en la pregunta anterior. Si no se quita, la
     respuesta al tamaño de la copia será SIEMPRE dicha tecla ENTER, es
     decir, para este programa se elegiría SIEMPRE el tamaño oficio.

     PRUÉBELO quitando una de las líneas con la llamada a scanf() */

     scanf("%c", &tcopia);
     scanf("%c", &tcopia);

     if (tcopia == 'c' || tcopia == 'C')
          tpagar = ncopias * carta;
     else
          tpagar = ncopias * oficio;

      printf("\n El total a pagar es de: $ %f pesos", tpagar);
}
```

Listado 3.9.10

```
/* ----------------------------------------------------------------
Programa: Ejercicio3-9-10.c
Objetivo: Uso de estructuras condicionales compuestas, calculando el
índice de masa corporal para una persona
----------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int peso, estatura;
     float imc, doble_esta;

     printf("\n Déme su peso (en kilos y sin decimales): ");
     scanf("%d", &peso);

     printf("\n Déme su estatura (en centímetros): ");
     scanf("%d", &estatura);

     /* Operaciones para calcular el índice de masa corporal (IMC) */
     doble_esta = estatura + estatura;
     imc = peso / doble_esta;

     printf("\n Su índice de masa corporal es de: %f", imc);

     if (imc < 0.20)
          printf("\n ¡¡¡ ALERTA !!!, BAJO PESO”);
     else if ((imc >= 0.20) && (imc < 0.25))
          printf("\n ESTÁ EN EL PESO NORMAL”);
     else if ((imc >= 0.25) && (imc < 0.30))
          printf("\n ESTÁ EN UN PESO MODERADO”);
     else
          printf("\n ¡¡¡ CUIDADO !!!, TIENE SOBREPESO”);

}
```

Listado 3.9.12

```
/* ----------------------------------------------------------------
Programa: Ejercicio3-9-12.c
Objetivo: Uso de estructuras condicionales de selección.
----------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int saldo_ini = 30000;
     int saldo_ins, saldo_fin, retiro;
     char opcion;

     saldo_ins = 0;

     printf("\n BIENVENIDO A BANCONALEP \n");

     printf("(1) 20\t\t\t(4) 200 \n");
     printf("(2) 50\t\t\t (5) 500 \n");
     printf("(3) 100\t\t\t (6) 1000 \n");
     printf("(*) Una cantidad distinta a las anteriores \n");
     printf("Digite su selección: ");

     opcion = getchar();

     switch(opcion)
     {
          case '1':
               retiro = 20;
               saldo_fin = saldo_ini - retiro;

               if (saldo_fin < 0)
                    saldo_ins = 1;

               break;

          case '2':
               retiro = 50;
               saldo_fin = saldo_ini - retiro;

               if (saldo_fin < 0)
                    saldo_ins = 1;

               break;

          case '3':
               retiro = 100;
               saldo_fin = saldo_ini - retiro;

               if (saldo_fin < 0)
                    saldo_ins = 1;

               break;

          case '4':
               retiro = 200;
```

```
            saldo_fin = saldo_ini - retiro;

            if (saldo_fin < 0)
                 saldo_ins = 1;

            break;

       case '5':
            retiro = 500;
            saldo_fin = saldo_ini - retiro;

            if (saldo_fin < 0)
                 saldo_ins = 1;

            break;

       case '6':
            retiro = 1000;
            saldo_fin = saldo_ini - retiro;

            if (saldo_fin < 0)
                 saldo_ins = 1;

            break;

       default:
            printf("\n Cantidad a retirar(múltiplos de 100)? ");
            scanf("%d", &retiro);

            saldo_fin = saldo_ini - retiro;

            if (saldo_fin < 0)
                 saldo_ins = 1;

            break;

  }

  if (saldo_ins == 1)
       printf("\n Lo sentimos, no cuenta con fondos suficientes para
       hacer ese retiro.");
  else
       printf("\n Retiro autorizado por $ %d pesos. Su saldo final es
       de $ %d pesos.", retiro, saldo_fin);

}
```

Listado 3.9.14

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-9-14.c
Objetivo: Uso de la estructura repetitiva for.
------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int cuenta, suma;
     int  num1, num2;

     printf("\n Déme el primer número: ");
     scanf("%d", &num1);

     printf("\n Déme el segundo número: ");
     scanf("%d", &num2);

     suma = 0;

     for (cuenta = 1; cuenta <= num2; cuenta++)
         suma = suma + num1;

     printf("\n %d x %d = %d", num1, num2, suma);

}
```

Listado 3.9.16

```
/* ----------------------------------------------------------------
Programa: Ejercicio3-9-16.c
Objetivo: Uso de la estructuras repetitiva do...while.
----------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int cuenta, suma;
     int num1, num2;
     char resp;

     do
     {
          resp = 'n';

          printf("\n\n Déme el primer número: ");
          scanf("%d", &num1);

          printf("\n Déme el segundo número: ");
          scanf("%d", &num2);

          suma = 0;

          for (cuenta = 1; cuenta <= num2; cuenta++)
               suma = suma + num1;

          printf("\n %d x %d = %d", num1, num2, suma);

          printf("\n\n Desea realizar otro cálculo (s/n): ");

          /* Se requiere llamar dos veces a scanf() para quitar, de la
          entrada estándar, la tecla ENTER que se presiona para introducir
          el segundo número solicitado en la pregunta anterior. Si no se
          quita, la respuesta o si se desea realizar otro cálculo será
          SIEMPRE dicha tecla ENTER, es decir, para este programa se
          elegiría SIEMPRE no realizar otro cálculo.

          PRUÉBELO quitando una de las líneas con la llamada a scanf() */

          scanf("%c", &resp);
          scanf("%c", &resp);

     } while ((resp == 's') || (resp == 'S'));

}
```

Listado 3.9.18

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-9-18.c
Objetivo: Uso de la estructuras repetitiva do...while.
------------------------------------------------------------------*/
#include <stdio.h>

int main()
{
     int clitros;
     float plitro, tpagar;
     char resp;

     do
     {
          resp = 'n';

          printf("\n\n Déme la cantidad de litros a despachar: ");
          scanf("%d", &clitros);

          printf("\n Precio por litro: ");
          scanf("%f", &plitro);

          tpagar = clitros * plitro;

          printf("\n El total a pagar es $ %f pesos", tpagar);
          printf("\n\n Despachar gasolina a otro auto (s/n): ");

          scanf("%c", &resp);
          scanf("%c", &resp);

     } while ((resp == 's') || (resp == 'S'));

}
```

Listado 3.9.20

```
/* ------------------------------------------------------------------
Programa: Ejercicio3-9-20.c
Objetivo: Uso de la estructura repetitiva do...while.
------------------------------------------------------------------*/
#include <stdio.h>
#include <conio.h>      /* Prototipo de la función getch() */
#include <stdlib.h>     /* Prototipo de la función exit() */

int main()
{
     float total, num1, num2, respuesta;
     char resp;

     do
     {
          resp = 'n';

          printf("\n\n Déme el primer número: ");
          scanf("%f", &num1);

          printf("\n Déme el segundo número: ");
          scanf("%f", &num2);

          if (num2 == 0.0)
          {
               printf("\n\n No se puede dividir entre 0.");
               getch();
               exit(1);
          }
          else
               total = num1 / num2;

          printf("\n Intente adivinar el resultado: ");
          scanf("%f", &respuesta);

          printf("\n %f / %f = %f", num1, num2, total);

          if (respuesta == total)
             printf("\n Felicidades, acertó !!!");
          else
             printf("\n\n Lo siento, su respuesta es incorrecta");

          printf("\n\n Desea realizar otro cálculo (s/n): ");

          scanf("%c", &resp);
          scanf("%c", &resp);

     } while ((resp == 's') || (resp == 'S'));

}
```

Cuarta parte

Funciones

4. Funciones

4.1. Ejercicios

Ejercicio 1

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Usar funciones en C se llama modulación. ( )
2. Es imposible hacer llamadas a funciones ( )
3. Es más fácil interpretar un programa complejo con la separación del mismo en funciones. ( )
4. Una función es una parte del programa que realiza operaciones bien definidas. ( )
5. La función puede devolver un valor con la instrucción goto. ( )
6. Al llamar a una función proporcionamos un conjunto de datos llamados parámetros. ( )
7. La función *main( )* es una función complementaria opcional. ( )
8. La cabecera de la función es el conjunto de instrucciones encerradas entre llaves y, en él, se ejecutan las acciones de la función. ( )
9. El tipo de función se encuentra en el cuerpo de la función. ( )
10. Se llama a una función con su nombre y poniendo entre paréntesis sus parámetros. ( )

Ejercicio 2

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Función	( )	1.- Esta instrucción devuelve un solo valor del tipo declarado en la cabecera de la función.
b) Clases de almacenamiento	( )	2.- Es el conjunto de datos que se introducen a la función.
c) Cabecera de la función	( )	3.- Son las clases de almacenamiento en una función.
d) return	( )	4.- Es la forma en la que el lenguaje C permite la modularidad, ya que se pone el nombre de la función a ejecutar.
e) #include	( )	5.- Esta palabra indica que la función no devuelve un valor.
f) Lista de parámetros	( )	6.- Modifican el ámbito o pertenencia de la variable dentro del programa.
g) extern, auto, static y register	( )	7.- Estas variables son usadas en una o todas las funciones, se encuentran declaradas fuera de la función principal.
h) void	( )	8.- Realiza una operación o procedimiento especial dentro de un programa, a cada una de estas las llamamos módulos.
i) Variables globales	( )	9.- En ella se especifica el tipo de dato que devuelve, la lista de parámetros que recibe y el nombre de una función.
j) Llamadas a las funciones.	( )	10.-Es la directiva utilizada para incluir los archivos de cabecera que contienen el conjunto de funciones predefinidas por el compilador.

Ejercicio 3

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Ventaja de una función	Parámetro por valor	Iteración	Parámetro por referencia
Recursividad	Automáticas	Registro	
Estáticas	Cabecera	Parámetros formales	

Enunciados:

1.- La _____ es cuando una función se llama a sí misma.

2.- Un _____ consiste en transferir una copia del valor de una variable hacia una función.

3.- Un _____ es un apuntador a la variable que se pasa a una función, cuando se realiza una llamada a esta.

4.- Los _____ son creados y utilizados únicamente dentro de la función donde son declarados.

5.- Las variables locales _____ se crean al entrar a la función y se destruyen al salir de ella, se indican con la palabra clave *auto*.

6.- Las variables locales _____ no son destruidas al salir de la función.

7.- La variables locales de _____ son variables de tipo carácter o entero, que se almacenan en un registro de la CPU y no en memoria.

8.- La _____ de una función está formada por el tipo de datos que devuelve, su nombre y su lista de parámetros.

9.- Una _____ es que se puede llamar desde diferentes partes del programa.

10.- La _____ es cuando una función se llama en repetidas ocasiones desde un mismo punto del programa.

Ejercicio 4

Instrucciones: Lea con atención cada uno de los enunciados que enseguida se muestran. De las opciones enlistadas después de ellos, subraye solamente el inciso cuya definición corresponda correctamente con el enunciado.

1.- Momento en que se enuncia la función para ser utilizada.

a) Destrucción de la función
b) Llamada a la función
c) Declaración
d) Parámetros de la función

2.- Término para la declaración de una función que no retorna un valor.

a) int
b) void
c) Recursiva
d) Estático

3.- Las funciones dividen un programa en…

a) Variables
b) Instrucciones
c) Módulos
d) Constantes

4.- Contiene un conjunto de funciones en el lenguaje C.

a) Biblioteca
b) Encabezado
c) Fichero
d) Bloque

5.- Función para introducir datos al programa, por parte del usuario.

a) printf ( )
b) time ( )
c) scanf ( )
d) return

6.- Desde la función *main( )* se pueden hacer llamadas a funciones almacenadas en un archivo conocido como…

a) Biblioteca
b) Encabezado
c) Variables
d) Bloque

7.- Son los que transfieren los datos a una función.

a) Variables b) Parámetros c) Recursividad d) Declaración de tipo

8.- Estas funciones son utilizadas en el proceso de depuración de un programa, incluyen las funciones *exit*( ) y *assert*( ).

a) Matemáticas b) Aleatorias c) De carácter d) De utilería

9.- Palabra reservada utilizada para devolver un valor con la función.

a) Función b) return c) Array d) Formales

10.- Estas funciones incluyen los procedimientos para la manipulación del tiempo.

a) De tiempo b) Aleatorias c) De carácter d) Utilidad

4.2. Ámbito y clase de almacenamiento de datos

Ejercicio 5

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Las variables se almacenan de acuerdo a su tipo de dato. ( )
2. El ámbito se refiere a que la variable es local o global. ( )
3. En el lenguaje C, los modificadores de tipo de variable se utilizan de manera diferente a otros lenguajes. ( )
4. La palabra reservada int es un modificador. ( )

5. El ámbito es la parte del programa en que la variable es reconocida ( )
6. register es una palabra reservada utilizada como modificador de tipo de variable. ( )
7. Para declarar una variable global siempre utilizamos la palabra reservada extern. ( )
8. Cuando se declara una variable fuera de todas las funciones, su ámbito es local. ( )
9. Las variables globales siempre están dentro del cuerpo de una función. ( )
10. Una variable se considera automática, ya que asigna un espacio de memoria al llamar a la función que la contiene, y se libera dicho espacio al salir de la función. ( )

4.3. Biblioteca de funciones

Ejercicio 6

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Biblioteca estándar	( ) 1.- Archivo de cabecera de las funciones matemáticas.
b) Funciones predefinidas	( ) 2.- En este archivo se encuentran las funciones de entrada y salida de datos.
c) time.h	( ) 3.- Conjunto de funciones que dan soporte a las operaciones básicas que permite realizar el lenguaje C.

d) math.h	( ) 4.- Define una variedad de macros de utilería.
e) string.h	( ) 5.- Archivo de cabecera con funciones de manejo de tiempo y hora.
f) stdio.h	( ) 6.- Es otra manera de llamar a las funciones de la biblioteca estándar.
g) stdlib.h	( ) 7.- Es una función de manejo del tiempo.
h) asctime()	( ) 8.- Archivo con funciones de manejo de cadenas de caracteres.

Ejercicio 7

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Prototipo	Archivos	Función	Memoria
Globales	Recursivas	Apuntadores	Estáticas
return	Encabezado	Argumentos	Locales
Parámetros	stdio	scanf	printf
void	Variables	include	Biblioteca

Una _____ es un bloque de programa que realiza una operación específica, a su declaración la podemos llamar _____. Utilizamos una función llamada _____ para que el usuario introduzca datos y, _____, para que muestre datos en pantalla. Con la declaración #_____ abrimos los _____ necesarios para poder llamar a las funciones de _____. Uno de estos archivos es _____.h.

Las _____ _____ se declaran cuando se encuentran fuera de la función *main()* y, las variables _____, se declaran dentro de ella o de las funciones creadas por el programador. Existen, además, las variables _____, que se mantienen en _____ durante la ejecución del programa.

En una función se utilizan los _____ para almacenar la dirección de memoria donde se encuentra un dato. Además, en el _____ de la función introducimos la lista de _____ con los datos de entrada hacia la función. A los parámetros también los llamamos _____. Dicha función devuelve un dato con la palabra reservada _____ y, si no devuelve un valor, se utiliza la palabra reservada _____. Cabe mencionar que las funciones son _____ , cuando se llaman a sí mismas.

Sopa de letras

Instrucciones: Encuentre las palabras siguientes en esta sopa de letras: Variable, encabezado, scanf, recursividad, parámetro, puntero, función, include, memoria, printf, global, fichero, return, prototipo, biblioteca, argumento, stdio, locales, estática y void.

S	D	A	R	T	Q	P	R	O	T	O	T	I	P	O	R	S	R	H	K
F	I	C	H	E	R	O	Q	U	D	F	B	S	D	E	T	R	R	O	H
T	U	V	L	Q	E	F	G	J	A	S	D	A	P	U	N	T	E	R	O
B	H	N	J	T	F	D	S	C	C	G	Y	U	I	K	N	Q	C	R	G
O	V	B	C	T	F	X	E	S	T	A	T	I	C	A	M	A	U	D	J
I	X	N	C	I	S	T	A	S	D	R	T	L	K	A	J	S	R	S	B
P	N	A	G	L	O	B	A	L	G	F	P	W	I	H	Y	D	S	E	Q
Q	J	C	O	I	F	N	U	E	H	A	S	R	H	K	R	C	I	G	S
E	D	S	L	P	L	Ñ	O	W	O	J	O	G	I	L	E	V	V	Y	D
W	S	B	A	U	S	I	Q	W	E	M	R	T	Y	N	Y	I	I	E	F
U	I	S	D	F	D	O	Q	R	E	T	U	R	N	G	T	Y	D	Q	G
B	A	S	D	T	H	E	T	M	Z	X	C	V	B	N	M	F	A	U	N
D	W	H	S	H	J	F	G	D	S	A	V	T	R	E	W	Q	D	I	K
A	D	F	P	A	R	A	M	E	T	R	O	A	Y	U	I	O	P	Y	U
Ñ	A	S	D	F	G	D	S	A	Q	W	I	R	R	U	I	P	F	R	W
E	O	K	J	A	W	E	R	T	Y	Q	D	T	Y	I	D	N	S	E	R
A	R	G	U	M	E	N	T	O	S	F	A	L	O	C	A	L	E	S	G
E	E	R	Y	A	O	H	T	R	F	V	S	D	S	C	Q	B	D	W	H
T	E	N	C	A	B	E	Z	A	D	O	W	R	S	O	T	I	L	A	S
S	J	L	U	R	E	B	D	T	U	W	U	R	Y	O	P	B	V	E	X

Crucigrama

HORIZONTALES

3.- Cuando una función se llama a sí misma.

5.- Las funciones de _____ son las que tiene el lenguaje C de manera predeterminada.

6.- Son los argumentos que utilizamos en la llamada a una función.

8.- El _____ de una función contiene los parámetros de la función y el tipo de dato que devuelve.

9.- Se puede utilizar como lista de parámetros de una función que no recibe valores.

10.- Recursividad donde intervienen más de dos funciones.

12.- Las funciones _____ por el programador, son las que realizan tareas específicas en el programa donde se encuentran.

13.- Las variables de _____ son aquellas que se almacenan en un lugar de acceso rápido.

15.- Son los argumentos que utilizamos en la definición y/o declaración de una función.

16.-Las funciones de _____ son las creadas por el programador, de acuerdo a sus necesidades.

17.- Nombre con el que se denomina a la declaración de una función.

19.- Cuando se hace referencia al acceso a una función, desde cualquier parte del programa, se le denomina _____ a la función.

20.- Bloque de código que realiza una operación o procedimiento en especial, dentro de un programa. A cada una de ellas las llamamos módulos.

VERTICALES

1.- La lista de _____ contiene los datos que se introducen a la función.

2.- En él se encuentra la instrucción o instrucciones de la función.

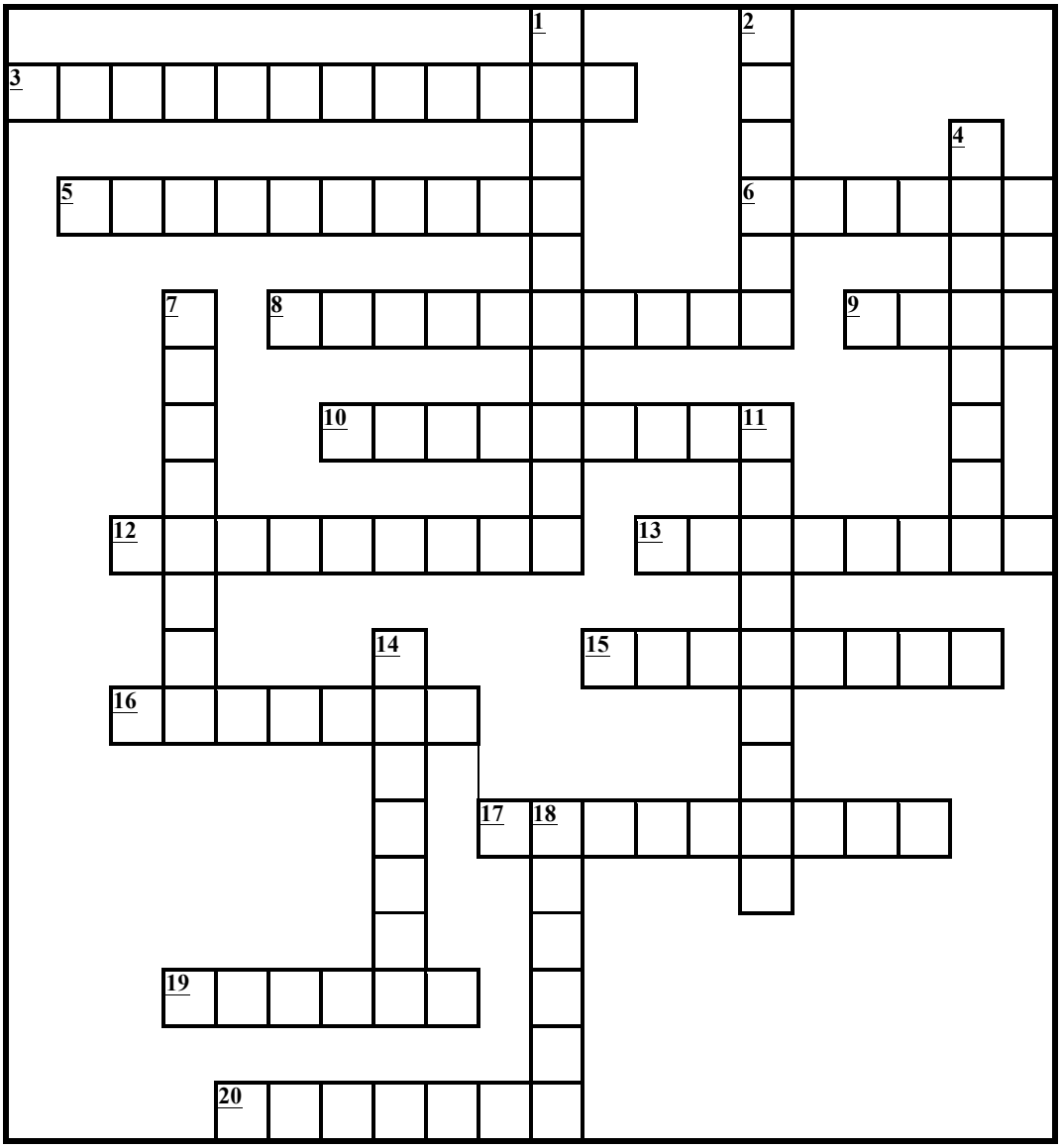
4.- Palabra reservada que declara una variable de tipo registro.

7.- En los _____ de cabecera (con extensión de archivo *.h) están las funciones de biblioteca.

11.- Es otra forma de llamar a un parámetro.

14.- Es la recursividad que se realiza dentro de la misma función.

18.- Palabra reservada utilizada para devolver un valor de una función.



4.4. Ejercicios prácticos

Para simplificar la revisión de este documento, se aclara que los ejercicios están alternados, es decir, en el apéndice de este apartado se presenta la solución de un ejercicio y se propone que el siguiente sea resuelto por el lector.

El significado de la nomenclatura utilizada, por ejemplo "Ejercicio 2.4.3", tiene la explicación siguiente: el primer número corresponde al apartado; el segundo corresponde al tema específico; y, el tercero, es el número de ejercicio, es decir, apartado 2, tema 4 y ejercicio o programa número 3.

Por otra parte, hay dos tipos de prácticas, las prácticas *normales* y las de *paso a paso*. Para las prácticas normales se recomienda utilizar los ejercicios propuestos para que sean resueltos por el lector y, para las prácticas paso a paso, guiarse con los ejercicios resueltos, sólo se debe seguir el procedimiento siguiente:

1. Encender el equipo de cómputo.
2. Abrir sesión del ambiente gráfico Windows.
3. Ejecutar la aplicación que se tenga instalada para editar los programas escritos en lenguaje C.
4. Transcribir el programa en cuestión.
5. Guardar el archivo que contiene el programa transcrito.
6. Compilar el programa.
7. Iniciar un proceso de depuración del programa (corrección de errores) hasta cerciorarse de que el programa cumpla con el objetivo establecido.
8. Imprimir el resultado de la ejecución o ejecuciones del programa (paso opcional).

Ejercicio 4.2.1: Frecuentemente, las amas de casa cuando quieren preparar una receta se encuentran con que algunas proporciones se manejan en onzas (unidad de medida de peso inglesa) y eso dificulta su elaboración. Con el uso de la estructura de funciones, desarrolla un programa en C que convierta de onzas a gramos, partiendo de que 1 gramo = 0.035270 onzas. Las nombres para las funciones son: pedir(), convertir() e imprimir(), sin parámetros.

a) **Análisis:** A partir de este ejercicio será necesario que se piense de manera más modular para que se pueda dividir el problema en pequeños problemas y que cada problema sea resuelto de manera individual, por lo que en este caso podemos dividirlo en 3 problemas y esto sería: datos de entrada, proceso o cálculo y salida de datos. Es por eso que se sugiere el uso de 3 funciones, ya que cada función representa cada solución individual.

b) **Codificación:** Una vez desmenuzado el problema en módulos o funciones, la tarea siguiente es determinar el contenido de esta función y, en este caso, la función pedir(), contendrá las instrucciones para solicitar los datos; convertir(), las operaciones necesarias para obtener el resultado; e, imprimir(), permitirá mostrar en pantalla el resultado planteado en el problema original. Otra consideración inicial, como en este caso la función principal main(), solo llamará o invocará a las funciones y, estas a su vez como no

manejaran argumentos o devolverán valores, será importante entonces declarar las variables necesarias como globales, de tal forma que puedan ser empleadas en todas las funciones.

c) **Comprobación**: Cuando se ejecute este programa y se pongan los datos para las conversiones, no se notará nada diferente a la manera en que se ha venido programando. Por lo anterior, se sugiere el uso de una ejecución paso a paso, que permita visualizar cómo se hacen las llamadas a las funciones; cómo nos permite el lenguaje de programación, mediante esta técnica, transferir el control y la secuencia de ejecución fuera de main() y regresar, para continuar con la siguiente instrucción. Y, si es otra función, entonces se repetirá el proceso de ir al cuerpo de la función y ejecutar una a una las instrucciones que la conforman hasta concluir. Sólo en ese momento se regresaría a la función principal main().

Ejercicio 4.2.2: Utilizando los mismos nombres de funciones del ejercicio 4.2.1, es decir: pedir(), convertir() e imprimir(), sin parámetros; desarrolla un programa en C que convierta de kilos a libras y, esto, porque generalmente los agricultores exportan sus productos a Estados Unidos y ellos les piden que manden la fruta o verdura en bolsas o cajas etiquetadas con su peso en libras. Conversión que se les dificulta a los agricultores. Parte del hecho de que 1 Kg = 2.205 libras.

a) **Análisis:** Este ejercicio está planteado con la misma estructura del ejercicio 4.2.1. y el propósito es que se continúe ensayando con la modularidad, es decir, con el manejo de funciones, para que de esta forma comience a apreciar las ventajas que ofrece esta técnica donde cada función realizará una tarea específica, entre ellas, el hecho de que si el programa no está haciendo bien la lectura de datos, hay errores en los cálculos o simplemente no aparece el resultado, directamente se dirige a la función que realiza esa tarea y se corrige la falla. Una ventaja más sería que si se pide un cambio al programa, sencillamente se hacen los cambios pertinentes en la función a modificar.

b) **Codificación:** Ya se habló del contenido de las funciones pedir(), convertir() e imprimir(), pero para ser más precisos, en el caso particular del contenido de la función convertir(), es importante precisar que si se conoce la cantidad en kilos para obtener las libras, la operación a realizar es la multiplicación y con eso estaríamos encontrando el resultado solicitado. Respecto a la declaración de las variables, nuevamente se sugiere que sean declaradas como globales, de tal forma que puedan ser empleadas en todas las funciones y su valor se conserve.

c) **Comprobación**: Otra vez se propone el uso de una ejecución paso a paso, para poder seguir visualmente cómo se transfiere la secuencia, en la ejecución, a las funciones cuando éstas son llamadas una a una; y ejecutar, secuencialmente, cada una de las instrucciones hasta concluir. Sólo en ese momento se regresaría a la función principal main(). Con esto, no estamos dejando de lado la comprobación al sugerir datos sencillos para verificar el buen funcionamiento del programa, como en este caso, que si se introduce una cantidad de 10 Kg el resultado deberán ser 22 libras, esa es la idea de introducir cifras para las que,

mentalmente, podamos encontrar el resultado para verificar la exactitud del programa. Probado esto, puede intentar con su propio peso en kilos, para lo que la computadora dará su peso equivalente en libras. Así, es factible que pueda buscar otras aplicaciones a estos ejercicios que venimos desarrollando.

Ejercicio 4.3.1: Existen programas de compatibilidad de parejas, donde toman en cuenta varios aspectos, entre ellas sus preferencias, religión, profesiones y otras. Colabora con este tipo de programas incorporando otro aspecto más que sería la edad. El objetivo es que el programa informe sobre la diferencia de edades entre los enamorados e informe si hay una posible compatibilidad cuando dicha diferencia sea menor a 5 años o incompatibilidad si es mayor a 5 años y cuál de los dos enamorados es menor que el otro. Desarrolle el programa que utilice funciones con el paso de parámetros por valor, pero que no devuelva un valor. El nombre de la función sugerida es *compara(edad1, edad2)*.

a) **Análisis:** En este ejercicio y en los siguientes nos centraremos en las funciones, porque ahí se encuentra la parte medular del programa, aunque en este caso si bien es cierto que se pueden crear más funciones, como son solicitar datos o imprimir resultados, sólo se nos está pidiendo el emplear mínimo una función que cubra la parte del proceso, es decir los cálculos u operaciones necesarias para encontrar el resultado. En este caso, con una resta podremos saber si existe o no diferencia de edades y en la misma función podríamos analizar esa diferencia para sacar el mensaje apropiado, pero podemos respetar el objetivo de las funciones, el cual consiste en realizar una sola tarea. Por lo que, sean en la función principal o en otra función haremos el análisis respectivo.
b) **Codificación:** Continuando con la dinámica de centrarnos en las funciones es importante considerar que si esta función va a recibir un par de valores, entonces habrá que declarar del mismo tipo de datos a los parámetros que recibirán dichos valores. Esto con el propósito de que no haya pérdida de información o errores en el programa. Así, por ejemplo, si las variables son del tipo entero, los argumentos de la función serán del mismo tipo. Una consideración más es el hecho de que si esta función no devuelve un valor, el resultado tendrá que ser guardado en una variable global para que de esa manera pueda ser utilizado en otras funciones. Respecto al análisis de diferencia de edades, como el programa no pide propiamente un resultado sino una interpretación de ese resultado, tenemos entonces que usar la estructura de control **if-else** tantas veces como sea necesario, para que avise de la compatibilidad o de la incompatibilidad.
c) **Comprobación**: Resulta interesante continuar empleando el uso de una ejecución paso a paso para poder seguir visualmente cómo se transfiere la secuencia de la ejecución a las funciones, cuando éstas son llamadas, pero en este caso más, porque se podrá observar cómo toman un valor los argumentos de la función y cómo es utilizado ese valor mientras dura la ejecución de la función que recibe dichos argumentos. Finalmente, cuando se le pidan en la ejecución las edades, escriba datos donde la diferencia de edades sea menor a

5 años, sea mayor a 5 o no exista diferencia de años, es decir, que la pareja tenga la misma edad. Observe los diferentes mensajes que aparecerán, ¿son los correctos?

Ejercicio 4.3.2: En un programa de TV del tipo *reality show*, el concurso es de personas con sobrepeso, dicho concurso consiste en expulsar a uno de dos concursantes y que será aquél que después de una semana su peso sea mayor al de su oponente, porque lo que el objetivo es que el programa en C informe al jurado a cuál de los dos concursantes expulsará y cuál es la diferencia de peso entre ambos. La función sugerida es *compara(peso1, peso2)*.

a) **Análisis:** Nuevamente se nos está pidiendo el emplear mínimo una función que cubra la parte del proceso para encontrar el resultado, en este caso también con una resta podremos saber si existe o no diferencia de peso y ya, en la función principal o en otra función, haremos el análisis respectivo. Una vez obtenida la diferencia, tenemos que determinar cuál de los dos concursantes tiene un mayor peso y mediante un mensaje avisarle al jurado para que proceda con la expulsión.
b) **Codificación:** Como esta función va a recibir un par de valores, no olvidar que en la declaración de sus argumentos los tipos de datos tienen que coincidir con los valores que se recibirán en la función. Así, por ejemplo, si las variables son del tipo float, los argumentos de la función serán del mismo tipo. Otra vez, no olvidar que si esta función no devuelve un valor, el resultado tendrá que ser guardado en una variable global para que de esa manera pueda ser utilizado en otras funciones. Respecto al análisis de diferencia de peso, como el programa no pide propiamente un resultado sino una interpretación de ese resultado, tenemos entonces que usar la estructura de control **if-else** tantas veces como sea necesario para que avise cuál de los dos concursantes será expulsado, cuando su peso sea mayor a su oponente.
c) **Comprobación**: Nuevamente se recomienda ampliamente el empleo del uso de una ejecución paso a paso para que nos ayude a entender cómo trabaja el lenguaje C la parte de la modularidad. Referente a probar el programa solo hay dos casos a probar: cuando el primer oponente pesa más que el segundo después de una semana y, el otro caso, cuando el segundo oponente cuenta con un peso mayor a su rival. Pruebe ambos casos, para ver qué pasa en cada situación y a quién se está expulsando en el programa en cada caso.

Ejercicio 4.4.1: Elaborar un programa en lenguaje C que simule el funcionamiento de una calculadora, es decir que pida dos números y que dé la opción de seleccionar la operación a realizar. Resuélvase mediante la construcción de funciones para las operaciones de suma y división de tal forma que las mencionadas funciones regresen el resultado.

a) **Análisis:** El planteamiento es simple, sin embargo requiere mucho análisis, comencemos. Lo primero es que si el usuario puede elegir el tipo de operación, como en una calculadora, entonces requeriremos de un menú, donde se listen las opciones posibles.

Segundo, como solamente se habla de habilitar la solución con las funciones, arovechamos el hecho de que, por ejemplo para sumar, necesitamos mínimo dos números; lo que nos ayuda a deducir que las funciones requerirán de parámetros, dos en específico. También se señala que la misma función, una vez encontrado el resultado, nos devuelva el total. Atención especial le debemos dedicar a la división porque cualquier calculadora, por sencilla que sea, impide la división entre 0 (cero). Así que, si nosotros queremos cuidar este aspecto, tendremos que verificar a través de una estructura de control (**if**) que, si el segundo número es 0 (cero), se avise con un mensaje del error y se evite que el programa falle.

b) **Codificación:** Dando prioridad a la implementación de las funciones, este caso se presta de manera excelente para entender el hecho de que una función puede devolver valores, ya que después de sumar o dividir, tenemos un total y ese dato lo necesita el programa en otra parte para informar al usuario del resultado de la operación. Por eso, en este caso, la instrucción **return** nos auxiliará con la parte de devolver un valor. Una consideración más es declarar en este caso la función del tipo de dato que arroje el resultado de la operación y, a su vez, si se emplea una variable que almacene temporalmente dicho resultado, también tendrá que ser del mismo tipo con el que la función fue declarada. Recordar que, de acuerdo al análisis previo, estas funciones requerirán de los parámetros respectivos. En cuanto al resto de la implementación es innegable que sí se requiere el uso de un menú, en el cual aparezcan las operaciones permitidas por nuestra calculadora, construido por medio de la estructura de control **switch**().

c) **Comprobación**: Puede seguir el empleo de la ejecución paso a paso, a menos que ya quede comprendida del todo, pero en este caso sí será importante al menos hacer dos operaciones, una suma y una división y, en el caso de la división, verificar que no se intente dividir entre 0 (cero).

Ejercicio 4.4.2: Complementa el programa del ejercicio 4.4.1 para que realice las funciones de restar y multiplicar.

a) **Análisis:** Como se trata de complementar al ejercicio previo, el análisis es más simple, porque basta recordar que para estas operaciones, resta y multiplicación, también necesitamos dos números, lo que nos lleva a concluir que las funciones requerirán otra vez de parámetros, dos en específico. Ahora bien, no se dice en el planteamiento, pero debemos suponer que si se trata de una implementación homogénea estas funciones también van a devolver un resultado.

b) **Codificación:** Para incorporar este par de funciones a la implementación previa, se tendrá que iniciar por ampliar el menú construido con **switch**(), para que el usuario pueda observar que ahora también puede restar y multiplicar. Respecto a las funciones nuevas no olvidar el hecho de que, al momento de declarar la función, el tipo de dato que devuelve deberá ser igual al que arroje el resultado de la operación y, a su vez, si se emplea una variable que almacene temporalmente dicho resultado, también tendrá que

ser del mismo tipo con el que la función fue declarada. Recordar que de acuerdo al análisis previo estas funciones requerirán de los parámetros respectivos.

c) **Comprobación**: Una vez más, puede seguir el empleo de la ejecución paso a paso si así lo considera necesario. Referente a la comprobación, pruebe nuevamente con al menos dos operaciones y, en el caso particular de la resta, ponga un valor más pequeño para el primer número que para el segundo, ¿cuál fue el resultado? ¿un valor positivo o negativo?

Ejercicio 4.4.3: Calcular la ganancia de las televisoras cuando dan premios de autos o dinero en los concursos o rifas a través del envío de mensajes de texto. Suponer que cada mensaje cuesta $3.85 y preguntar el monto del premio y el total de mensajes recibidos. Realizar una función en lenguaje C (dentro de un programa completo) que reciba el número total de mensajes y regrese el monto total.

a) **Análisis:** La lectura del planteamiento nos arroja que se da más libertad para implementar la solución, ya que sólo se pide el uso de una función que devuelva el ingreso por conceptos de mensajes recibidos, sin embargo se podría utilizar una segunda función para conocer la utilidad o ganancia, pero eso ya quedaría a criterio del programador. Como lo que se busca promover es la cultura de la modularidad, sí es altamente recomendable el desarrollar todas las funciones que se consideren necesarias. Es decir, que los datos se pueden pedir también en una función.

b) **Codificación:** Al parecer este programa no requiere del uso de estructuras de control, ya que la función solicitada simplemente multiplicará ambos parámetros y devolverá el resultado. Sin embargo, con ayuda de las estructuras de control repetitivo se podría enriquecer esta solución, pensando en que la televisora tiene más de un concurso y que se requiere conocer de cada uno de ellos la ganancia que arrojó.

c) **Comprobación**: Para probar la solución, trate con datos de tal manera que primero sea mayor el premio que el ingreso y tome nota del resultado, luego intente meter datos con cifras muy aproximadas a las reales y se sorprenderá de lo redituable que son esos concursos para las televisoras.

Ejercicio 4.4.4: En la final de futbol mexicano, las televisoras tienen jugosas ganancias con los anunciantes, ya que el costo por anuncio se incrementa notablemente. Elabora un programa que calcule el ingreso por ese concepto, de acuerdo al recuadro siguiente:

Nº	Duración del anuncio	Costo
1	15 segundos	$100,000.00
2	30 segundos	$250,000.00
3	45 segundos	$500,000.00

Con base en los minutos que se destinen para comerciales durante el medio tiempo, informar cuántos anuncios se logran transmitir de manera combinada, de tal manera que el ingreso sea mayor a los 5 millones de pesos. Para facilitar la solución se debe crear al menos una función que, de acuerdo al número de comerciales y al costo por comercial, informe cuál sería la cantidad obtenida total. Para facilitar la solución, que el programa pregunte al usuario cuántos comerciales de cada tipo requiere, le avise si con esa combinación es suficiente y cuántos minutos requeriría esa combinación.

a) **Análisis:** Seria interesante lograr que el programa desarrollara las combinaciones posibles de tal manera que informará únicamente aquellas combinaciones que dan como resultado una cifra mayor a los 5 millones de pesos y también aquéllas que emplean el menor tiempo posible. Sin embargo, como eso requiere de un manejo de las matemáticas más avanzado, nos limitaremos a implementar lo que se nos pide y que sea el usuario quien escoja aquellas combinaciones que le parezcan más adecuadas. Por este motivo, se le pregunta al usuario cuántos comerciales requiere de cada tipo y con esos datos se le informa el monto total y el tiempo aire en televisión que llevaría la transmisión de dichos comerciales.
b) **Codificación:** De nuevo nos encontramos con la necesidad de implementar una función que maneje al menos dos argumentos, es decir, el número de comerciales y el costo de cada comercial. Con esto, el programa puede decirnos el monto obtenido, es decir, que vamos a requerir del manejo de parámetros y la devolución de valores. Respecto a las estructuras de control se pueden emplear para que el usuario elija la duración del comercial y, con base en eso, manejar el precio por comercial. El resto del programa se puede resolver con multiplicaciones y sumas, para que se nos diga la duración de los mensajes, ya sea los segundos totales o aplicando una pequeña conversión para mostrarlo en minutos. Lo que sí es un hecho es que se requerirá de la estructura de control simple **if()** para comparar el monto total de todos los comerciales contra los 5 millones de pesos que se ponen como base, a fin de reportar si el monto obtenido es superior o no. Cuidar que la declaración de las variables tendrá que ser del tipo entero largo para que no se pierda precisión en los resultados.
c) **Comprobación**: En la ejecución del programa sería interesante probar con un solo tipo de comercial y ver cuánto tiempo lleva alcanzar los 5 millones. Luego se puede intentar con otro y así sucesivamente hasta probar las tres duraciones distintas. No olvidar apuntar la duración en cada caso, para luego emplear a criterio un número determinado de cada tipo de duración de comercial y ver si de manera combinada el tiempo es menor que al elegir comerciales con una sola duración.

Ejercicio 4.6.1: Un estudiante camina 30 cuadras para llegar a su escuela, cada cuadra la camina en 45 segundos, ¿qué tiempo le lleva llegar a la escuela? Ahora bien, si sale de su casa a las 6:45 A. M. y las clases inician a las 7:00 A. M., por lo que a él se le hace tarde, sugerirle la hora adecuada de salida. La función propuesta es *tiempo()*, que permita convertir segundos a minutos para verificar si llega tarde o no. Desarrolla un programa con manejo de funciones que resuelva

el planteamiento anterior y posteriormente guardarlo para que pueda ser utilizada la función *tiempo()* para la solución del problema 4.6.2.

a) **Análisis:** Lo curioso de este planteamiento es que ya se dan todos los datos, entonces sólo es meramente informativo en el sentido de que se avisará si la hora a la que se levanta el estudiante es la adecuada o requiere levantarse a una hora diferente para que alcance a llegar puntualmente. Además, hay que decirle cuántos minutos le lleva hacer ese recorrido a pie.
b) **Codificación:** La función que se está proponiendo es necesaria porque al multiplicar las cuadras por el tiempo de recorrido por cuadra el resultado es en segundos, y como la hora de salida y llegada del estudiante está en minutos, existe una necesidad de unificar las unidades de tiempo, ya sean todas en segundos o todas en minutos. En el caso presente se especifica que todo se maneje en minutos. Para complementar la implementación del código es necesario apoyarse en la estructura de control **if-else**, para comparar el tiempo del que dispone el estudiante para llegar, contra el tiempo que le lleva recorrer esa distancia. Del resultado de esa comparación, sugerirle que se levante más temprano o que camine más aprisa cada cuadra.
c) **Comprobación**: En este caso solo es ejecutar el programa y observar los resultados. Si se quisiera un resultado diferente, internamente se puede modificar la velocidad de cuadra, por ejemplo en lugar de que sean 45 segundos, que sean 30 segundos y probar de nuevo el programa. También, si suponemos que el estudiante usa bicicleta, poner como velocidad 15 segundos y observar los resultados, tal vez esa sería otra solución, el proponerle que use un medio de transporte como la bicicleta o la moto y de esa manera se puede continuar saliendo a la misma hora de su casa.

Ejercicio 4.6.2: Un joven limpiaparabrisas limpia un parabrisas en 15 segundos y un semáforo en rojo dura 30 segundos. El joven labora un número variable de horas. Calcula su ingreso ideal y su ingreso real por día. Suponiendo que limpia al menos un auto en cada señal roja o en cada alto del semáforo y que en cada limpiada recibe un peso. Crea una función que calcule su ingreso y reciba como parámetro las horas trabajadas. Desarrolla un programa con manejo de funciones que resuelva el planteamiento anterior**. La función sugerida es la misma función tiempo() del ejercicio 4.6.1, que permita convertir segundos a minutos para verificar el número de parabrisas limpiados por minuto.**

a) **Análisis:** Una ventaja que ofrece esta solución es que, si se resolvió satisfactoriamente el ejercicio 4.6.1, podremos utilizar la función para convertir segundos en minutos y de esa forma determinar cuántos parabrisas limpia por minuto el joven y, al recibir las horas trabajadas, podamos calcular su ingreso diario. En cuanto a los supuestos existen varios, uno de ellos es que para el ingreso ideal es necesario tomar en cuenta que en cada señal roja el joven alcanza a limpiar dos parabrisas en 30 segundos y con eso obtiene $2.00 pesos de ingreso. Esto nos llevaría a pensar que el ingreso ideal es el doble del ingreso

real. Aquí viene otro supuesto y es que en cada limpiada le dan un peso y, como mínimo, limpia un parabrisas por señal roja. El tercer supuesto viene con el hecho de que será necesaria la conversión en algún momento determinado de las horas proporcionadas por el usuario a minutos, y así poder calcular ambos ingresos, el ideal y el real.

b) Codificación: El uso nuevamente de la función tiempo es por la necesidad de pasar los segundos empleados en limpiar parabrisas a minutos, para saber cuántos parabrisas se alcanzan a limpiar por minuto. Una conversión adicional es pasar las horas trabajadas en minutos, y analizar si se puede utilizar la misma función, o si es necesaria una segunda que se implemente para convertir las horas a minutos. Para calcular el ingreso, sea ideal o real, de obtenerse con una función se requerirá de un par de parámetros para conocer el tiempo en minutos y la cuota recibida. Si se calcula sin función, de todos modos se ocupará multiplicar los minutos trabajados por día contra el número de parabrisas limpiados en un minuto y eso, a su vez, multiplicado por la cooperación recibida. En el análisis se dedujo que el ingreso ideal sería el doble al ingreso real, entonces obtenido el ingreso real ese total se duplica y obtenemos el ideal.

c) Comprobación: Al probar este programa, suponga que trabajó 1 hora y vea el resultado, ahora si se vuelve a ejecutar, escriba 2 horas y el ingreso deberá duplicarse. De ser correctos los datos, se puede calcular con cualquier número de horas.

Ejercicio 4.7.1: La sumatoria es una serie de sumas sucesivas desde la unidad hasta un límite cualesquiera.

Ejemplo:
S(4) = 4 + 3 + 2 + 1

o también:
S(4) = 4 + S(3)
S(3) = 3 + S(2)
S(2) = 2 + S(1)
S(1) = 1 + S(0)
S(0) = 0 ==> Base o límite.

Es decir, entonces, que *S(N) = N + S(N - 1)*; por lo que se aprecia que se trata de una fórmula recursiva para todo valor N > 0. Desarrolle un programa que resuelva este planteamiento con una función recursiva.

a) Análisis: La solución es muy simple y de hecho puede ser resuelta con una estructura de control repetitiva. Sin embargo, una solución alternativa es utilizar la recursión que ofrece el lenguaje C, la cual permite los llamados de una función a sí misma, las veces que sean necesarias, y definir, a través de una estructura de control condicional, la forma de interrumpir los llamados (evitando una recursión infinita). Al ser un planteamiento muy completo lo único que nos queda es decir que se ocupa una función y que, en su cuerpo, se requiere una suma tal y como se describe en la fórmula, para que al final se obtenga el total solicitado.

b) Codificación: Para el programa se debe solicitar el número que marque el fin de la sumatoria, matemáticamente decimos que se debe solicitar el límite superior de la

sumatoria. La función por implementar deberá hacer el resto que, en este caso, es auto-llamarse hasta realizar la suma de todos los valores inferiores a dicho valor proporcionado por el usuario. Con esto, la función requerirá de un parámetro del tipo entero y devolverá un valor del mismo tipo una vez sumado, por lo que es indispensable al declarar el tipo de función, que éste sea del mismo tipo que la variable que almacene el total de la suma. Por último se imprime la suma total que resuelva el planteamiento solicitado.

c) **Comprobación**: Anteriormente se recomendó la ejecución paso a paso, para este caso es indispensable, además, activar el depurador, a fin de realizar un seguimiento de cómo va cambiando parcialmente el valor de la suma. También se podrán seguir las auto-llamadas recursivas de la función. De otra manera, si sólo se ejecuta el programa y se observa el resultado, no podrá apreciar cómo funciona el concepto recursivo.

Ejercicio 4.7.2: Resuélvase el ejercicio 4.7.1, referente a sumatorias, mediante una función que aplique el concepto de iteración.

a) **Análisis:** Ya se decía en el análisis del ejercicio 4.7.1 sobre la claridad del planteamiento, la necesidad de una función y que en su cuerpo se requiere una suma, tal y como se describe en la formula, para que al final se obtenga el total solicitado. Sin embargo, como ya no se restringe a aplicar la recursividad, se deberá buscar la solución mediante las estructuras de control repetitivas para hacer las sumas sucesivas hasta alcanzar el total.

b) **Codificación:** El programa nuevamente solicitará el número que marque el fin de la sumatoria, pero la función deberá resolver el resto que, en este caso, será apoyarse con las estructuras de control repetitivas. Aunque puede hacerse con cualquiera, se propone emplear el ciclo **for()**, ya que conocemos de antemano el número de sumas que se ocupan para obtener el total. La función seguirá requiriendo un parámetro del tipo entero para controlar el número de veces que se repetirá el ciclo y, a su vez, devolverá un valor del mismo tipo una vez hechas las sumas. Por este motivo, seguirá siendo indispensable al declarar el tipo de función, que sea del mismo tipo que la variable que almacene el total de la suma. Por último, se imprime la suma total que resuelva el planteamiento solicitado.

c) **Comprobación**: Anteriormente se recomendó la ejecución paso a paso, para este caso es indispensable además activar el depurador, para dar seguimiento al cómo va cambiando parcialmente el valor de la suma. De otra manera, si sólo se ejecuta el programa y se observa el resultado, no podrá apreciar la diferencia entre el concepto iterativo y el recursivo del problema anterior.

Ejercicio 4.7.3: Aplicando el concepto de recursividad o iteración realiza un programa que a través de una función imprima en pantalla los números de 1 a *N*, en donde *N* sea el valor proporcionado por el usuario.

a) Análisis: Esta solución es aún más simple que el ejercicio 4.7.1 y, de hecho, también puede ser resuelto con una estructura de control repetitiva. Sin embargo, con el ánimo de pensar una solución alternativa, se solicita el utilizar nuevamente la recursión que ofrece el lenguaje C. En este caso, nos toca dilucidar la manera de ir imprimiendo de uno en uno hasta el límite fijado por el usuario. Entonces, se coincide en el hecho de que se ocupa una función y que en su cuerpo se requiere una operación o cálculo que, dado el límite superior, lo vaya decrementando de uno en uno, hasta llegar a 1, Finalmente, las autollamadas de la función serán las que realicen el proceso de impresión.

b) Codificación: El programa otra vez necesita solicitar el número que marque el fin de la secuencia numérica, matemáticamente podemos decir el límite superior de la secuencia o serie. La función deberá imprimir dicha secuencia, que en este caso es auto-llamarse hasta mostrar en pantalla todos sus valores. Con esto, la función requerirá de un parámetro del tipo entero y no devolverá un valor, ya que solamente imprimirá el valor recibido. La clave está en el orden en que la función se llamará a sí misma y el orden en que imprimirá el número recibido como parámetro. No olvidar la estructura de control que maneje la condición de terminación de la recursión.

c) **Comprobación**: Reiteramos el uso de la ejecución paso a paso, para terminar de comprender este concepto recursivo que ofrece el lenguaje C. De otra manera, si sólo se ejecuta el programa y se observa el resultado, no podrá apreciar diferencia alguna, con respecto a otras soluciones.

Ejercicio 4. 7.4: Para concluir esta unidad agregamos un *súper ejercicio*, y le damos este nombre porque se va a desmenuzar el programa solicitado en pequeños subproblemas, para que sean resueltos a través de funciones que, al incorporarse a la función principal, darán solución a lo aquí solicitado. Comenzamos diciendo que el objetivo es contar con un programa "Conversor de unidades de medida", el cual deberá ofrecer de inicio la alternativa al usuario para seleccionar si quiere convertir unidades de peso, de longitud o de moneda. En este apartado 4 y el apartado 3, ya se hicieron algunos programas para convertir de dólares a pesos y de kilos a libras. Esos mismos programas, con pequeños ajustes, podrán incluirse en este programa *conversor*. Por lo pronto, delimitemos lo que se pide de inicio y se deja la libertad al usuario para que le incorpore, conforme vaya necesitando, otro tipo de conversiones como pudieran ser de tamaño, en el caso de los bytes, ya que es muy común el necesitar conocer entre diferentes medidas la equivalencia entre kilobytes, megabytes y gigabytes, entre otras. El programa, una vez ofrecidas las alternativas posibles, deberá presentar un submenú con opciones disponibles dentro de esa categoría, por ejemplo, si se selecciona longitud, pudiera brindarse la opción de convertir metros a yardas, centímetros a pulgadas o metros a pies y así, sucesivamente, con el resto de las unidades de medida; pero ya serán especificadas en cada uno de los incisos siguientes:

A. En la función principal main(), o dentro de una función, crear un menú que presente las opciones principales: PESO, LONGITUD, VOLUMEN y MONEDA.

a. Análisis: Al ser un programa más completo requiere ir realizando un análisis parcial por cada una de la partes que la integran y aquí, al ser la parte que

controlara el programa, será necesario colocar el menú con las opciones principales dentro de la función main(), la cual gobernará el resto de las alternativas.

b. **Codificación:** Como se trata únicamente de presentarle al usuario del programa las alternativas posibles de conversión, eso se logra con la estructura de control condicional switch() y de ahí será transferir el control, mediante una función, a las opciones que se ofrezcan por cada unidad de medida.

c. **Comprobación**: Si se ejecuta este programa se sugiere poner de manera provisional un mensaje con la función printf(), para saber que sí está trabajando correctamente el menú.

B. Crear un submenú, por cada medida, de tal manera que el usuario seleccione la conversión que requiera, dadas las siguientes posibilidades: PESO (lb-kg, kg-lb, onzas-gramos y gramos-onzas); LONGITUD (yarda-metro, metro-yarda, pulgada-centímetro y centímetro-pulgada); VOLUMEN (galón-litro, litro-galón, mililitro-onza y onza-mililitro) y MONEDA (este se deja opcionalmente).

a. **Análisis:** Una vez construido el menú principal, el siguiente paso es hacer un submenú por cada unidad de medida, es decir, que si la persona eligió volumen se deberá presentar un submenú, donde la persona pueda seleccionar entre las opciones disponibles, por ejemplo: 1.- Galón-litro, 2.- Litro-galón, 3.- Mililitro-onza, 4.- Onza-mililitro y 0.- Regresar al menú principal, esto para el Volumen, pero se tendrá que hacer lo mismo con cada unidad de medida presentada en el menú principal.

b. **Codificación:** Al igual que el menú principal, el propósito es presentar las alternativas posibles de conversión exclusivamente de esa unidad de medida, cosa que se logra con la estructura de control condicional switch() y, de nuevo, transferir el control mediante llamados a las funciones que realizaran la conversión pertinente. Algo adicional y que no se contempla en el menú es que aquí sí será necesario pedir los datos indispensables para la conversión y, una vez hecho el cálculo, presentar el resultado. El prototipo sugerido para las funciones es el nombre sin parámetros, por ejemplo **kg-a-lb()**, y dentro de ellas establecer todos los elementos antes descritos.

c. **Comprobación**: Para probar cada uno de los submenús del programa se sugiere una vez más el poner de manera provisional mensajes con la función printf(), para saber que sí están trabajando correctamente las elecciones del submenú. Lo anterior mientras se van habilitando una a una las diferentes alternativas de conversión.

C. Al ir en cascada y una vez hecho los menús, se pide iniciar una a una las opciones del submenú. En este caso y siguiendo un orden, realiza con funciones las operaciones necesarias para convertir de lb-kg, para ello es necesario solicitar la cantidad en libras y, ya con la equivalencia en kilos, dar el resultado.

a. **Análisis:** Concluidos el menú principal y el submenú por cada unidad de medida, sólo resta ir incorporando las funciones que permitan la conversión con los datos proporcionados por el usuario. En este caso, toca analizar la función para pasar de libras a kilos y, en ese sentido, de inicio se necesita la cantidad a convertir y la equivalencia de una libra en kilogramos, tarea que se tiene que realizar antes de construir la función.

b. **Codificación:** Dada la similitud entre todas las conversiones, conviene aquí diseñar una función sin que se pidan unidades específicamente, para que si le proporcionamos los datos a convertir y la equivalencia, ésta nos dé el resultado ya convertido. Hacer esto hará más ligero el diseño e implementación de las funciones subsecuentes. El prototipo de la función, será: **convertir(cantidad, equivalencia)** donde la cantidad es la cifra que se quiere convertir y la equivalencia es la relación que hay entre ambos sistemas de medida. Por ejemplo, sabemos que un kilogramo equivale a 2.2 libras, entonces, 10 libras a cuántos kilos equivale será la tarea de la función convertir(). Respecto al cuerpo de la función **lb_a_kg()**, en ella se pondrán las instrucciones para pedir los datos al usuario y para informarle el resultado.

c. **Comprobación**: La solicitud para probar esta función es elegir la opción PESO Y luego la opción lb-a-kg. Los datos a introducir podrán corresponder a la misma equivalencia para que, en este caso, dé 1. Esto es, si decimos 2.2 libras a cuántos kilogramos equivale, el resultado deberá ser uno. De ser correcto el resultado en el programa desarrollado, ya se podrá probar con otros datos.

D. La siguiente función sería hacer lo inverso que, en este caso, es pasar de kilos a libras y solicitar la cantidad a convertir.

a. **Análisis:** En esta segunda función lo que hay que tomar en cuenta es que como un kilo es casi el doble en libras entonces, si se quiere saber a cuanto equivale 10 kg en libras, la operación obligada es la multiplicación. Aparte de investigar la equivalencia correspondiente, es importante recordar que ya se hizo un planteamiento similar en el Ejercicio 4.2.2, por lo que con unas pequeñas adecuaciones esa implementación se puede habilitar aquí.

b. **Codificación:** Si decide incorporar la función realizada en el Ejercicio 4.2.2, sólo recuerde que en el cuerpo de la función **kg_a_lb()** se pondrán las instrucciones para pedir los datos al usuario, hacer la conversión e informar el resultado.

c. **Comprobación**: La solicitud para probar esta función es elegir la opción PESO Y luego la otra opción correspondiente a kg-a-lb. El dato a introducir podrá ser 1 kg y el resultado que arroje la conversión debe ser la misma equivalencia, que en este caso es de 2.2 libras. De ser correcto el resultado, ya se podrá probar el programa desarrollado con datos diferentes.

E. Otra función sería para pasar de gramos a onzas.

a. **Análisis:** El análisis se va ir simplificando debido al patrón que se va presentando al momento de convertir, esto porque forzosamente se requiere solicitar los datos, hacer la conversión una vez que se conoce la equivalencia respectiva e informar el resultado. En este caso sabemos que 1 gramo es igual a 0.035270 onzas.
b. **Codificación:** Respecto a la codificación, ya venimos diciendo que el prototipo de la función es sin parámetros y, más bien, en el cuerpo de la función se escriben las instrucciones que solicitarán, procesarán e informarán el resultado.
c. **Comprobación**: La solicitud para probar esta función es elegir la opción PESO Y luego la otra opción correspondiente a gramo-a-onza. El dato a introducir podrá ser 1 gramo y el resultado que arroje la conversión debe ser la misma equivalencia, que en este caso es de **1 gramo = 0.035270 onzas**. De ser correcto el resultado, ya se podrá probar con datos diferentes.

F. Una función más es hacer lo contrario, esto es, de onzas a gramos.
   a. **Análisis:** El análisis de esta conversión se simplifica debido a que se hizo un planteamiento similar en el Ejercicio 4.2.1, por lo que con unas pequeñas adecuaciones esa implementación puede habilitarse aquí.
   b. **Codificación:** Si decide incorporar la función realizada en el Ejercicio 4.2.1, sólo recuerde que en el cuerpo de la función **onza-a-gr()** se pondrán las instrucciones para pedir los datos al usuario, hacer la conversión e informar el resultado. Un cuidado que sí se tiene que considerar es que, como se van a manejar cantidades muy pequeñas o muy grandes, las declaraciones tienen que ser del tipo adecuado, ya sea entero largo o con decimales, según corresponda.
   c. **Comprobación**: La solicitud para probar esta función es elegir la opción PESO Y luego la otra opción correspondiente a onza-a-gr. El dato a introducir podrá ser 1 onza y el resultado que arroje la conversión debe ser la misma equivalencia que, en este caso, es de **1 onza = 28.349523 gramos**. De ser correcto el resultado, ya se podrá probar con datos diferentes.

G. Respecto a la longitud sería algo similar, es decir, hacer la función para convertir de yardas a metros.
   a. **Análisis:** Después de haber hecho un análisis exhaustivo por cada una de las alternativas de la unidad de medida del PESO, podemos concluir que tanto para las unidades de medida como de volumen, el patrón es el mismo; así que las tareas son repetitivas, es decir, investigar la equivalencia y multiplicar o dividir según corresponda, para encontrar la equivalencia solicitada. En este caso 1 yarda es igual a 0.914411 metros.
   b. **Codificación:** Con relación a la codificación, el patrón también se repite, es decir, una función sin argumentos y dentro de la función colocar todo lo necesario que ya se ha venido diciendo en los ejercicios previos, así como tener cuidado en la declaración de las variables locales.

c. **Comprobación**: Respecto a comprobar, no está demás decir, que se tiene que recorrer tanto el menú como el submenú por las opciones que permitan ir probando cada una de las funciones implementadas. De inicio el dato obligado es escribir el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma, de no ser así, se tendrá que revisar la función correspondiente; de lo contrario, con plena confianza se podrá intentar con cifras diferentes.

H. La siguiente función a habilitar sería la conversión de metros a yardas.

a. **Análisis:** Por lo dicho en el análisis del inciso G, sólo resta decir que la tarea es investigar la equivalencia y determinar qué operación corresponde. Para este inciso la equivalencia es 1 metro = 1.9361 yardas y, poco a poco, en los incisos restantes, tanto de longitud como de volumen, se irán dejando a investigación del lector.

b. **Codificación:** El prototipo de la función es **mt-a-yarda()**. Se pide nuevamente tener cuidado en la declaración de las variables locales.

c. **Comprobación**: Seleccionar las opciones adecuadas para probar la función en turno e introducir como dato el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

I. Para la misma opción de longitud, la otra función es de pulgadas a centímetros.

a. **Análisis:** La equivalencia es que 1 pulgada es igual a 2.54 cm, además, determinar si para convertir se requiere multiplicar o dividir.

b. **Codificación:** El prototipo de la función es **pulgada_a_cm()**. Cabe recordar que en todos los casos donde se requiera dividir, en la codificación del inciso C, se planteó una función que podrá ser utilizada cada vez que se requiera ese tipo de operación, porque en ocasiones lo que se requiere es multiplicar para encontrar el resultado.

c. **Comprobación**: Seleccionar las opciones adecuadas para probar la función en turno e introducir como dato el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

J. La última opción de longitud será la función para pasar de centímetros a pulgadas.

a. **Análisis:** La equivalencia puede ser deducida por el estudiante a partir de la información proporcionada en el análisis del inciso I. Lo que sí habría que contestar es cuándo dividir y cuándo multiplicar. En este sentido, se puede partir del hecho de que si en el sistema inglés una unidad corresponde a más de una unidad del sistema internacional, entonces se tiene que multiplicar. Por ejemplo: 1 pulgada = 2.54 cm, entonces, si quiero saber cuántos centímetros son 10 pulgadas, lo que ocupo es multiplicar 10 x 2.54 pero, si la conversión es al contrario, es decir, ocupo saber cuántas pulgadas son 10 centímetros, necesito dividir. En otras palabras 10 / 2.54 y obtendré el resultado.

b. **Codificación:** El prototipo de la función es **cm_a_pulgada ()**. Cabe recordar que en todos los casos donde se requiera dividir, en la codificación del inciso C se planteó una función que podrá ser utilizada cada vez que se requiera ese

tipo de operación, porque como se vió en el análisis, en ocasiones lo que se requiere es multiplicar para encontrar el resultado.

c. **Comprobación**: Seleccionar las opciones adecuadas para probar la función en turno e introducir como dato el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

K. En el caso del volumen la función necesaria será de galones a litros.

a. **Análisis:** En el análisis exhaustivo del inciso J se define cuándo dividir y cuándo multiplicar. Con relación a la equivalencia se puede decir que un galón equivale a 3.78 litros.

b. **Codificación:** El prototipo de la función es **galon_a_lt ()**. Cabe recordar que en todos los casos donde se requiera dividir, en la codificación del inciso C se planteó una función que podrá ser utilizada cada vez que se requiera ese tipo de operación porque, como se vió en el análisis del inciso J, en ocasiones lo que se requiere es multiplicar para encontrar el resultado.

c. **Comprobación**: Seleccionar las opciones adecuadas para probar la función en turno e introducir como dato el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

L. La antepenúltima opción de conversión del volumen es pasar de litros a galones, construir la función necesaria.

a. **Análisis:** Para la equivalencia de litros a galones se puede deducir la equivalencia a partir de la equivalencia proporcionada en el inciso K, o bien investigarlo por su cuenta.

b. **Codificación:** El prototipo de la función es **lt_a_galon()**. Cabe recordar que en todos los casos donde se requiera dividir, en la codificación del inciso C se planteó una función que podrá ser utilizada cada vez que se requiera ese tipo de operación porque, como se vió en el análisis del inciso J, en ocasiones lo que se requiere es multiplicar para encontrar el resultado.

c. **Comprobación**: Seleccionar las opciones adecuadas para probar la función en turno e introducir como dato el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

M. La penúltima opción del VOLUMEN será pasar cantidades de mililitros a onzas.

a. **Análisis:** La equivalencia de mililitros a onzas se deja a investigación.

b. **Codificación:** El prototipo de la función es **ml_a_onzas()**. Cabe recordar que en todos los casos donde se requiera dividir, en la codificación del inciso C se planteó una función que podrá ser utilizada cada vez que se requiera ese tipo de operación porque, como se vió en el análisis del inciso J, en ocasiones lo que se requiere es multiplicar para encontrar el resultado.

c. **Comprobación**: Seleccionar las opciones adecuadas para probar la función en turno e introducir como dato el 1 (uno). En todos los casos se deberá devolver

como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

N. Para complementar al volumen, la última conversión deberá ser de onzas a mililitros.
   a. **Análisis:** La equivalencia de onzas a mililitros se deja a investigación.
   b. **Codificación:** El prototipo de la función es **onzas_a_ml()**. Cabe recordar que en todos los casos donde se requiera dividir, en la codificación del inciso C se planteó una función que podrá ser utilizada cada vez que se requiera ese tipo de operación porque, como se vió en el análisis del inciso J, en ocasiones lo que se requiere es multiplicar para encontrar el resultado.
   c. **Comprobación**: Seleccionar las opciones adecuadas para probar la función en turno e introducir como dato el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

O. Respecto a la opción de monedas, esto es muy basto, ya que se puede pasar de Euro a pesos, de Pesos a Euros, de Dólares a Pesos y de Pesos a Dólares o de Euros a Dólares y viceversa. En otras palabras, hay muchas combinaciones, así que dejamos al usuario que implemente las que considere necesarias (CABE RECORDAR QUE ESTO ES OPCIONAL).
   a. **Análisis:** Tras hacer el análisis para el PESO, LONGITUD y VOLUMEN; podemos decir que la clave está en contar con las equivalencias, porque lo demás presenta el mismo patrón, es decir, solicitar la cantidad, realizar la conversión y presentar en pantalla el resultado.
   b. **Codificación:** Los prototipos de las funciones serán similares a los construidos en incisos anteriores, es decir, sin argumentos y dentro de su cuerpo las instrucciones para la lectura de datos, efectuar el cálculo para la conversión e imprimir en pantalla la conversión hecha. Cabe recordar que en todos los casos donde se requiera dividir, en la codificación del inciso C se planteó una función que podrá ser utilizada cada vez que se requiera ese tipo de operación porque, como se vió en el análisis del inciso J, en ocasiones lo que se requiere es multiplicar para encontrar el resultado.
   c. **Comprobación**: Finalmente, a través de los menús se podrán seleccionar las opciones adecuadas para probar la función en turno y se sigue insistiendo en introducir como dato el 1 (uno). En todos los casos se deberá devolver como resultado la equivalencia misma y revisar que dé la misma equivalencia para poder probar con valores distintos.

APÉNDICE

Ejercicio 1

1.- V; 2.- F; 3.- V; 4.- V; 5.- F; 6.- V; 7.- F; 8.- F; 9.- F; 10.- V

Ejercicio 2

1.- d**;** 2.- f**;** 3.- g; 4.- j**;** 5.- h; 6.- b; 7.- i; 8.- a; 9.- c; 10.- e

Ejercicio 3

1.- Recursividad. 2.- Parámetro por valor. 3.- Parámetro por referencia. 4.- Parámetros formales. 5.- Automáticas . 6.- Estáticas. 7.- Registro. 8.- Cabecera. 9.- Ventaja de una función. 10.- Iteración.

Ejercicio 4

1.- b); 2.- b); 3.- c); 4.- a); 5.- c); 6.- a); 7.- b); 8.- d); 9.- b); 10.- a)

Ejercicio 5

1.- V; 2.- V; 3.- F; 4.- F; 5.- V; 6.- V; 7.- F; 8.- F; 9.- F; 10.- V

Ejercicio 6

1.- (d); 2.- (f); 3.- (a); 4.- (g); 5.- (c); 6.- (b); 7.- (h); 8.- (e)

Ejercicio 7

Una **función** es un bloque de programa que realiza una operación específica, a su declaración la podemos llamar **prototipo**. Utilizamos una función llamada **scanf** para que el usuario introduzca datos y, **printf**, para que muestre datos en pantalla. Con la declaración #**include** abrimos los **archivos** necesarios para poder llamar a las funciones de **biblioteca**. Uno de estos archivos es **stdio**.h.

Las **variables** **globales** se declaran cuando se encuentran fuera de la función *main()*, y, las variables **locales**, se declaran dentro de ella o de las funciones creadas por el programador. Existen, además, las variables **estáticas**, que se mantienen en **memoria** durante la ejecución del programa.

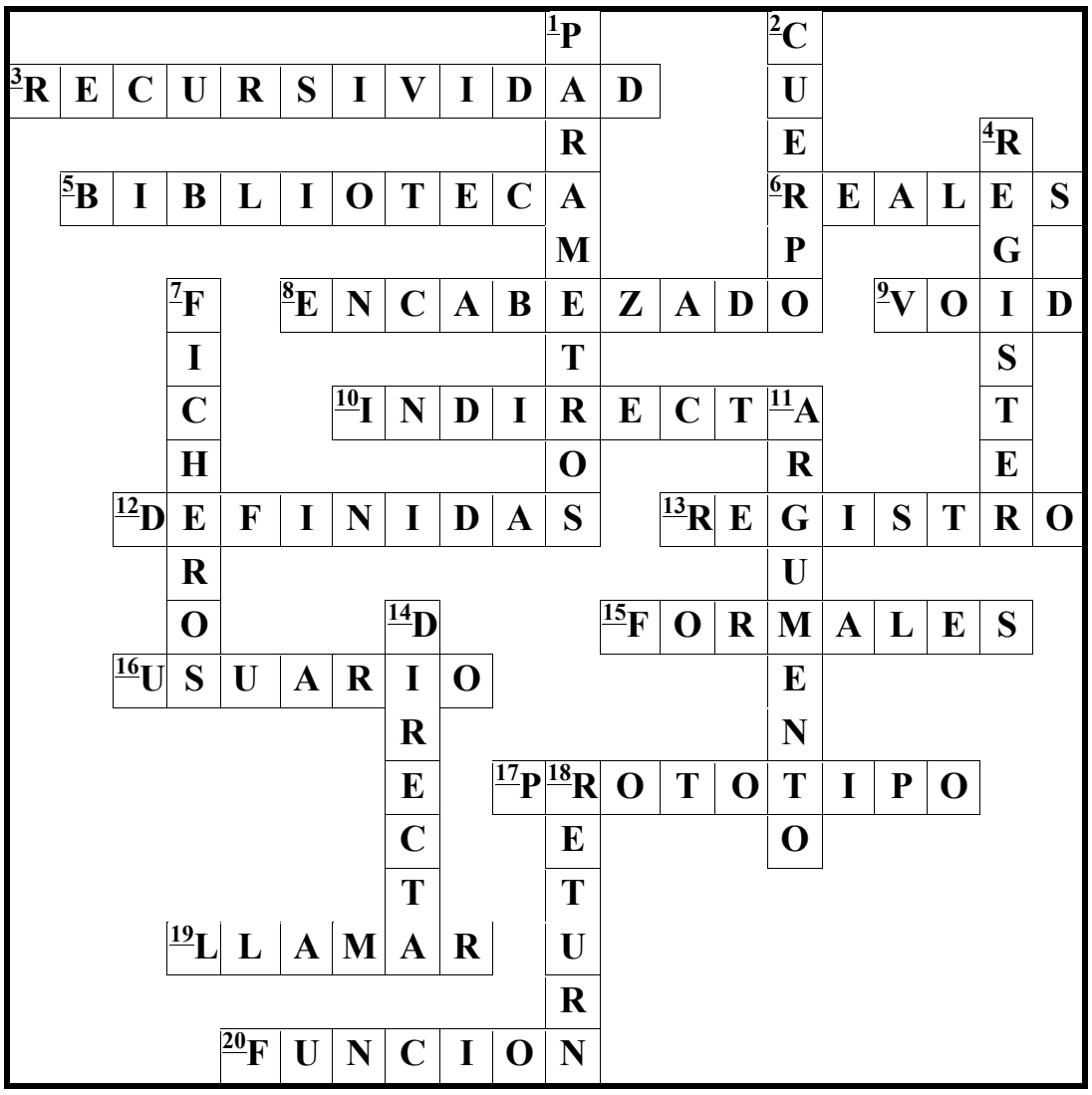
En una función se utilizan los **apuntadores** para almacenar la dirección de memoria donde se encuentra un dato. Además, en el **encabezado** de la función introducimos la lista

de **parámetros** con los datos de entrada hacia la función. A los parámetros también los llamamos **argumentos**. Dicha función devuelve un dato con la palabra reservada **return** y, si no devuelve un valor, se utiliza la palabra reservada **void**. Cabe mencionar que las funciones son **recursivas**, cuando se llaman a sí mismas.

Sopa de letras

						P	R	O	T	O	T	I	P	O					
F	I	C	H	E	R	O											R		
	U								A				P	U	N	T	E	R	O
		N						C									C		
			C				E	S	T	A	T	I	C	A			U		
I				I		T								A			R		
	N		G	L	O	B	A	L			P		I				S		
		C		I		N						R					I		
			L				O				O		I				V		
		B		U		I				M				N			I		
	I				D			R	E	T	U	R	N		T		D		
B				T		E		M								F	A		
			S								V						D		
			P	A	R	A	M	E	T	R	O	A							
											I		R				F		
											D			I		N			
A	R	G	U	M	E	N	T	O				L	O	C	A	L	E	S	
														C		B			
	E	N	C	A	B	E	Z	A	D	O			S				L		
																		E	

Crucigrama


1P
2C
3RECURSIVIDAD
U
R
E
4R
5BIBLIOTECA
6REALES
M
P
G
7F
8ENCABEZADO
9VOID
I
T
S
C
10INDIRECTA
11A
T
H
O
R
E
12DEFINIDAS
13REGISTRO
R
U
O
14D
15FORMALES
16USUARIO
E
R
N
E
17P
18ROTOTIPO
C
E
O
T
T
19LLAMAR
U
R
20FUNCION

Listado de Ejercicios RESUELTOS del apartado 4

Nota: Las soluciones a los ejercicios fueron probadas en las versiones para Windows de los compiladores Turbo C++ versión 3.0 y Dev-C++ versión 4.9.9.2.

Listado 4.2.1:

```
/* ----------------------------------------------------------------
Programa: Ejercicio4-2-1.c
Objetivo: Uso de funciones para ejemplificar la modularización.
----------------------------------------------------------------*/
#include <stdio.h>

float onza, gramo = 28.3495;
float pesoC;

void pedir();      /* Declaración de los prototipos de las funciones */
void convertir();
void imprimir();

void main()
{
      clrscr();   /* Función utilizada para limpiar la pantalla */
      pedir();
      convertir();
      imprimir();

      getch();    /* Función que espera un carácter del teclado y
                     permite ver resultados */
}

/* ----------------------------------------------------------------
Función: pedir()
Objetivo: Solicita una cantidad a convertir.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------*/
void pedir()
{
      printf("\n Dar la cantidad de onzas a convertir: ");
      scanf("%f", &onza);
}

/* ----------------------------------------------------------------
Función: convertir()
Objetivo: Convierte una cantidad de onzas a gramos.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------*/
void convertir()
{
      pesoC = onza * gramo;

}
```

```
/* ----------------------------------------------------------------------
Función: imprimir()
Objetivo: Imprime la cantidad convertida.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------------*/
void imprimir()
{
      printf("\n %8.2f onzas en gramos son %8.2f ", onza, pesoC);

}
```

Listado 4.3.1:

```
/* ----------------------------------------------------------------
Programa: Ejercicio4-3-1.c
Objetivo: Uso de funciones con paso de parámetros.
----------------------------------------------------------------*/
#include <stdio.h>

int  diferenciaE, num_enamorado;
void compara(int, int);

void main()
{
      int edad1, edad2;

      clrscr();
      printf("\n Cual es la edad del enamorado 1? ");
      scanf("%d", &edad1);
      printf("\n Cual es la edad del enamorado 2? ");
      scanf("%d", &edad2);

      compara(edad1, edad2);

      if ((diferenciaE > 0) && (diferenciaE < 6))
      {
        printf("\n El enamorado %d es más joven por %d años",
            num_enamorado, diferenciaE);
        printf("\n Sin embargo existe COMPATIBILIDAD por edad ");

      }

      if (diferenciaE > 5)
      {
         printf("\n El enamorado %d es más joven por %d años",
            num_enamorado, diferenciaE);
         printf("\n por lo tanto NO existe COMPATIBILIDAD por edad ");
      }

      if (diferenciaE == 0)
        printf("\n Existe 100'%' de COMPATIBILIDAD por edad");

      getch();

}

/* ----------------------------------------------------------------
Función: compara()
Objetivo: Comparar dos números enteros.
Parámetros: e1 y e2 que son los números a comparar.
Salida: Ninguna.
----------------------------------------------------------------*/
void compara(int e1, int e2)
{
      if (e1 < e2)
      {
        num_enamorado = 1;
```

```
        diferenciaE = e2 - e1;
      }
      else
      {
        num_enamorado = 2;
        diferenciaE = e1 - e2;
      }
}
```

Listado 4.4.1

```
/* ----------------------------------------------------------------
Programa: Ejercicio4-4-1.c
Objetivo: Creación y uso de funciones con devolución de valores.
----------------------------------------------------------------*/
#include <stdio.h>

int suma(int, int);
float dividir(int, int);

void main()
{
      int cuenta;
      char opcion, resp;
      int num1, num2;
      float total;

      do
      {
            clrscr();

            printf("\n Dame el primer número: ");
            scanf("%d", &num1);
            printf("\n Dame el segundo número: ");
            scanf("%d", &num2);

            printf("\n BIENVENIDO A LA CALCULADORA ELECTRONICA \n");
            printf("(+) Sumar                    (/) Dividir \n");
            printf("(-) Restar                   (x) Multiplicar \n");
            printf("(=) SALIR DE LA CALCULADORA \n ");

            printf("\n Digite el operando: ");
            opcion = getch();

            switch(opcion)
            {
              case '+':
                  total = suma(num1, num2);
                  break;

              case '/':
                  total = dividir(num1, num2);
                  break;

              default:
                  exit(0);
            }

            printf("\n El resultado es %8.2f", total);

            printf("\n Desea realizar otro calculo (s\n):");
            resp = getch();

      } while (resp == 's' || resp == 'S');
}
```

```
/* ----------------------------------------------------------------
Función: sumar()
Objetivo: Sumar dos números enteros.
Parámetros: n1 y n2 que son los números enteros a sumar.
Salida: El resultado de la suma.
----------------------------------------------------------------*/
int suma(int n1, int n2)
{
   return (n1 + n2);

}

/* ----------------------------------------------------------------
Función: dividir()
Objetivo: Dividir dos números enteros, evitando la división entre cero.
Parámetros: n1 y n2 que son los números enteros a dividir.
Salida: El resultado de la división.
----------------------------------------------------------------*/
float dividir(int n1, int n2)
{
    float total;

    if (n2 == 0)
    {
      printf("No se puede dividir entre 0 (cero)");
      total = 0;

    }
    else
    {
      /* n1 / n2 es una división entera, que truncaría los decimales. Por
         eso se convierte primero uno de los valores al tipo float */
      total = n1;
      total = total / n2;

    }

    return (total);

}
```

Listado 4.4.3:

```
/* ------------------------------------------------------------------
Programa: Ejercicio4-4-3.c
Objetivo: Uso de funciones que devuelven valores.
------------------------------------------------------------------*/
#include <stdio.h>

#define PRECIO_MSM 3.85

long calculaIng(long);
long utilidad(long, long);

void main()
{
      long premio, totalMSM, ingresoB;
      float ganancia;

      clrscr();
      printf("\n Cual es monto del premio? ");
      scanf("%ld", &premio);

      printf("\n Cual es el total de mensajes? ");
      scanf("%ld", &totalMSM);

      ingresoB = calculaIng(totalMSM);
      ganancia = utilidad(ingresoB, premio);

      printf("\n La utilidad por ese concurso es de $ %8.2f", ganancia);
      getch();
}

/* ------------------------------------------------------------------
Función: calculaIng()
Objetivo: Calcular el ingreso total, multiplicando mensajes recibidos con
el precio por mensaje.
Parámetros: El número de mensajes que la televisora recibió.
Salida: El ingreso de la televisora dados los mensajes enviados.
------------------------------------------------------------------*/
long calculaIng(long mensajes)
{
  return (PRECIO_MSM * mensajes);

}

/* ------------------------------------------------------------------
Función: utilidad()
Objetivo: Calcular la utilidad obtenida por una televisora, generada
      por la recepción de mensajes de texto en un concurso.
Parámetros: El ingreso obtenido por la recepción de mensajes de texto
      y el valor del premio otorgado en un concurso de TV.
Salida: La utilidad neta de la televisora.
------------------------------------------------------------------*/
long utilidad(long ing, long valorP)
{
      return (ing - valorP);
}
```

Listado 4.6.1:

```
/* ----------------------------------------------------------------
Programa: Ejercicio4-6-1.c
Objetivo: Uso de funciones que devuelven valores.
----------------------------------------------------------------*/
#include <stdio.h>

#define CUADRAS 30;

int calculaTR(int, int);
float tiempo(int);

void main()
{
      int cuadra1 = 45, sale = 45, debeLlegar = 60;
      int tiempoD, tiempoR;
      float diferenciaT, tiempoRmin;

      clrscr();

      tiempoD = calculaTR(sale, debeLlegar);
      tiempoR = cuadra1 * CUADRAS;
      tiempoRmin = tiempo(tiempoR);
      diferenciaT = tiempoD - tiempoRmin;

      if (diferenciaT > 0)
      {
          printf("\n FELICIDADES SIGUE A ESE PASO");

      }
      else
      {
          printf("\n A ese paso nunca vas a llegar a tiempo");
          printf("\n ... Necesitas levantarte a las 6:%d A. M.",
            (int) (debeLlegar - tiempoRmin));

      }

    getch();

}

/* ----------------------------------------------------------------
Función: calculaTR()
Objetivo: Calcular el tiempo máximo permitido para recorrer cierto
      número de cuadras.
Parámetros: El tiempo de inicio del recorrido y el tiempo límite de
      llegada puntual (ambos en minutos).
Salida: El tiempo máximo calculado.
----------------------------------------------------------------*/
int calculaTR(int inicio, int limite)
{
      return (limite - inicio);

}
```

```
/* ------------------------------------------------------------------
Función: tiempo()
Objetivo: Convertir una cantidad de segundos en minutos.
Parámetros: El número de segundos a convertir.
Salida: La equivalencia en minutos del tiempo proporcionado.
------------------------------------------------------------------*/
float tiempo(int t)
{
      /* 60.0 para convertir el resultado a float*/
      return (t / 60.0);

}
```

Listado 4.7.1:

```
/* ----------------------------------------------------------------
Programa: Ejercicio4-7-1.c
Objetivo: Uso de funciones recursivas.
----------------------------------------------------------------*/
#include <stdio.h>

long sumatoria(int);

void main()
{
	int num;
	long sum;

	printf("\n Cual es límite de la sumatoria? ");
	scanf("%d", &num);

	sum = sumatoria(num);

	printf("\n La sumatoria de %2d es %2d", num, sum);

	getch();

}

/* ----------------------------------------------------------------
Función: sumatoria()
Objetivo: Realizar de manera recursiva una serie de sumas sucesivas
	desde la unidad hasta un cierto límite.
Parámetros: El limite de la sumatoria.
Salida: La sumatoria de todos los valores menores al límite dado.
----------------------------------------------------------------*/
long sumatoria(int n)
{
	if (n > 0)
		return (n + sumatoria(n - 1));
	else
		return (0);

}
```

Listado 4.7.4.a

```
/* ----------------------------------------------------------------------
Programa: Ejercicio4-7-4a.c
Objetivo: Presentar un menú con las opciones principales del conversor.
----------------------------------------------------------------------*/
#include <stdio.h>

void peso();
void longitud();
void volumen();
void moneda();
float convertir(float, float);
void lb_a_kg();
void gr_a_onza();
void yarda_a_mt();
void pulgada_a_cm();
void galon_a_lt();
void ml_a_onza();

void main()
{
    char opcion;

    do
    {
      clrscr();
      printf("\n PROGRAMA CONVERSOR DE UNIDADES DE MEDIDA \n");
      printf("(1) PESO \n");
      printf("(2) LONGITUD \n");
      printf("(3) VOLUMEN \n");
      printf("(4) MONEDA \n");
      printf("(0) SALIR DEL CONVERTIDOR\n");

      printf("\n Oprima el número acorde a su elección: ");
      opcion = getch();

      switch(opcion)
       {
             case '1':
                    peso();
                    break;
             case '2':
                    longitud();
                    break;
             case '3':
                    volumen();
                    break;
             case '4':
                    moneda();
                    break;
             default:
                    exit(0);
                    break;
       }
    } while (opcion != '0');
}
```

Listado 4.7.4c.

```
/*-------------------------------------------------------------------
Función: lb-a-kg()
Objetivo: Convertir una cantidad de libras a kilogramos.
Parámetros: Ninguno.
Salida: Ninguna.
-------------------------------------------------------------------*/
void lb_a_kg()
{
    float cantxconv, cantConv, equivale = 2.2;

    printf("\n Dar la cantidad de Libras a convertir: ");
    scanf("%f", &cantxconv);

    cantConv = convertir(cantxconv, equivale);

    printf("\n %10.2f libras en kilogramos son %10.4f ", cantxconv,
    cantConv);
    getch();
}

/* -------------------------------------------------------------------
Función: convertir()
Objetivo: Realizar el cálculo que permita la conversión entre medidas.
Parámetros: La cantidad por convertir y la equivalencia entre los
      diferentes sistemas de medición.
Salida: La cantidad ya convertida.
-------------------------------------------------------------------*/
float convertir(float c, float eq)
{
    return (c / eq);

}
```

Listado 4.7.4e.

```
/* ----------------------------------------------------------------------
Función: gr-a-onza()
Objetivo: Convertir una cantidad de gramos a onzas.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------------*/
void gr_a_onza()
{
     float cantxconv, cantConv, equivale = 0.035270;

     printf("\n Dar la cantidad de gramos a convertir: ");
     scanf("%f", &cantxconv);

     cantConv = cantxconv * equivale;

     printf("\n %10.2f gramos en onzas son %12.6f ", cantxconv,
     cantConv);
     getch();

}
```

Listado 4.7.4g.

```
/* ----------------------------------------------------------------------
Función: yarda-a-mt()
Objetivo: Convertir una cantidad de yardas a metros.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------------*/
void yarda_a_mt()
{
     float cantxconv, cantConv, equivale = 0.914403;

     printf("\n Dar la cantidad de yardas a convertir: ");
     scanf("%f", &cantxconv);

     cantConv = cantxconv * equivale;

     printf("\n %10.2f yardas en metros son %12.6f ", cantxconv,
     cantConv);
     getch();

}
```

Listado 4.7.4i.

```
/* ----------------------------------------------------------------
Función: pulgada-a-cm()
Objetivo: Convertir una cantidad de pulgadas a centímetros.
Parámetros: Ninguno.
Salida: Ninguna.
-----------------------------------------------------------------------*/
void pulgada_a_cm()
{
     float cantxconv, cantConv, equivale = 2.54;

     printf("\n Dar la cantidad de pulgadas a convertir: ");
     scanf("%f", &cantxconv);

     cantConv = cantxconv * equivale;

     printf("\n %10.2f pulgadas en centímetros son %10.2f ", cantxconv,
     cantConv);
     getch();

}
```

Listado 4.7.4k.

```
/* ----------------------------------------------------------------
Función: galon-a-lt()
Objetivo: Convertir una cantidad de galones a litros.
Parámetros: Ninguno.
Salida: Ninguna.
-----------------------------------------------------------------------*/
void galon_a_lt()
{
    float cantxconv, cantConv, equivale = 3.78;

    printf("\n Dar la cantidad de galones a convertir: ");
    scanf("%f", &cantxconv);

    cantConv = cantxconv * equivale;

    printf("\n %10.2f galones en litros son %10.2f ", cantxconv,
    cantConv);
    getch();

}
```

Listado 4.7.4m.

```
/* ----------------------------------------------------------------
Función: ml_a_onza()
Objetivo: Convertir una cantidad de mililitros a onzas.
Parámetros: Ninguno.
Salida: Ninguna.
-----------------------------------------------------------------*/
void ml_a_onza()
{
    float cantxconv, cantConv, equivale = 30;

    printf("\n Dar la cantidad de mililitros a convertir: ");
    scanf("%f", &cantxconv);

    /* convertir() es una función definida en el inciso C */
    cantConv = convertir(cantxconv, equivale);

    printf("\n %10.2f mililitros en onzas son %10.4f ", cantxconv,
       cantConv);
    getch();

}
```

Quinta parte

Arreglos

5. Arreglos

5.1. Arreglos de una dimensión o vectores

Ejercicio 1

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Cero	Homogéneos	Vectores	Rango
Estructurados	Memoria	Elemento	Arreglos
Búsqueda	Ordenamiento		

Enunciados:

1.- Los arreglos son datos llamados compuestos o _____ que permiten agrupar un conjunto de datos simples o compuestos.

2.- El tamaño o _____ es el número de elementos del vector.

3.- Los _____ son arreglos que permiten almacenar datos homogéneos, de una dimensión.

4.- Se le denomina _____ a cada uno de los datos almacenados en un vector.

5.- Los datos_____ son datos del mismo tipo.

6.- Los _____ son un conjunto de datos del mismo tipo que se relacionan con un nombre en común, pueden ser unidimensionales o multidimensionales.

7.- Una característica de los arreglos es que los datos están en posiciones contiguas dentro de la _____.

8.- El valor _____ es el que se utiliza como índice para el primer elemento de un arreglo.

9.- Una de las operaciones características de los arreglos es el _____ de sus elementos.

10.- Otra de las operaciones características de los arreglos es la _____ de sus elementos, utilizando un valor clave o de referencia.

Ejercicio 2

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Un vector es un arreglo de una dimensión. ( )
2. Los datos de un vector pueden ser de tipos diferentes. ( )
3. Los índices de un vector inician en 0. ( )
4. El número entre [ ] indica la posición de un dato en un vector. ( )
5. Es opcional declarar el tipo de datos del vector. ( )
6. En la declaración de un vector, el número entre [ ] indica el número de datos con los que cuenta el vector. ( )
7. Los elementos de un vector se acomodan en lugares dispersos de memoria. ( )
8. Es opcional inicializar un vector en todos los casos. ( )
9. Es posible almacenar una cadena de caracteres en un vector. ( )
10. Un arreglo es una secuencia de datos de tipos diferentes. ( )

Ejercicio 3

Utilizando las expresiones del lado derecho, escriba la sintaxis correcta de cada una de las instrucciones siguientes:

Declaración de un arreglo tipo vector

_____ _____ [_____] ;	Rango Tipo de dato Nombre del arreglo

Consulta de un arreglo tipo vector

_____ [_____] ;	Nombre del arreglo Índice

Declaración de un arreglo bidimensional

_____ _____ [_____] [_____];	Índice de filas Índice de columnas Nombre del arreglo Tipo de dato

5.2. Arreglos de dos dimensiones o matrices

Ejercicio 4

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Carácter nulo (**'\0'**).	( ) 1.- Está entre los corchetes, indicando el tamaño o rango del arreglo.
b) La dirección de memoria donde inicia el arreglo.	( ) 2.- Es el arreglo tipo carácter.
c) Filas.	( ) 3.- En un arreglo de dos subíndices, a lo que el segundo de ellos hace referencia.
d) Cadena de caracteres.	( ) 4.- Se utiliza cuando hacemos referencia a un elemento en particular del arreglo.
e) Columnas.	( ) 5.- El nombre del arreglo es un apuntador constante a…
f) Matrices.	( ) 6.- En un arreglo de dos subíndices, a lo que el primero de ellos hace referencia.
g) Multidimensionales.	( ) 7.- También son llamados arreglos de dos dimensiones.
h) Subíndice del arreglo.	( ) 8.- Son los arreglos de más de una dimensión.
i) Indización de arreglos.	( ) 9.- Carácter que marca el fin de una cadena.

Ejercicio 5

Utilizando las expresiones del lado derecho, escriba la sintaxis correcta de cada una de las instrucciones siguientes:

Consulta de un arreglo tipo bidimensional

_____ [_____] [_____];	Índice de columna Nombre del arreglo Índice de fila

Consulta de un elemento del arreglo

_____ [_____];	Nombre del arreglo Índice del elemento

Declaración de cadena de caracteres con sus elementos

_____ _____[_____]= “ _____ ”;	Nombre del arreglo char cadena 7

Declaración con asignación de valores de arreglo multidimensional

_____ _____ [_____] [_____] = { {___ , ___ }, { ___ , ___ }};	rango de las columnas fila0col0 rango de las filas fila1col1 Tipo de dato fila0col1 Nombre del arreglo fila1col0

5.3. Arreglos multidimensionales

Ejercicio 6

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Un arreglo multidimensional solo puede ser de dos dimensiones. ( )
2. Cada uno de los índices entre [ ] indica una dimensión del arreglo. ( )
3. Por medio de ciclos podemos introducir valores en los elementos de los arreglos. ( )
4. Un arreglo multidimensional nunca debe ser inicializado. ( )
5. Los arreglos no pueden ser utilizados como parámetros en una función. ( )
6. La ordenación de datos es una operación que se realiza con arreglos. ( )
7. El código de búsqueda de elementos en un vector, no puede ser generalizado a arreglos multidimensionales. ( )
8. Es necesario declarar un arreglo multidimensional. ( )
9. Es imposible almacenar en un arreglo multidimensional varias cadenas de caracteres. ( )
10. Otra manera de denominar a un arreglo es con la palabra *array*. ( )

Ejercicio 7

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Matriz	Bidimensional	Vector	Rango
Índices	Cadenas	Arrays	Tamaño
Parámetros	Memoria	Elementos	Arreglos
Búsqueda	Ordenamiento	Inicializar	Declarar
Función	Tipo	Nulo	Filas

Enunciados:

Los _____ o _____, son conjuntos de datos del mismo _____. Estos se deben _____ para indicar el tipo de datos que contendrán, e _____ para indicar el _____ o _____ del arreglo. Podemos tener un arreglo de una dimensión llamado _____ o de dos dimensiones llamado _____ o arreglo _____. Independientemente del tipo de arreglo, estos ocupan un bloque bien definido de _____.

Los _____, en un arreglo de dos dimensiones, indican el número de _____ y de columnas que lo conforman, estos son colocados entre [ ]. Podemos utilizar a los arreglos como _____ de una _____, siendo cada uno de los _____ del arreglo un dato de entrada a dicha función.

Podemos tener arreglos o _____ de caracteres y, para indicar su final, empleamos el carácter _____ (‘\0’).

Es importante recordar que los arreglos pueden ser utilizados para realizar operaciones de _____, ya sea creciente o decreciente, así como de _____ de elementos.

Sopa de letras

Encuentre las palabras siguientes en esta sopa de letras: Vector, índice, ordenamiento, rango, cadena, inicializar, fila, búsqueda, nulo, arrays, tipo, bidimensional, arreglo, memoria, función, declarar, matrices, elementos, parámetros y tamaño.

G	H	F	D	G	J	F	U	N	C	I	O	N	J	S	C	D	P	E	R
T	J	G	B	F	H	J	K	L	N	T	Y	J	H	D	E	I	A	R	F
R	U	I	N	I	C	I	A	L	I	Z	A	R	T	F	H	F	R	T	D
F	I	E	R	T	D	F	D	F	G	U	O	M	F	H	Y	H	A	Y	R
D	O	P	J	L	F	I	L	A	G	H	W	E	A	E	R	U	M	H	S
N	U	L	O	H	R	E	M	T	Y	U	I	O	P	Ñ	T	Y	E	J	D
T	H	K	J	G	A	S	D	E	D	G	V	J	D	K	O	T	T	K	O
R	G	R	R	T	D	S	R	D	N	R	T	E	J	K	L	R	R	L	R
E	F	A	R	R	E	G	L	O	S	S	C	F	C	H	J	E	O	I	D
W	R	N	T	Y	U	D	G	U	D	L	I	H	K	T	R	S	S	U	E
S	T	G	F	H	Q	C	F	T	A	E	J	O	U	T	O	D	T	G	N
D	Y	O	D	N	S	F	G	R	E	J	H	F	N	R	T	R	G	Y	A
F	U	I	R	G	U	S	A	R	R	A	Y	S	E	A	W	Y	H	R	M
G	I	W	N	F	B	R	S	Q	A	S	D	F	G	H	L	T	R	T	I
C	O	O	Q	D	S	F	G	Q	E	L	E	M	E	N	T	O	S	E	E
N	A	P	W	T	I	P	O	W	P	L	M	N	B	H	U	Y	J	S	N
S	D	D	S	Ñ	H	C	V	E	S	D	M	E	M	O	R	I	A	A	T
D	F	A	E	L	J	H	E	W	F	G	H	U	Y	R	E	W	R	H	O
F	G	H	D	N	K	R	E	D	M	A	T	R	I	C	E	S	E	R	E
A	S	D	F	G	A	H	J	K	L	Ñ	O	I	U	Y	T	R	E	Y	R

Crucigrama

HORIZONTALES

1.- Nombre de tipo de variable `char`.

5.- Operación realizada para introducir los elementos en un arreglo.

6.- En un arreglo matriz, es el primer subíndice al que se hace referencia.

7.- Arreglo con dimensiones superiores a las de una matriz.

8.- Nombre que se le da a un dato almacenado en un vector.

11.- Arreglo de tipo carácter.

12.- Una de las operaciones que se pueden realizar sobre los elementos de un arreglo, en forma creciente o decreciente.

13.- Tamaño asignado a un arreglo.

14.- Es posible utilizar un arreglo como _____ de una función.

16.- En un arreglo matriz, es el segundo subíndice al que se hace referencia.

17.- Conjunto de datos del mismo tipo que se relaciona con un nombre en común.

18.- Un _____ permite accesar un elemento específico de un arreglo.

VERTICALES

2.- Entre _____ se indica el número de elementos de un vector.

3.- Arreglo llamado también matriz.

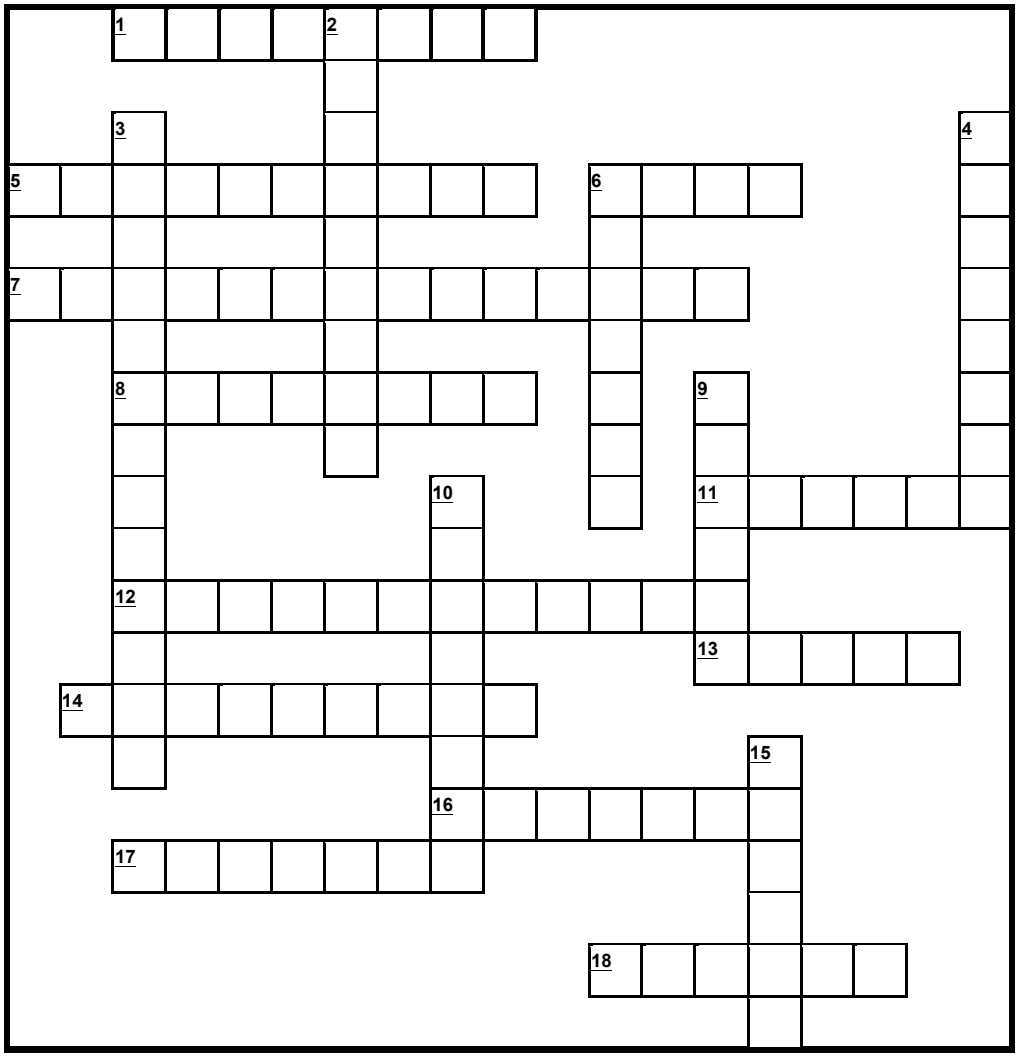
4.- Operación que se realiza con los arreglos para encontrar uno de sus elementos, utilizando un valor clave o de referencia.

6.- En una _____ podemos colocar como parámetros los elementos de un arreglo.

9.- Arreglo que permite almacenar datos homogéneos, de una dimensión.

10.- La matriz puede contener elementos de tipo _____ y, con ellos, realizar operaciones matemáticas.

15.- A un arreglo de dos dimensiones también se le llama _____.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

5.4. Ejercicios prácticos

Para simplificar la revisión de este documento, se aclara que los ejercicios están alternados, es decir, en el apéndice de este apartado se presenta la solución de un ejercicio y se propone que el siguiente sea resuelto por el lector.

El significado de la nomenclatura utilizada, por ejemplo "Ejercicio 2.4.3", tiene la explicación siguiente: el primer número corresponde al apartado; el segundo corresponde al tema específico; y, el tercero, es el número de ejercicio, es decir, apartado 2, tema 4 y ejercicio o programa número 3.

Por otra parte, hay dos tipos de prácticas, las prácticas *normales* y las de *paso a paso*. Para las prácticas normales se recomienda utilizar los ejercicios propuestos para que sean resueltos por el lector y, para las prácticas paso a paso, guiarse con los ejercicios resueltos, solo se debe seguir el procedimiento siguiente:

1. Encender el equipo de cómputo.
2. Abrir sesión del ambiente gráfico Windows.
3. Ejecutar la aplicación que se tenga instalada para editar los programas escritos en lenguaje C.
4. Transcribir el programa en cuestión.
5. Guardar el archivo que contiene el programa transcrito.
6. Compilar el programa.
7. Iniciar un proceso de depuración del programa (corrección de errores) hasta cerciorarse de que el programa cumpla con el objetivo establecido.
8. Imprimir el resultado de la ejecución o ejecuciones del programa (paso opcional).

Ejercicio 5.2.1: Elabora un pequeño programa que almacene en arreglos unidimensionales de manera predefinida, las iniciales de los días de la semana y el número de cada mes del año. Además, realiza la declaración de un arreglo estático con 8 elementos del tipo numérico, a fin de representar la jornada laboral de un día hábil, que son 8 horas. Finalmente, imprime el sexto mes del año, 4 horas de trabajo y el tercer día de la semana.

a) **Análisis:** Revisando el planteamiento, vemos que el objetivo consiste en declarar arreglos de diferentes tipos, es decir, en unos se almacenarán números y en otros texto. Una cosa más que se pretende es el introducir valores a los arreglos al momento de su declaración. En esta ocasión no se solicita ningún dato, únicamente hay que imprimir ciertos elementos del arreglo, tal y como se señala al final del párrafo.

b) **Codificación:** Para elaborar el programa se sugiere declarar de manera global dos arreglos, uno del tipo carácter para almacenar los días de la semana y uno numérico para los meses del año. El tercer arreglo se puede declarar del tipo entero, pero estático como se solicita. Cabe recordar que para inicializar un arreglo se usa el signo igual y después se colocan los valores que serán incluidos en el arreglo, encerrados entre llaves y separados por comas. Para imprimir los valores dentro del arreglo se usa el nombre de este y, dentro de los corchetes, un

número que nos da acceso al arreglo. A este número se le conoce como índice, con el formato siguiente: NombreDelArreglo[índice].

c) **Comprobación**: Respecto a la comprobación, el único resultado que veremos en pantalla será un: 6, un 4 y una M; en virtud de que no se pide información, ni se realiza algún tipo de cálculo.

Ejercicio 5.2.2: Elabora un pequeño programa que almacene en arreglos unidimensionales de manera predefinida, el total de semestres que se cursan en el CONALEP, en otro los años que se dura estudiando y, en otro más, las iniciales de los meses del año que dura el ciclo escolar. Además, se debe mostrar información del semestre en el que se cursa la materia *Programación Básica* y en qué año comienza el bachillerato. Finalmente la inicial del mes de cuándo comienzan las clases.

a) **Análisis:** Nuevamente el objetivo consiste en declarar arreglos de diferentes tipos, es decir, en unos se almacenarán números y en otros texto. Además, se inicializan dichos arreglos con ciertos valores. De acuerdo a las peticiones nos damos cuenta que en dos arreglos se van a guardar números y en un tercer arreglo texto, para finalmente imprimir ciertos elementos del arreglo, tal y como se señala al final del párrafo.

b) **Codificación:** La recomendación para declarar los arreglos es que sea uno del tipo carácter para almacenar las iniciales de los meses y en el primer arreglo de tipo numérico cada uno de los semestres. El tercer arreglo se puede declarar también del tipo entero y almacenar ahí los 3 años que dura la carrera. Respecto a lo que se pide imprimir en pantalla es bueno recordar que nos podemos apoyar con *printf* y, dentro de los paréntesis, opcionalmente poner un texto. Si no se desea emitir mensaje alguno, simplemente se colocará el nombre del arreglo con el elemento requerido, al cual se quiere tener acceso. Por ejemplo: mes[7], para ver en pantalla la letra A de agosto, el mes 8 del año.

c) **Comprobación**: Para comprobar si el programa funciona adecuadamente, los resultados que veremos en pantalla son: 3, 2 y una A. De ser otros los datos, verificar el índice que se colocó dentro de los corchetes y recuerde que los arreglos inician con el elemento 0 (cero), este almacena el primer valor.

Ejercicio 5.2.3: Elabora un pequeño programa que almacene en un arreglo 10 valores del tipo entero, solicitados al usuario, y que los imprima en pantalla, con la finalidad de confirmar si son correctos y, si no, pedirlos nuevamente.

a) **Análisis:** Las peticiones son muy concretas, sin embargo las tareas son varias y son interesantes. Por ejemplo, tenemos que declarar el arreglo de tal forma que pueda guardar 10 valores numéricos. Después, se le tiene que pedir al usuario 10 valores que serán almacenados dentro del arreglo y, finalmente, mostrarle la lista de valores para comprobar que fueron introducidos correctamente.

b) **Codificación:** La propuesta en la declaración del arreglo es el tipo entero, para almacenar los 10 elementos que serán solicitados al usuario. Para requerir los 10 valores se podría hacer uno a la vez, pero emplearía alrededor de 10 instrucciones printf() y otras 10 de scanf(), por eso es que nos podemos ayudar de la instrucción for(), ya que permite repetir las veces necesarias un grupo de instrucciones. En este caso particular las instrucciones consisten en solicitar 10 datos. Por último, para mostrar el contenido completo del arreglo, también nos ayuda el ciclo for(), ya que los elementos del arreglo son consecutivos y podemos ir avanzando con el ciclo de uno en uno hasta recorrer todo el arreglo. Como se va a preguntar al usuario si los datos son correctos y, en su caso, repetir la petición de información, el ciclo do-while() resulta idóneo para auxiliarnos con esta parte.

c) **Comprobación**: El usuario introducirá 10 valores, mismos que se imprimirán a continuación en pantalla en el mismo orden en el que fueron capturados. Luego, el usuario deberá confirmar si los datos introducidos son correctos o si se desea volver a escribirlos. Se volverán a pedir una vez más otros 10 datos, que reemplazarán a todos los anteriores, si la respuesta del usuario es en dicho sentido.

Ejercicio 5.2.4. Elabora un pequeño programa que almacene en un arreglo 10 vocales, en minúsculas y mayúsculas, solicitadas al usuario previamente. También deberá imprimir en pantalla los valores almacenados, a fin de confirmar si son correctos o, de lo contrario, pedirlos nuevamente.

a) **Análisis:** Al solicitar que el arreglo acepte las vocales, estamos hablando de un arreglo tipo carácter. Los elementos del arreglo serán las 10 vocales, tanto minúsculas como mayúsculas, introducidas por el usuario en el orden que así lo decida. Tras preguntarle, se le deberán mostrar en forma de lista las vocales tecleadas.

b) **Codificación:** En este caso se sugiere que el arreglo se declare del tipo carácter, para almacenar correctamente los elementos que serán solicitados al usuario. Como son 10 valores los que hay que solicitar, la instrucción for() nos permitirá llenar el arreglo. Por último, para mostrar el contenido completo del arreglo, nos podemos apoyar también en el ciclo for(), ya que los elementos del arreglo son consecutivos y podemos ir avanzando con el ciclo de uno en uno hasta recorrer todo el arreglo. Como se pregunta al usuario si los datos son correctos y, en caso negativo, el programa se vuelve a reiniciar, se puede emplear el ciclo do-while() para realizar dicha tarea.

c) **Comprobación**: Al escribir los 10 valores se sugiere escribir todas las vocales, primero en minúsculas y luego en mayúsculas, para determinar si se almacenan todas y si se guardan correctamente. Si quiere probar el ciclo do-while, puede contestar que los datos son incorrectos y llenar nuevamente el arreglo.

Ejercicio 5.2.5: Escriba un programa en C que lea un mensaje introducido desde el teclado y que realice una pequeña codificación imprimiéndolo a la inversa. Por ejemplo: *Luis* y que imprima s*iuL*.

a) **Análisis:** Como se solicita que el programa acepte cadenas de texto, la declaración del arreglo tendrá que ser del tipo carácter. Además, como lo que se pide es que se imprima al revés, eso significa que se comienza del último elemento del arreglo al primero.
b) **Codificación:** Para que un usuario pueda escribir cadenas de texto no es necesario hacerlo mediante ciclos, ya que el lenguaje C ofrece una función de biblioteca llamada **gets(),** la cual simplifica esa tarea. Para mostrar el contenido completo del arreglo a la inversa, hacemos que el ciclo for() vaya del máximo al mínimo valor de índice, al ir decrementándolo. Lo anterior permitirá recorrer el arreglo del último al primer elemento. Para generalizar el programa, también se emplea la función de la biblioteca del lenguaje C llamada **strlen()**, la cual determina la longitud de una cadena de caracteres (sin considerar el carácter nulo final), es decir, el número de caracteres que el usuario tecleó y que la función gets() guardó en nuestro arreglo.
c) **Comprobación**: Para comprobar el funcionamiento de este programa escriba CONALEP y, el programa, deberá mostrar PELANOC. Además, pruebe con su nombre y otras palabras, el resultado puede ser divertido.

Ejercicio 5.2.6: Escriba un programa en C que lea la edad de 5 personas y que el usuario las introduzca cronológicamente, es decir, del más joven al de mayor edad. Posteriormente, que el programa los imprima a la inversa, para que se muestren del mayor al más joven.

a) **Análisis:** Al ser 5 personas serán 5 edades diferentes, lo que nos lleva a pensar que el arreglo tendrá un tamaño para 5 elementos y deberá aceptar únicamente números del tipo entero, ya que la edad se maneja generalmente en años. Para imprimir en el orden inverso al que el usuario los capturó, nos habla de recorrer el arreglo del último al primer elemento. Aquí se necesita la colaboración del usuario al momento de la captura, para que cuando se imprima en pantalla la lista aparezca tal y como se pide. Claro está, existen métodos de ordenación de arreglos, pero no es el momento oportuno de abordar ese tipo de técnicas, lo que aquí se solicita es más sencillo y es solo recorrer el arreglo del final al principio.
b) **Codificación:** En este caso sí es necesario apoyarnos con los ciclos, tanto para solicitar la información como para desplegar en pantalla los datos. Si bien es cierto que con cualquier ciclo se puede hacer, el ciclo ideal es el for(), ya que el contador lo podemos inicializar con el último elemento válido en el arreglo y, al ir disminuyéndolo en 1, podremos recorrer el arreglo del último al primer elemento.
c) **Comprobación**: Para verificar el funcionamiento de este programa escriba 5 edades de sus compañeros. Sea cuidadoso de que queden del más joven al mayor, a fin de que el programa los presente exactamente en orden contrario.

Ejercicio 5.2.7: Escriba un programa en C que lea una cadena de caracteres que represente un valor numérico válido (que no incluya letras, por ejemplo) y la convierta a su equivalente como número entero. La conversión debe ser con una función que solicite como parámetro una cadena y regrese la cadena convertida a valor numérico. El propósito, además, es que dicha función pueda ser empleada en otros programas.

a) **Análisis:** Tal y como está el planteamiento se podría pensar que si se escribe una palabra la función devolverá su valor equivalente en número, sin embargo, se podrá dar cuenta que al usar la función scanf() se requiere manejar un formato distinto para una cadena de caracteres y para un valor numérico, esto es, con scanf() no se puede leer una cadena de caracteres y almacenarla en una variable de tipo entero o float, por ejemplo. Entonces, la idea es sustituir a la función scanf() para simplificar la solicitud de datos.
b) **Codificación:** De inicio el programa requerirá de un arreglo donde reciba la información proporcionada por teclado. Para la captura se puede emplear la función gets() y, adicionalmente, será necesario investigar si el lenguaje C cuenta con alguna función que haga la conversión de una cadena de caracteres a número. La cadena recibida deberá ser enviada como parámetro a nuestra función de conversión, la cual a su vez, empleará la función de conversión de la biblioteca del lenguaje C (si es que existe). El resultado numérico final será devuelto de la función de biblioteca a nuestra función de conversión y, de esta, a quien la haya llamado.
c) **Comprobación**: Probar este programa consistiría en escribir un número primero sin decimales y luego con decimales y observar los resultados.

Ejercicio 5.2.8: Escriba un programa en C que lea dos valores numéricos y que realice una multiplicación con ellos. El propósito es que se utilice la función de conversión desarrollada en el ejemplo 5.2.7 al momento de solicitar cada número a multiplicar, es decir, que se puede prescindir de la función scanf().

a) **Análisis:** El hacer un programa que multiplique carece de mucha complejidad a este nivel de avance, la única variante es que en lugar de utilizar scanf() se pide usar la función desarrollada en el ejercicio anterior, al momento de solicitar los números.
b) **Codificación:** Todos los elementos para implementar el programa ya se conocen, lo único distinto es el empleo de una función scanf() para pedir los datos y hay dos alternativas: una, se guarda el ejercicio previo con la extensión .h y se incluye en la cabecera u, otra, se reescribe la función para convertir un número en cadena de caracteres a valor numérico desarrollada en el ejercicio previo, de tal manera que pueda ser empleada para sustituir a scanf() al momento de pedir los valores a multiplicar.
c) **Comprobación**: Valdría la pena, por la simpleza al introducir los datos que puede ser 5 x 5 y que seguramente el resultado será 25, que lo mejor es seguir paso a paso la ejecución para que le dé seguimiento al llamado a las funciones y a cómo el número que se recibe como cadena se convierte en un valor numérico.

Ejercicio 5.2.9: Realiza un programa que solicite una cadena de texto y que luego se escriba un carácter que será buscado en el texto anterior, a fin de conocer su posición. Con el uso de funciones y arreglos, realiza una función que reciba como parámetros los datos antes mencionados y que devuelva el número de posición en la que aparece dicho carácter.

a) **Análisis:** El pedir la captura de una cadena o un carácter es algo que ya se ha hecho antes, lo que está interesante es que se pide recorrer el arreglo, pero en búsqueda de elementos. Por

este motivo es necesario incorporar más estructuras de control condicional para ir comparando cada elemento del arreglo con el carácter propuesto por el usuario y, de ser localizado, informar en qué posición se encuentra.

b) **Codificación:** Después de pedir tanto la cadena caracteres como el carácter individual se invocará a la función de búsqueda, pasando como argumentos dichos valores. Será la función la que nos devolverá si fue localizado o no dicho carácter en la cadena. Para este caso se propone un ciclo while(), ya que mientras va recorriendo el arreglo puede ir comparando el carácter con cada uno de sus elementos y, al mismo tiempo, revisar si aún no se llega al final de la cadena. Así que lo único que se necesitaría dentro del ciclo es una variable que vaya incrementando su valor, a fin de poder comparar con el siguiente elemento del arreglo. La primera coincidencia encontrada terminará el ciclo, y la función devolverá el valor almacenado en la variable usada como contador.

c) **Comprobación**: Si decide la ejecución del programa desarrollado paso a paso, se recomienda escribir una palabra de cuatro o cinco letras para que ayude a entender la lógica del ciclo y que no sea tan tedioso el ir comparando elemento por elemento del arreglo. Otra recomendación es precisamente que le pida al depurador que vaya mostrando los valores del arreglo, de la variable, así como del contador; para poder darle seguimiento al desarrollo del programa. Una última recomendación es que siempre escriba caracteres que sí fueron tecleados, ya que así se pide en el planteamiento del ejercicio, solo búsquedas exitosas.

Ejercicio 5.2.10: Desarrolla un programa que simule la captura de un mensaje de texto para celular y, al final, avise si se cobrará por el servicio de envío un mensaje o dos, si el texto supera los 40 caracteres. El objetivo es que se determine la longitud del texto capturado y, con base en esto, se avise al usuario lo que tendrá que pagar.

a) **Análisis:** Sabemos que existe la función gets() que nos permite introducir una cadena y que esa cadena puede ser almacenada en un arreglo, pero cómo saber cuántos caracteres fueron tecleados por el usuario. Existe una función de biblioteca llamada strlen(), la cual nos ayuda a saber la longitud de una cadena, sin embargo, nos dará el mismo resultado un ciclo que vaya recorriendo la cadena hasta localizar el carácter nulo del final (‘\0’), a la vez que contabiliza los elementos *recorridos*. Se debe incorporar, además, una estructura de control condicional para comparar la longitud del texto y avisarle al usuario que se le hará un cobro doble en el caso de que rebase los 40 caracteres, que es la marca que se está poniendo como límite.

b) **Codificación:** Respecto al código, basta con solicitar el texto y determinar su tamaño. Como se dijo en el análisis, hay dos opciones. En este caso sugerimos la más sencilla, usar strlen() que viene incluida en la librería <string.h> o, si el programador lo prefiere, implementar su propia función. Además, aquí se pide habilitar un control condicional que se usa mucho en la televisión cuando pasan mensajes por celular y es cortarlos cuando rebasan el tope permitido, aunque aquí como la compañía lo puede mandar en dos partes, lo único es que con un if() se avise si se quiere pagar el envío doble.

c) **Comprobación**: Probar el programa consistirá en ejecutar el programa en dos ocasiones. En una escribir un texto breve, alrededor de 20 caracteres y, la segunda vez, un texto que rebase los 40 caracteres. Notar la diferencia.

Ejercicio 5.2.11. Elaborar un programa que simule el acceso a un sistema y que dé 3 oportunidades de escribir una contraseña. Después del tercer intento se debe avisar que el equipo será bloqueado por seguridad y, de ser correcta la contraseña, que presente un mensaje de bienvenida.

a) **Análisis:** Desde 2 ejercicios previos, nos damos cuenta que el objetivo ha sido manejar operaciones con cadenas y este no es la excepción. La petición que se nos hace es muy socorrida por los sistemas operativos y en ciertas páginas web que ofrecen correo electrónico donde el usuario tiene que identificarse, entonces nos encontramos con que ahora se necesita comparar dos cadenas y, para eso, habrá que investigar si el lenguaje C tiene alguna función que realice tal operación. Del resultado de esa comparación se podrá decidir cuál mensaje se imprime, es decir, un tipo de mensaje cuando acierte el usuario y otro cuando falle.
b) **Codificación:** Con relación a si el lenguaje C ofrece algún tipo de función de comparación entre cadenas de caracteres, efectivamente, existe y se denomina strcmp(), la cual viene incluida en la librería <string.h>. La función strcmp() devuelve 0 (cero) cuando las cadenas comparadas son iguales y un valor diferente de 0 (cero) si no lo son. En nuestro caso nos interesa cuando devuelva 0 (cero), porque con un if() podremos presentar un mensaje u otro. Además, deberemos llevar un control del número de intentos realizados por el usuario, a través de una variable, para que, en caso de que agote sus tres oportunidades, el sistema avise que se bloqueará. Este control puede ser a través de un ciclo do-while, ya que permitirá la ejecución del ciclo de manera condicionada, es decir, que dependerá de un evento para su interrupción.
c) **Comprobación**: De acuerdo a la solución programada por los autores, la clave es “geras”, pero puede tratar escribiendo “Geras” o “GERAS”; al final, la idea es equivocarse varias veces para analizar el comportamiento del programa.

Ejercicio 5.2.12: Elaborar un programa en lenguaje C que pregunte las 6 asignaturas que cursa un estudiante del CONALEP, las almacene en un arreglo, calcule el promedio semestral e informe si tiene “derecho a beca”, si su promedio final semestral es mayor a 9. Además deberá mostrar un mensaje de “repite curso” en aquellas materias cuyo promedio de calificación sea menor a 7.

a) **Análisis:** Es claro, que se requiere la declaración de un arreglo del tipo float que permita almacenar la calificación promedio de cada asignatura, pero como no se pide almacenar el nombre, únicamente la posición en el arreglo nos orientará para saber si se trata de la segunda o de la cuarta asignatura. Además, mientras se recorre el arreglo se puede ir verificando si la calificación es aprobatoria o no, y avisar en ese instante. El arreglo se recorre también para ir sumando las calificaciones y, posteriormente, dividir la suma entre el número de asignaturas.

El programa termina cuando le avise al alumno si tiene derecho a beca, una vez calculado el promedio y si este es mayor a 9.

b) **Codificación:** Para implementar este programa se requerirá, como ya se dijo en el análisis, de un arreglo de 6 elementos del tipo float, un ciclo for() para recorrer el arreglo e ir acumulando las calificaciones y una instrucción if(), para ir verificando si alguna de las asignaturas está reprobada. Una vez hecho lo anterior podemos avisarle al usuario o alumno sobre su situación, con los mensajes apropiados en cada caso.

c) **Comprobación**: El funcionamiento del programa se puede probar en 3 casos diferentes, como mínimo: Cuando el estudiante aprueba todas sus asignaturas pero su promedio es menor a 9, cuando su promedio es mayor a 9 y cuando reprueba algunas o todas sus asignaturas. Intente los casos anteriores, escribiendo calificaciones que reflejen esas 3 situaciones y observe los mensajes que aparecerán en cada caso.

Ejercicio 5.3.1: Elaborar un programa en lenguaje C que pida un número entero y avise a qué mes del año es equivalente, usando para ello arreglos bidimensionales.

a) **Análisis:** Este programa puede ser resuelto empleando estructuras de control condicional, sin embargo se nos pide aplicar el concepto de arreglos, los cuales permiten almacenar cadenas completas en cada posición del arreglo. Así que el programa solicita un número entre 1 y 12 y el programa dirá a qué mes del año corresponde.

b) **Codificación:** Opcionalmente se puede utilizar la instrucción if() para avisar si el usuario escribió un numero diferente al rango 1 a 12, que son los meses del año. Fuera de eso, con un simple printf() e incluyendo en su formato que se trata de una cadena lo que se quiere imprimir, lo único que queda es poner el nombre del arreglo y dentro de los corchetes la variable que almacena el número introducido por el usuario. No está de más decir que el arreglo tendrá que ser declarado y, en ese momento, inicializado con todos los meses del año; con la finalidad de poder presentarle al usuario el mes que elija.

c) **Comprobación**: El programa podrá ser probado escribiendo cualquier mes con un número, y deberá presentarse en pantalla el nombre del mes al cual es equivalente. Por ejemplo, si escribe 6, en pantalla aparecerá Junio; pero intente con un valor mayor a 12 o menor a 1 y revise lo que ocurre en tales casos.

Ejercicio 5.3.2: Elaborar un programa en lenguaje C que pida un número entero y avise a qué estación del año es equivalente, usando arreglos bidimensionales.

a) **Análisis:** Como el enunciado solo señala que se pida un dato, en particular el número de estación, entonces, al momento de declarar el arreglo, será necesario inicializarlo con los nombres de cada una de las estaciones. Luego, el programa deberá solicitar un número entre 1 y 4 y el programa avisará a qué estación del año corresponde.

b) **Codificación:** Se puede utilizar la instrucción if() opcionalmente para avisar si el usuario escribió un numero diferente al rango 1 a 4, que son las estaciones del año, porque un printf() nos ayudará para mostrar el contenido del arreglo. Lo único que falta es poner el nombre del arreglo y, dentro de los corchetes, la variable que almacena el número introducido por el usuario.

c) **Comprobación**: Si no se utilizó un if() para validar la respuesta del usuario, se recomienda escribir únicamente valores entre 1 y 4, para evitar un mal comportamiento del programa con respecto a la memoria. De tal caso que si escribe 3, la computadora deberá contestar OTOÑO.

Ejercicio 5.3.3: Con el manejo de arreglos bidimensionales crear un programa que permita guardar el nombre de 10 concursantes de un certamen de belleza así como la puntuación total asignada por 3 jueces a cada concursante. Al final, el programa debe informar el nombre de la ganadora.

a) **Análisis:** La solución a este problema podría ser con arreglos multidimensionales, sin embargo como no se definen requerimientos en ese sentido, se puede implementar la solución con dos arreglos independientes, en uno se almacena el nombre de la concursante y en otro la puntuación que le asignaron los jueces. Para localizar a la ganadora, se requiere ir comparando la puntuación entre sí y detectar en qué posición está la puntuación más alta. Con ese dato podremos saber quién es la ganadora.

b) **Codificación:** Es cierto que con un ciclo iremos capturando los nombres de las concursantes y aprovechar ese momento para que en esa misma posición en el otro arreglo se guarde la puntuación total que les corresponda. Esa parte es clave, porque con otro ciclo se irán comparando las puntuaciones y se guardará en una variable en qué posición está el puntaje más alto. Con eso y mediante la función printf(), se presentará el nombre de la ganadora, confiando en que ambas ubicaciones coinciden (el nombre y el puntaje).

c) **Comprobación**: Una forma rápida de probar la eficacia del programa es ir escribiendo las letras del abecedario y en la puntuación poner, para la primera concursante un 1 para cada juez; para la segunda concursante un 2, para todos los jueces; 3 para la tercera concursante y así hasta la última. La ganadora deberá ser la concursante número 10. Con esto, se pueden introducir otros datos ya con nombres y puntuaciones aleatorias y que el programa imprima el nombre de la ganadora.

Ejercicio 5.3.4: Con el manejo de arreglos bidimensionales crear un programa que permita guardar el nombre de 16 “cantantes” en un concurso de *Reality Show* llamado *La blasfemia de academia* y, que a cada cantante, se le asigne el total de votos enviados por la gente vía mensajes de texto. El programa también deberá indicar el nombre del expulsado del show, para aquel cantante que tenga menos votos.

a) **Análisis:** Para simplificar la solución se pueden utilizar 2 arreglos, en uno se guarda el nombre de los “cantantes” y, en otro, la votación hecha por la gente. Para localizar al expulsado se requiere ir comparando la votación entre sí y detectar, en una variable, en qué posición está la votación más baja. Con ese dato podremos saber quién es el expulsado.

b) **Codificación:** De manera paralela, al momento de ir pidiendo los nombres se puede ir registrando la votación, para conseguir con eso que ambos datos queden en la misma posición, aunque estén en diferentes arreglos. Esa parte es clave, porque con otro ciclo se irán comparando las votaciones y se guardará en qué posición está el menor número de votos. Con eso y mediante la función printf(), se presentará el nombre del expulsado, confiando en que ambas ubicaciones coinciden (el nombre y la votación).

c) **Comprobación**: Se puede probar este programa con el mismo procedimiento utilizado en la comprobación del ejercicio 5.3.3. Una vez comprobada la precisión del programa, se pueden introducir otros datos ya con nombres y votaciones al azar y que el programa imprima el nombre del expulsado.

Para más ejercicios sobre el manejo de arreglos con el lenguaje de programación C, véase:

http://www.jorgesanchez.net/programacion/ejerciciosc/arrays.html. Recuperado el 5 de julio de 2014.

APÉNDICE

Ejercicio 1

1.- Estructurados. 2.- Rango. 3.- Vectores. 4.- Elemento. 5.- Homogéneos. 6.- Arreglos. 7.- Memoria. 8.- Cero. 9.- Ordenamiento. 10.- Búsqueda.

Ejercicio 2

1.- V; 2.- F; 3.- V; 4.- V; 5.- F; 6.- V; 7.- F; 8.- F; 9.- V; 10.- F

Ejercicio 3

Declaración de un arreglo tipo vector

Tipo de dato nombre del arreglo[rango] ;

Consulta de un arreglo tipo vector

Nombre del arreglo[índice];

Declaración de un arreglo bidimensional

Tipo de dato Nombre del arreglo[Índice de filas] [Índice de columnas];

Ejercicio 4

1.- (h); 2.- (d); 3.- (e); 4.- (i); 5.- (b); 6.- (c); 7.- (f); 8.- (g); 9.- (a)

Ejercicio 5

Consulta de un arreglo tipo bidimensional

Nombre del arreglo[Índice de fila][Índice de columna] ;

Consulta de un elemento del arreglo

Nombre del arreglo [Índice del elemento];

Declaración de cadena de caracteres con sus elementos

char Nombre del arreglo[7] = "cadena";

Declaración con asignación de valores de arreglo multidimensional

Tipo de dato Nombre del arreglo[rango de las filas][rango de las columnas] =
{{fila0col0 , fila0col1},
{fila1col0 , fila1col1}};

Ejercicio 6

1.- F; 2.-V; 3.- V; 4.- F; 5.- F; 6.- V; 7.- F; 8.- V; 9.- F; 10.- V

Ejercicio 7

Los _**arreglos**__, o _**arrays**__, son conjuntos de datos del mismo _**tipo**__. Estos se deben _**declarar**_ para indicar el tipo de datos que contendrán, e _**inicializar**__ para indicar el _**tamaño**__ o _**rango**__ del arreglo. Podemos tener un arreglo de una dimensión llamado __**vector**__ o de dos dimensiones llamado _**matriz**__ o arreglo _**bidimensional**__. Independientemente del tipo de arreglo, estos ocupan un bloque bien definido de _**memoria**__.

Los _**índices**__, en un arreglo de dos dimensiones, indican el número de __**filas**__ y de columnas que lo conforman, estos son colocados entre [ ]. Podemos utilizar a los arreglos como _**parámetros**__ de una _**función**_, siendo cada uno de los _**elementos**__ del arreglo un dato de entrada a dicha función.

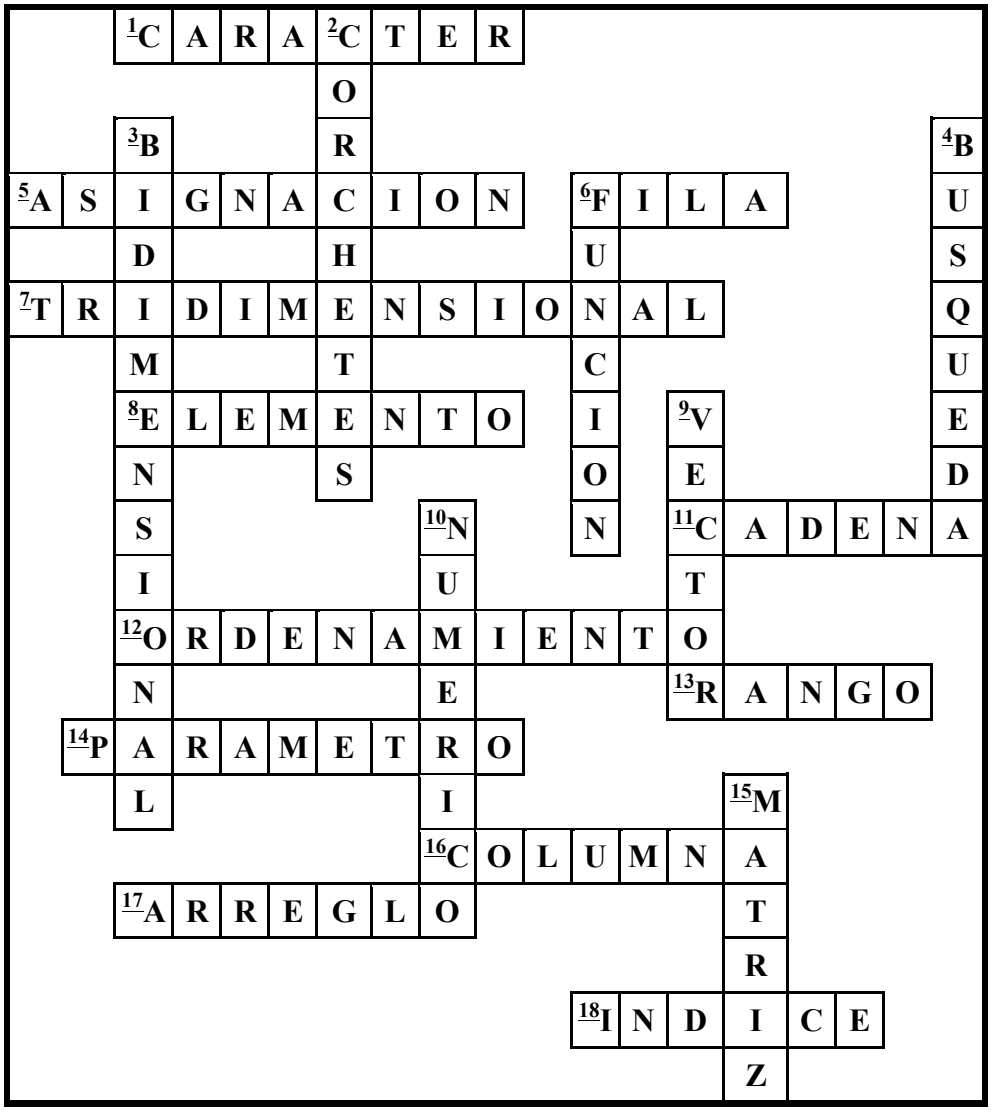
Podemos tener arreglos o **cadenas** de caracteres y, para indicar su final, empleamos el carácter **nulo** (‘\0’).

Es importante recordar que los arreglos pueden ser utilizados para realizar operaciones de **ordenamiento**, ya sea creciente o decreciente, así como de **búsqueda** de elementos.

Sopa de letras

						F	U	N	C	I	O	N					P		
			B							T							A		
		I	N	I	C	I	A	L	I	Z	A	R					R		
					D							M					A		
					F	I	L	A					A				M		
N	U	L	O				M							Ñ			E		
					A			E			V		D		O		T		O
		R			D				N			E					R		R
		A	R	R	E	G	L	O		S	C		C				O		D
		N			U					L	I			T			S		E
		G			Q				A			O			O				N
		O			S			R					N			R			A
		I			U		A	R	R	A	Y	S		A					M
			N		B	R									L				I
C				D					E	L	E	M	E	N	T	O	S		E
	A			T	I	P	O												N
		D				C					M	E	M	O	R	I	A		T
			E				E												O
				N					M	A	T	R	I	C	E	S			
					A														

Crucigrama


1C A R A 2C T E R
O
3B R 4B
5A S I G N A C I O N 6F I L A U
D H U S
7T R I D I M E N S I O N A L Q
M T C U
8E L E M E N T O I 9V E
N S O E D
S 10N N 11C A D E N A
I U T
12O R D E N A M I E N T O
N E 13R A N G O
14P A R A M E T R O
L I 15M
16C O L U M N A
17A R R E G L O T
R
18I N D I C E
Z

Listado de Ejercicios RESUELTOS del apartado 5

Nota: Las soluciones a los ejercicios fueron probadas en las versiones para Windows de los compiladores Turbo C++ versión 3.0 y Dev-C++ versión 4.9.9.2.

Listado 5.2.1

```
/* ----------------------------------------------------------------
Programa: Ejercicio5-2-1.c
Objetivo: Ejemplificar la inicialización y el acceso a arreglos de una
      dimensión.
----------------------------------------------------------------*/
#include <stdio.h>

/* Inicialización de arreglos */
char semana[8] = "LMMJVSD"; /* El octavo elemento es el '\0' */

/* Dimensionamiento automático (el compilador calcula el tamaño) */
int meses[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

void main()
{
      static int horas[8] = {1, 2, 3, 4, 5, 6, 7, 8};

      clrscr();

      /* Los índices empiezan en 0, el elemento N tiene el índice N-1 */
      printf("\n Sexto mes del año: %d", meses[5]);
      printf("\n Jornada de trabajo de medio tiempo: %d", horas[3]);
      printf("\n Tercer día de la semana: %c", semana[2]);

      getch();

}
```

Listado 5.2.3

```
/* ------------------------------------------------------------------
Programa: Ejercicio5-2-3.c
Objetivo: Llenar y accesar arreglos unidimensionales.
------------------------------------------------------------------*/
#include <stdio.h>

void main()
{
      static int valores[10], i;
      char resp = ' ';

      do
      {
            for (i = 0; i < 10; i++)
            {
                  printf("\n Introduce el valor %d: ", i + 1);
                  scanf("%d", &valores[i]);
            }

            printf("\n Los valores introducidos fueron \n ");

            for (i = 0; i < 10; i++)
                  printf("\n El valor %d fue: %d", i + 1, valores[ind]);

            printf("\n Los valores son correctos (s/n) ? ");
            getche(resp);
            getche(resp);

      } while (resp == 'n' || resp == 'N');

}
```

Listado 5.2.5

```
/* ----------------------------------------------------------------------
Programa: Ejercicio5-2-5.c
Objetivo: Manejo de cadenas de caracteres en arreglos unidimensionales.
----------------------------------------------------------------------*/
#include <stdio.h>

#define MAX 80

void main()
{
      static char mensaje[MAX];
      int i, tamcad;

      clrscr();

      printf("\n Introduzca mensaje (máximo %d caracteres): ", MAX);
      gets(mensaje);

      tamcad = strlen(mensaje);

      printf("\n El mensaje codificado es: \n");

      /* El primer elemento del arreglo es el 0, el último es tamcad - 1 */
      for (i = tamcad - 1; i >= 0; i--)
            printf("%c", mensaje[i]);

      getch();

}
```

Listado 5.2.7

```
/* ----------------------------------------------------------------
Programa: Ejercicio5-2-7.c
Objetivo: Manejo de cadenas de caracteres con funciones.
----------------------------------------------------------------*/
#include <stdio.h>

#define MAX 80

int convierteCaN(char []);

void main()
{
      char cadena[MAX];
      int num;

      clrscr();

      printf("\n Introduzca la cadena ");
      gets(cadena);

      num = convierteCaN(cadena);

      printf("\n El valor numérico de %s es: %d \n", cadena, num);
      getche();

}

/* ----------------------------------------------------------------
Función: convierteCaN()
Objetivo: Convertir una cadena de texto que represente un número en
      su equivalente como valor numérico.
Parámetros: La cadena de caracteres a convertir.
Salida: La cantidad convertida en valor numérico
----------------------------------------------------------------*/
int convierteCaN(char texto[])
{
      int valor;

      /* La función de biblioteca atoi() convierte una cadena de
      caracteres, que represente un número, en su equivalente numérico */
      valor = atoi(texto);

      return valor;

}
```

Listado 5.2.9

```
/* ----------------------------------------------------------------
Programa: Ejercicio5-2-9.c
Objetivo: Manejo de cadenas de caracteres con funciones.
----------------------------------------------------------------*/
#include <stdio.h>

#define MAX 80

int buscaletra(char, char []);

void main()
{
      char cadena[MAX], letra;
      int num, tam;

      clrscr();

      printf("\n Introduzca la cadena: ");
      gets(cadena);

      printf("\n Introduzca el carácter a buscar: ");
      scanf("%c", &letra);

      tam = strlen(cadena);
      num = buscaletra(letra, cadena);

      if (num != -1)
            printf("\n El carácter %c está en la posición %d",
                  letra, num + 1);
      else
            printf("\n Ese carácter no aparece en la cadena");

      getch();
 }

/* ----------------------------------------------------------------
Función: buscaletra()
Objetivo: Buscar un carácter en una cadena de caracteres y devolver su
      posición.
Parámetros: La cadena de caracteres donde se busca y el carácter buscado.
Salida: La posición en la cadena en la que aparece el carácter o -1 si no
      se localizó en ella.
----------------------------------------------------------------*/
int buscaletra(char caracter, char texto[])
{
      int pos = 0;

      while ((caracter != texto[pos]) && (texto[pos] != '\0'))
            pos++;

      if (texto[pos] == '\0') return -1; /* Carácter no encontrado */

      return pos;

}
```

Listado 5.2.11

```
/* ----------------------------------------------------------------------
Programa: Ejercicio5-2-11.c
Objetivo: Uso de arreglos unidimensionales
----------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>

#define MAX_INTENTOS 3

void main()
{
	int cuenta, incrementa;
	char pass[6] = "geras", cve[6] = {' ',' ',' ',' ',' ',' '}, nom[25];
	cuenta = 1;

	clrscr();

	do
	{
		printf("\n \n Nombre de usuario: ");
		gets(nom);

		printf("\n Contraseña (máximo 5 caracteres): ");
		gets(cve);

		if (strcmp(cve, pass) == 0)
		{
			printf("\n Bienvenido %s al sistema, Acceso autorizado",
				nom);
			getch();
			exit(); /* Terminar el programa para simular el acceso */

		}
		else
		{
			if (cuenta <= MAX_INTENTOS)
			{
				printf("\n Clave incorrecta, intente de nuevo");
				printf("\n Al 3er. fallo se bloquea el sistema");
			}

			printf("\n\n Intento %d agotado", cuenta++);

		}

	} while (cuenta <= MAX_INTENTOS);

	if (cuenta > MAX_INTENTOS)
		printf("\n Por cuestiones de seguridad el sistema se bloqueó");

	getch();

}
```

Listado 5.3.1

```
/* ----------------------------------------------------------------
Programa: Ejercicio5-3-1.c
Objetivo: Uso de arreglos bidimensionales y su inicialización
----------------------------------------------------------------*/
#include <stdio.h>

void main()
{
      int num_mes;

      char mes[][12] = {
            "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
            "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"
      };

      clrscr();

      printf("\n Número del mes (1-12): ");
      scanf("%d", &num_mes);

      if ((num_mes > 0) && (num_mes < 12))
      {
         printf("\n Su equivalente es %s", mes[num_mes - 1]);

      }
      else
      {
         printf("\n El número de mes introducido es incorrecto");

      }

      getch();

}
```

Listado 5.3.3

```
/* ----------------------------------------------------------------
Programa: Ejercicio5-3-3.c
Objetivo: Uso de arreglos bidimensionales para su acceso y recorrido.
----------------------------------------------------------------*/
#include <stdio.h>

char concursantes[80][10];

void main()
{
      int j1, j2, j3, i, mayor;
      int puntuacion[10];

      clrscr();
      printf("\n Introduzca los nombres de las concursantes ");

      /* Entrada de datos */

      for (i = 0; i < 10; i++)
      {
            printf("\n Nombre de la concursante N° %d: ", ind + 1);
            scanf("%s", concursantes[i]);

            printf("\n Calificación del juez N° 1: ");
            scanf("%d", &j1);

            printf("\n Calificación del juez N° 2: ");
            scanf("%d", &j2);

            printf("\n Calificación del juez N° 3: ");
            scanf("%d", &j3);

            puntuacion[i] = j1 + j2 + j3;

      }

      /* Determinar a la ganadora */

      mayor = 0;
      for (i = 1; i < 10; i++)
            if (puntuacion[i] > mayor)
                  mayor = i;

      /* Presentar a la ganadora */

      printf("\n La ganadora del concurso es %s, con %d puntos",
            concursantes[mayor], puntuacion[mayor]);

      getch();

}
```

Sexta parte

Apuntadores

6.1. Apuntadores

Ejercicio 1

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Valor izquierdo	Nulo	Expresiones	Valor
Variable	Indirección	Dirección	Contenido
Apuntadores	Operador de dirección		

Enunciados:

1.- Llamamos a los apuntadores variables de _____ porque indican la ubicación de otra variable en memoria.

2.- Los _____ son variables cuyo valor son direcciones de memoria.

3.- Se le conoce como valor derecho o _____, porque es el valor asignado a la variable.

4.- El _____ parte del apuntador que indica la dirección o ubicación de una variable.

5.- Se dice que un apuntador es _____ cuando su contenido es cero.

6.- El símbolo * en un operador, se le conoce como operador de _____.

7.- Se le llama _____ al símbolo **&** utilizado en el lenguaje C para conocer el valor izquierdo de la variable.

8.- En el valor izquierdo de un apuntador tenemos la dirección de otra _____.

9.- Los apuntadores no se aplican a constantes ni a _____.

10.- El operador de indirección devuelve el _____ del objeto al que apunta.

Ejercicio 2

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Incremento	( )	1.- Contiene la dirección de un apuntador que apunta a una variable.
b) void	( )	2.- Operación de apuntadores que sirve para conocer la posición relativa de dos apuntadores en memoria.
c) Desreferencia	( )	3.- Apuntador que no apunta a nada, es una constante simbólica definida en la cabecera <stddef.h>.
d) Comparación	( )	4.- Declaración de tipo genérica, permitiendo apuntar a variables de tipos diferentes.
e) Decremento	( )	5.- Error frecuente en la manipulación de las direcciones de memoria.
f) Indirección múltiple	( )	6.- Operación con apuntadores donde se incrementa la dirección de memoria del apuntador.
g) Variable válida	( )	7.- Un apuntador NULL no permite apuntar a una…
h) Asignación de apuntadores	( )	8.- Operación con apuntadores donde se decrementa la dirección de memoria del apuntador.
i) NULL	( )	9.- Operador que devuelve el valor de la variable a la que apunta.
j) Accesar una dirección no válida	( )	10.- Es hacer que dos apuntadores apunten a la misma variable, considerando que ambos son del mismo tipo.

6.2. Apuntadores y arreglos

Ejercicio 3

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Es imposible hacer que un apuntador haga referencia a un elemento de un arreglo. ( )
2. Es posible tener arreglos de apuntadores, ya que los apuntadores son variables. ( )
3. Es inadecuado utilizar operaciones aritméticas con apuntadores. ( )
4. Es imposible almacenar la dirección de un elemento de un arreglo en un apuntador. ( )
5. Un arreglo es una variable apuntador. ( )
6. Es inadmisible declarar una cadena de caracteres con apuntadores. ( )
7. En un arreglo de caracteres, cada elemento ocupa un byte de memoria. ( )
8. Es posible hacer comparaciones entre apuntadores. ( )
9. El nombre del arreglo es un índice a la dirección del comienzo del arreglo. ( )
10. Está prohibido utilizar apuntadores en arreglos multidimensionales. ( )

Ejercicio 4

Instrucciones: Utilizando las expresiones del lado derecho, escribe la sintaxis correcta de cada una de las instrucciones siguientes:

1.- Sintaxis de declaración de apuntador

_____ _____ _____ ;	Nombre del apuntador Tipo de dato *

2.- Sintaxis de apuntador a apuntador (indirección múltiple)

_____ _____ _____ _____;	Tipo de dato * Nombre del apuntador *

3.- Sintaxis de incremento de apuntadores

_____ = _____ _____ _____;	Apuntador Apuntador + Número de incremento

4.- Sintaxis de la declaración de un apuntador de tipo entero y de la asignación de la dirección del primer elemento de un arreglo de tipo entero al apuntador.

_____ _____ _____ ; _____ = _____ _____ [_____];	arreglo int & ptr * ptr 0

6.3. Manejo dinámico de memoria

Ejercicio 5

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Asignación dinámica de memoria	( )	1.- Esta función sirve para asignar un espacio de memoria en tiempo de ejecución, de un tamaño específico.
b) realloc( )	( )	2.- La función malloc( ) devuelve un apuntador de tipo…
c) free( )	( )	3.- Esta función sirve para asignar un espacio de memoria en tiempo de ejecución, en bloques de un tamaño específico.
d) void	( )	4.- Esta función sirve para modificar el espacio de memoria asignado en tiempo de ejecución.
e) malloc( )	( )	5.- Característica del lenguaje C que nos permite crear cambios en el tamaño de datos, en tiempo de ejecución, de arreglos y estructuras.
f) Copia	( )	6.- Archivo utilizado para las funciones de asignación y liberación de memoria dinámica.
g) calloc( )	( )	7.- En la función realloc( ) si el bloque asignado de memoria se incrementa, pero no puede redimensionarse, el bloque original se _____ a otro espacio de memoria.
h) stdlib.h	( )	8.- Esta función sirve para liberar espacio de memoria asignada en tiempo de ejecución.

Ejercicio 6

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Las variables globales se almacenan en lugares dinámicos de memoria. ( )
2. Las variables locales de una función son creadas y destruidas mientras se utiliza la función. ( )
3. Las variables estáticas se almacenan en memoria dinámica. ( )
4. Al declarar una variable el compilador no reserva espacio de memoria. ( )
5. La asignación dinámica de memoria es utilizada en tiempo de ejecución para variar el tamaño de diversas estructuras de datos. ( )
6. Con la función **free()** se libera espacio de memoria. ( )
7. El manejo de memoria estática ahorra espacio de memoria. ( )
8. Es imposible utilizar apuntadores en la asignación de memoria dinámica. ( )
9. **calloc()** es una función de asignación de memoria. ( )
10. Es posible asignar espacio de memoria de tamaño desconocido. ( )

Ejercicio 7

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Izquierdo	Asterisco	Asignación	Ampersand
Incremento	Apuntar	Punteros	Variable
Derecho	Elemento	Nulo	Decremento
Operaciones	Apuntadores	Dirección	Arreglos
Compilador	Matrices	Vectores	Lenguaje

Enunciados:

Los _____ o _____ son utilizados en el _____ C para apuntar a una _____ y almacenar su _____ en memoria, por este motivo se les llaman variables de dirección. Estos apuntadores utilizan el símbolo _____ (&) para indicar su dirección; indicamos al _____ por medio de un símbolo _____ que la variable es un apuntador.

En el valor _____ de un apuntador, tenemos la dirección de la variable a la que apunta; y, en su valor _____, su contenido. Pero si le asignamos un valor _____, no puede _____ a alguna variable válida. Asímismo, varios apuntadores pueden apuntar a la misma dirección, en este caso utilizaríamos una expresión de _____ para copiar dichos apuntadores.

Otra de las _____ que se pueden realizar con apuntadores es el _____ de la dirección a la que apuntan, al aumentar su valor. Asímismo, existe la operación opuesta, que consiste en el _____ de la dirección a la que apuntan, al disminuirla.

También es importante decir que los apuntadores son utilizados en conjunto con los _____ cuando se apunta a su primer _____ y, así, direccionar a los demás que lo conforman; tanto en una dimensión o _____, en dos dimensiones o _____ y en más de este número de dimensiones.

Sopa de letras

Encuentre las palabras siguientes en esta sopa de letras: Izquierda, asterisco, asignación, ampersand, apuntar, punteros, variables, derecho, elemento, nulo, decremento, operaciones, arreglos, apuntadores, dirección, arreglos, compilador, matrices, vectores y lenguaje.

A	S	D	D	I	R	E	C	C	I	O	N	O	I	U	Y	T	R	E	W
S	D	F	G	H	J	K	L	Ñ	Z	D	G	L	E	N	G	U	A	J	E
A	R	R	E	G	L	O	S	K	Q	D	F	G	H	J	K	L	O	V	D
P	F	G	R	F	G	U	I	L	U	S	R	T	U	H	I	L	S	U	E
U	G	T	E	A	S	T	E	R	I	S	C	O	F	Y	U	F	R	C	D
N	H	G	O	M	E	W	E	T	E	L	E	M	E	N	T	O	F	O	F
T	U	V	T	P	D	Q	D	Y	R	S	D	R	T	Y	U	O	V	M	V
A	Y	E	A	E	E	A	C	U	D	F	E	R	I	Y	T	V	D	P	G
D	T	C	A	R	V	R	S	R	A	P	U	N	T	A	R	U	S	I	T
O	R	T	D	S	I	B	A	S	O	S	E	D	R	U	J	J	C	L	Y
R	E	O	T	A	R	A	H	C	W	D	I	A	D	R	P	R	F	A	H
E	D	R	R	N	O	Y	B	U	I	F	H	G	E	U	G	T	R	D	N
S	F	E	U	D	L	T	J	L	S	O	D	S	N	D	R	N	T	O	J
I	O	S	I	O	S	R	H	R	E	D	N	T	S	A	F	F	Y	R	U
I	N	C	R	E	M	E	N	T	O	S	E	E	A	T	C	E	U	R	I
D	A	R	R	E	G	L	O	S	B	R	E	O	S	Y	U	I	I	E	K
R	D	R	T	U	I	O	L	H	O	E	U	R	D	T	I	S	O	S	D
E	M	A	T	R	I	C	E	S	A	W	T	U	I	C	K	G	K	N	F
S	M	I	K	O	T	E	S	E	D	E	C	R	E	M	E	N	T	O	G
A	D	E	R	E	C	H	O	E	R	T	G	F	R	T	Y	U	J	M	H

Crucigrama

HORIZONTALES

1.- Función que permite asignar espacio de memoria dinámica.

4.- Función que permite reasignar espacio de memoria dinámica.

5.- Operación de apuntadores que sirve para conocer la posición relativa entre dos apuntadores en memoria.

8.- Función para liberar espacio de memoria asignada en tiempo de ejecución.

10.- Operador que sirve para colocar en un apuntador, la memoria dinámica devuelta por la función malloc().

13.- Indica el lugar donde se encuentra en memoria el contenido del apuntador.

15.- Se le conoce también como valor derecho del apuntador.

16.- Operador que devuelve el valor de la variable a la que señala un apuntador.

17.- Tipo genérico de las variables de dirección.

18.- Las variables _____ se almacenan en un lugar fijo de memoria.

VERTICALES

2.- Podemos _____ a una función con un apuntador.

3.- Podemos variar el tamaño que ocupa un vector en _____.

6.- Hacer referencia a una variable a través de un apuntador.

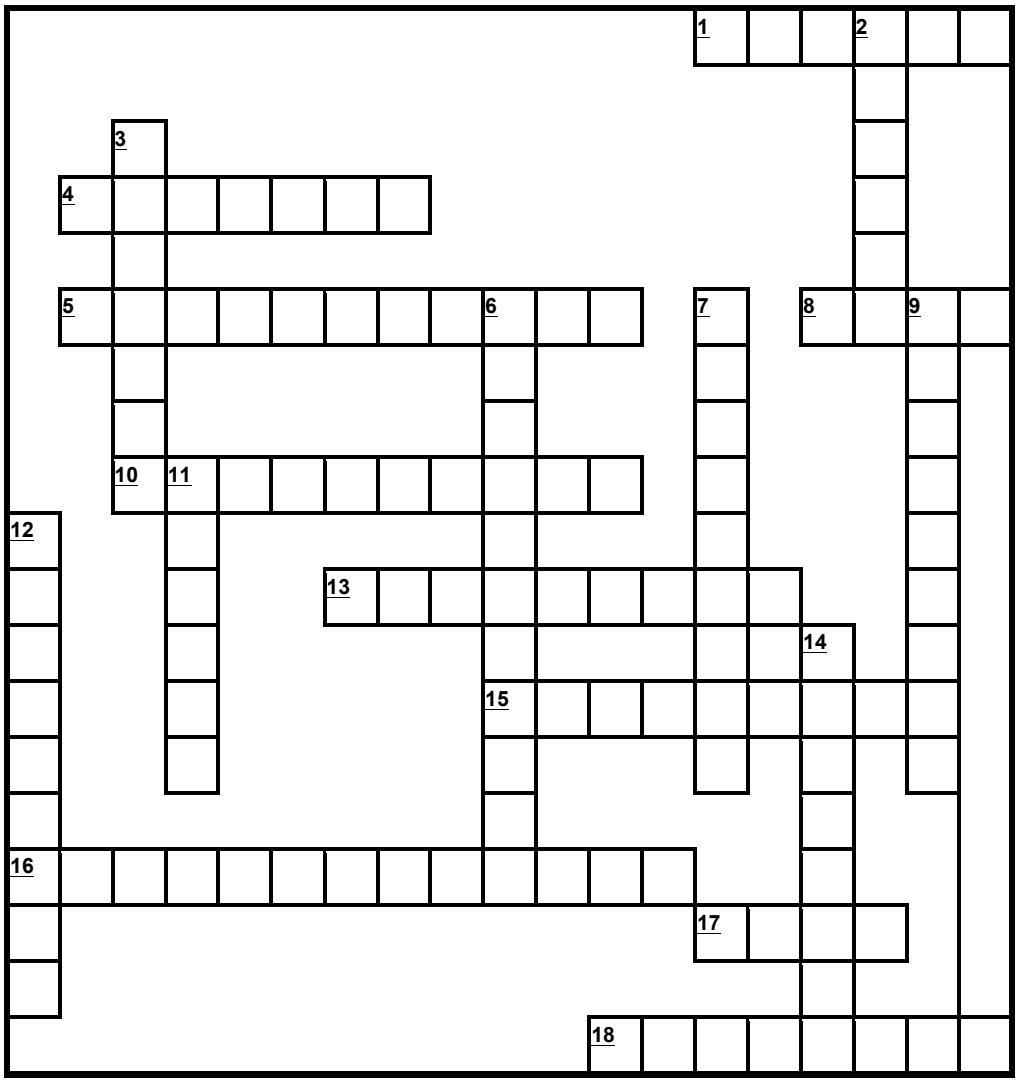
7.- Bloques de códigos que realizan una operación específica.

9.- La memoria dinámica permite variar el tamaño de memoria en tiempo de _____.

11.- Archivo utilizado para las funciones de asignación y liberación de memoria dinámica.

12.- Variable cuyo valor es una dirección de memoria.

14.- Se le llama memoria _____ a la característica del lenguaje C que permite crear cambios en el tamaño de diversas estructuras de datos, en tiempo de ejecución.



6.4. Ejercicios prácticos

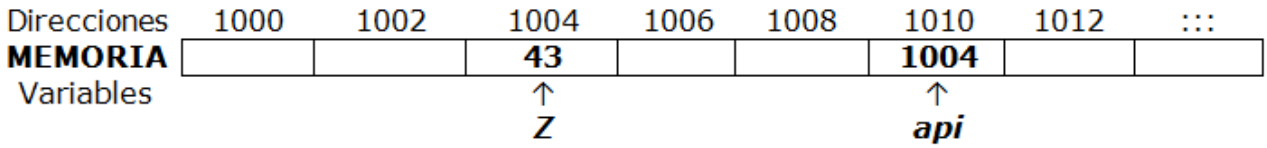
Para simplificar la revisión de este documento, se aclara que los ejercicios están alternados, es decir, en el apéndice de este apartado se presenta la solución de un ejercicio y se propone que el siguiente sea resuelto por el lector.

El significado de la nomenclatura utilizada, por ejemplo "Ejercicio 2.4.3", tiene la explicación siguiente: el primer número corresponde al apartado; el segundo corresponde al tema específico; y, el tercero, es el número de ejercicio, es decir, apartado 2, tema 4 y ejercicio o programa número 3.

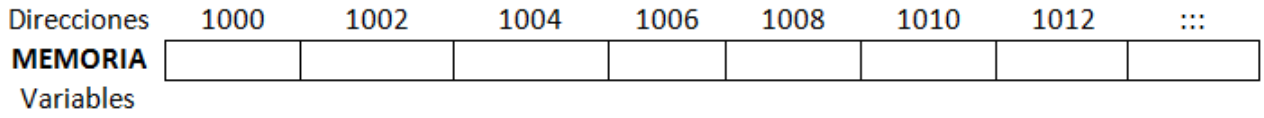
Por otra parte, hay dos tipos de prácticas, las prácticas *normales* y las de *paso a paso*. Para las prácticas normales se recomienda utilizar los ejercicios propuestos para que sean resueltos por el lector y, para las prácticas paso a paso, guiarse con los ejercicios resueltos, solo se debe seguir el procedimiento siguiente:

1. Encender el equipo de cómputo.
2. Abrir sesión del ambiente gráfico Windows.
3. Ejecutar la aplicación que se tenga instalada para editar los programas escritos en lenguaje C.
4. Transcribir el programa en cuestión.
5. Guardar el archivo que contiene el programa transcrito.
6. Compilar el programa.
7. Iniciar un proceso de depuración del programa (corrección de errores) hasta cerciorarse de que el programa cumpla con el objetivo establecido.
8. Imprimir el resultado de la ejecución o ejecuciones del programa (paso opcional).

Ejercicio 6.1.1: Realiza la declaración de la variable entera *z* y el apuntador a entero *api*, así como la asignación necesaria a la variable y al apuntador, de tal forma que ese apuntador señale a la misma dirección de la variable *z*. Presente en pantalla el valor de la variable z, pero a través del apuntador, de acuerdo al siguiente esquema gráfico:

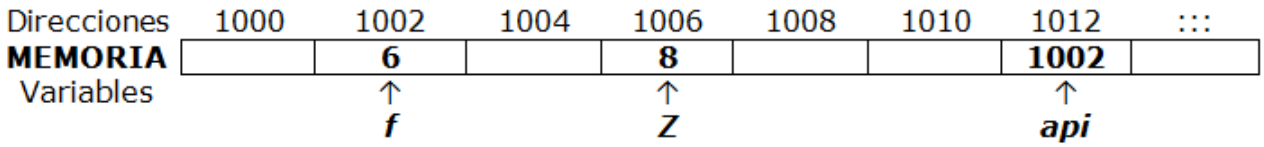


Ejercicio 6.1.2: Las líneas siguientes son un ejemplo simple de la declaración de una variable y su apuntador en una funcion(). El propósito es que se llene adecuadamente el esquema gráfico que aparece abajo, de tal forma que si la variable *VAR* guarda su contenido en la dirección 1010, pues que el apuntador *APUNTA* (almacenado en la dirección 1002) señale esa misma dirección:

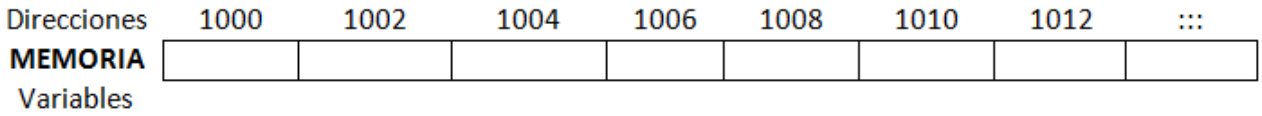


```
void funcion()
{
      int VAR, *APUNTA; /* VAR es un entero, APUNTA es un apuntador a entero */

      VAR = 34; /* Valor asignado a VAR directamente*/
      APUNTA = &VAR; /* Dirección de VAR asignada a APUNTA */

      printf("El valor de VAR es %d", *APUNTA);

      getch();/* Hace una pausa para ver el resultado */

}
```

Ejercicio 6.1.3: Realiza la declaración de variables y apuntadores, así como las asignaciones necesarias, de tal forma que el apuntador a entero *api* señale a la misma dirección de la variable entera *f*, que guarda el valor 6. Asigna el valor de la variable entera *f* a *Z*, pero indirectamente a través del apuntador *api* y, así, cambia el valor actual almacenado en la dirección 1006, de acuerdo al siguiente esquema gráfico:



Ejercicio 6.1.4: El siguiente fragmento de código es un ejemplo simple de la declaración de dos variables y un apuntador en la función main(). Suponiendo que si la variable *x* almacena su valor en la dirección 1010 y la variable TOTAL en la dirección 1002, después de que se ejecuta el código, qué valor almacena la variable TOTAL y cómo se representaría este proceso en el esquema gráfico que aparece abajo:



Listado 6.1.4

```
/* ----------------------------------------------------------------
Programa: Ejercicio6-1-4.c
Objetivo: Demostrar la declaración de apuntadores y asignación de
      valores de manera indirecta.
----------------------------------------------------------------*/
#include <stdio.h>

/* Inicializar api para que no haga referencia a algún valor */
int x, TOTAL, *api = NULL;

void main()
{
      TOTAL = 99;
      x = 10;
      api = &x;
```

```
	TOTAL = *api;

	clrscr();  /* Limpia la pantalla */

	printf("El valor final de TOTAL es %d", TOTAL);

	getch();  /* Hace una pausa para ver el resultado */

}
```

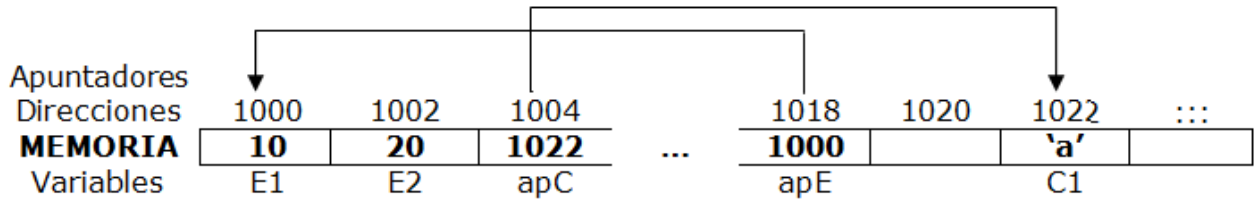
Ejercicio 6.1.5: Cuando una función recibe como argumento una dirección de una variable, se puede cambiar de manera indirecta su contenido. A esto se le conoce como paso de parámetros por referencia, tal y como se vio en el apartado 4 referente a funciones. Los apuntadores permiten simular el paso por referencia. A continuación, se presenta una función que intercambia los valores de dos lugares de memoria indicados por los apuntadores F y Z, técnica por cierto muy utilizada en los métodos de ordenación de arreglos. La tarea que se te pide es que representes de manera gráfica cómo se daría esta secuencia en la memoria, dado el fragmento de código siguiente:

Listado 6.1.5

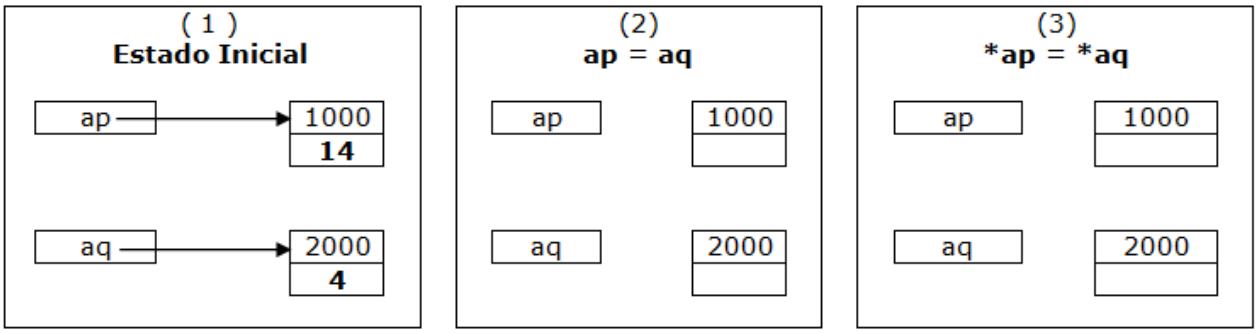
```
/* ------------------------------------------------------------------
Programa: Ejercicio6-1-5.c
Objetivo: Demostrar la declaración de apuntadores y asignación de
	valores de manera indirecta.
------------------------------------------------------------------*/
#include <stdio.h>

void intercambio(int *, int *);

void main()
{
	int x = 100, y = 200;

	clrscr();
	intercambio(&x, &y);
	printf("x = %2d, y = %2d", x, y);
	getch();
}

/* ------------------------------------------------------------------
Función: intercambio()
Objetivo: Intercambiar los valores de dos variables dadas.
Parámetros: Los apuntadores a las variables del intercambio.
Salida: Ninguno
------------------------------------------------------------------*/
void intercambio(int *F, int *Z) /* Simula el paso por referencia */
{
	int var_temp;

	var_temp = *F; /* var_temp recibe el dato contenido en la dirección de F */
	*F = *Z;       /* El contenido en la dirección de Z reemplaza el contenido
	                  en la dirección de F */
	*Z = var_temp; /* El contenido de var_temp reemplaza el contenido
```

```
                        en la dirección de Z */
}
```

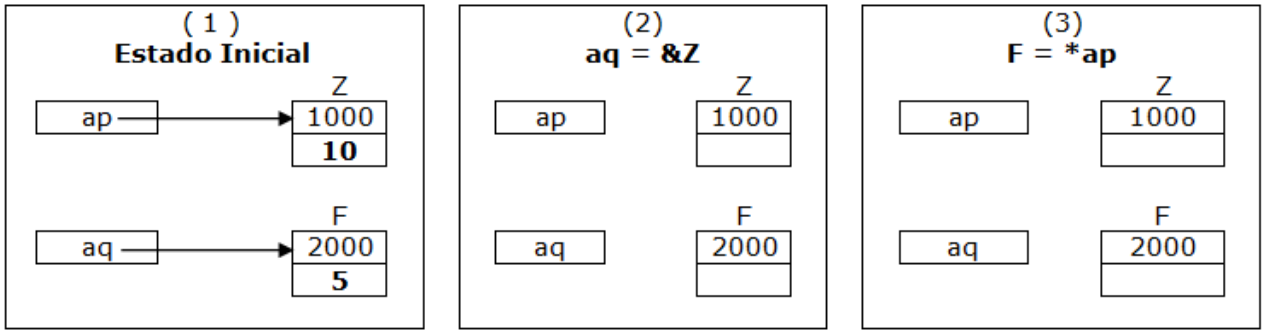
Ejercicio 6.1.6: A continuación se hace una representación de la memoria de datos que ejemplifica el comportamiento durante la ejecución de un programa. Para obtener este resultado se pide que se hagan las declaraciones, tanto de variables como de apuntadores, en un fragmento de código donde aparezcan de acuerdo a las direcciones de memoria a las que se hace referencia, los tipos apropiados de variables, apuntadores y las asignaciones correspondientes. Las direcciones de memoria aparecen en la parte superior y los nombres de las variables en la parte inferior:



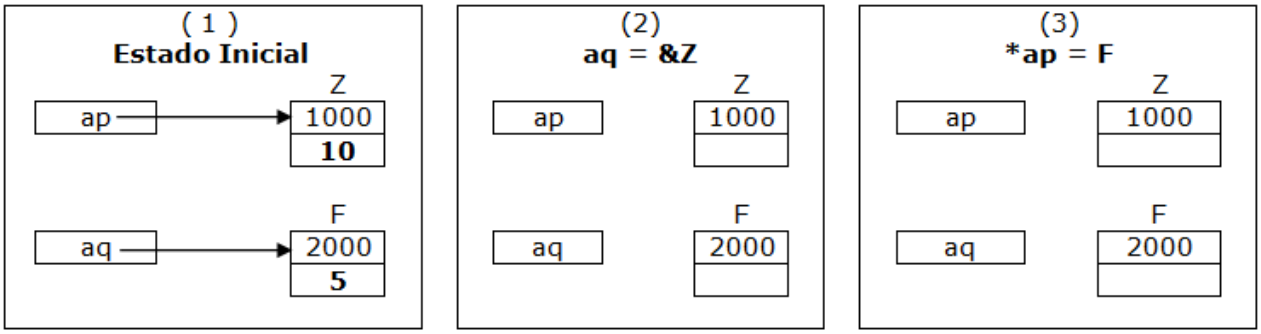
Ejercicio 6.1.7: Las asignaciones juegan un papel importante en los apuntadores, porque se obtienen diferentes resultados cuando se usa el operador de dirección que cuando se usa el operador de desreferencia. Asimismo, si la asignación se está haciendo de un apuntador a un apuntador. Veamos lo siguiente: *ap* apunta a la dirección 1000 y *aq* apunta a la dirección 2000, donde se guardan el 14 y el 4, respectivamente. Si ese es el estado inicial, representa gráficamente qué ocurrirá después de este par de operaciones (cada operación parte del estado inicial): ap = aq y *ap = *aq.



Ejercicio 6.1.8: Continuando con las asignaciones, supongamos que inicialmente ap y aq apuntan a 1000 y a 2000, respectivamente; donde están la variable Z y F que almacenan 10 y 5, tal y como aparece en el estado inicial de la gráfica. La actividad es representar gráficamente qué ocurrirá después de este par de operaciones (cada operación parte del estado inicial): aq = &Z y F = *ap. Aparte de señalar con las flechas, si cambia la dirección del apuntador, coloca dentro del recuadro los valores de las variables para saber si permanecen o cambian.



Ejercicio 6.1.9: Continuando con las asignaciones, supongamos que inicialmente ap y aq apuntan a 1000 y a 2000, respectivamente; donde están la variable Z y F que almacenan 10 y 5, tal y como aparece en el estado inicial de la gráfica. La actividad es representar gráficamente qué ocurrirá después de este par de operaciones (cada operación parte del estado inicial): aq = &Z y *ap = F. Además de señalar con las flechas, si cambia la dirección del apuntador, coloca dentro del recuadro los valores de las variables para saber si permanecen o cambian.



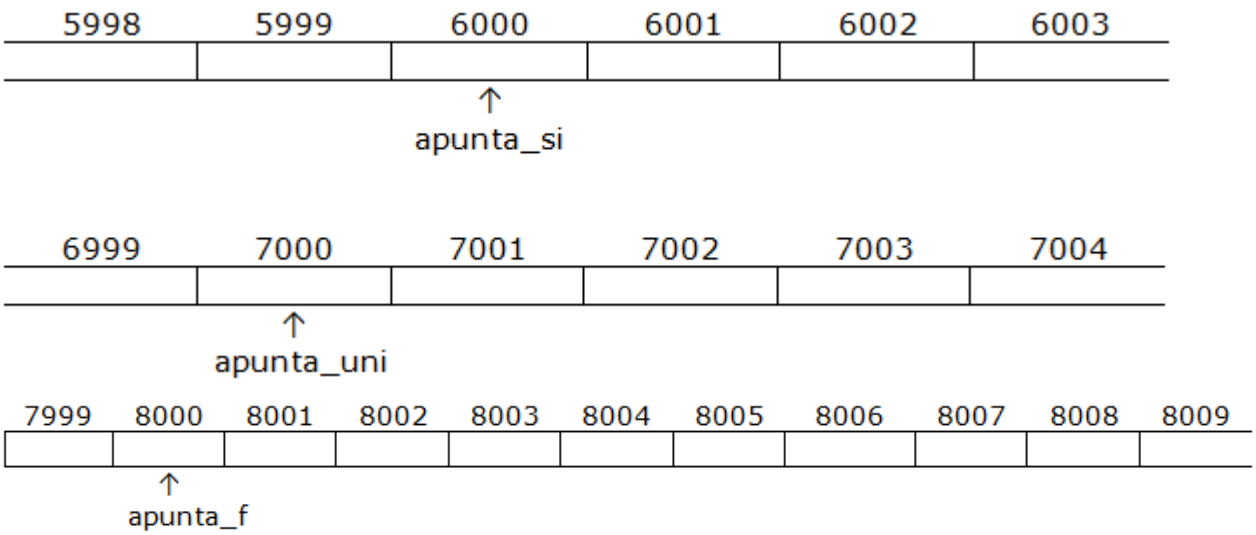
Ejercicio 6.1.10: Por los diferentes tipos de datos que existen, se sabe que algunos ocupan más o menos espacio que otros en memoria. Por ejemplo, en el caso de *char* se ocupa 1 byte, *short* ocupa 2 bytes y *long* ocupa 4. Se habla de esto debido a que las operaciones aritméticas permitidas por apuntadores son sumar y restar, pero ambas tienen un efecto diferente de acuerdo al tamaño del tipo de dato al cual apuntan. Se tienen las declaraciones siguientes de apuntadores:

```
short int     *apunta_si;
unsigned int  *apunta_uni;
float         *apunta_f;
```

Vamos a suponer que apuntan a 6000, 7000 y 8000, en ese orden. Después de realizar las operaciones siguientes:

```
apunta_si++;
apunta_uni++;
apunta_f++;
```

¿Cuál será la dirección nueva a la que apuntan? Representa tu respuesta gráficamente con el apoyo de los esquemas siguientes:



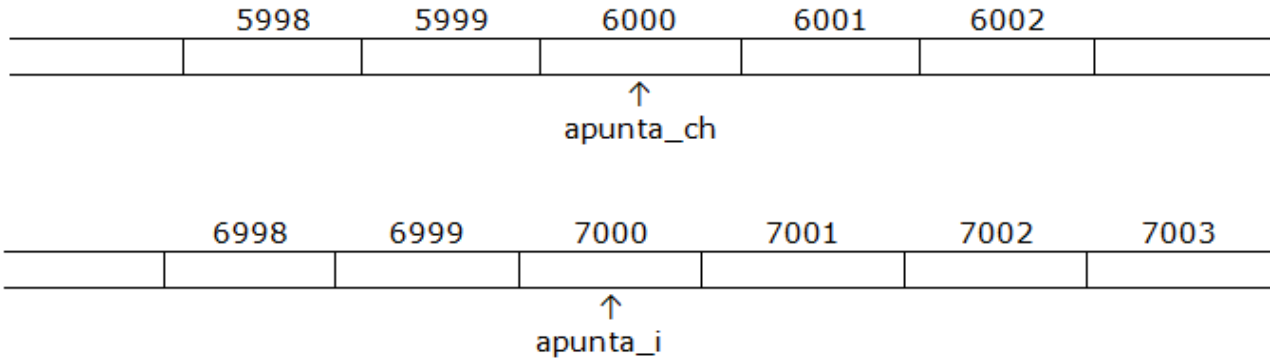
Ejercicio 6.1.11: Considerando que los apuntadores señalan de acuerdo al tamaño del tipo de dato al cual apuntan, se tiene la declaración siguiente de apuntadores:

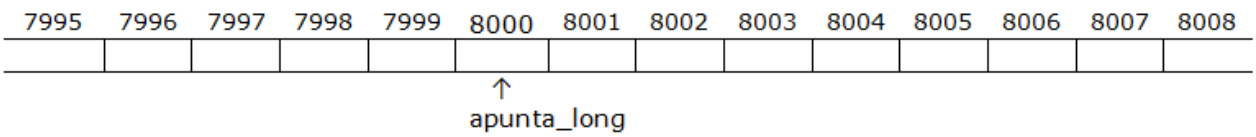
```
char      *apunta_ch;
int       *apunta_i;
long int *apunta_long;
```

Vamos a suponer que apuntan a 6000, 7000 y 8000, en ese orden. Después de realizar las operaciones siguientes:

```
apunta_ch--;
apunta_i--;
apunta_long--;
```

¿Cuál será la dirección nueva a la que apuntan? Representa tu respuesta gráficamente con el apoyo de los esquemas siguientes:





Ejercicio 6.2.1: Con el uso de apuntadores, realiza un programa que simule un traductor inglés-español, de tal forma que reciba en minúsculas las estaciones del año en español y nos devuelva su equivalente en inglés.

Ejercicio 6.2.2: Con el uso de apuntadores realiza un programa que simule un diccionario de antónimos, de tal forma que reciba en minúsculas un adjetivo y devuelva su antónimo. Las parejas a utilizar son: feliz-triste, inteligente-tonto, flaco-gordo y alto-chaparro.

Ejercicio 6.2.3: A continuación se presenta un súper ejercicio, el cual pretende que se realicen varias tareas alrededor de un tema y que cada planteamiento sea resuelto con la aplicación de apuntadores. Por lo tanto, defina las funciones siguientes y ejecútelas desde la función main().

a) **Capturar una lista de 5 a 10 alumnos y su correspondiente calificación. Al ser la primera función realiza las declaraciones necesarias.**

b) Con los datos de los alumnos del inciso A, realiza una función que calcule el porcentaje de alumnos aprobados y reprobados.

c) Habilitar una función que reciba el nombre de un alumno y que si lo encuentra regrese su calificación.

d) Implementar una función que muestre la lista completa con el nombre y calificación del alumno.

e) Copiar a un arreglo todos los alumnos cuya calificación sea mayor a la calificación mínima aprobatoria, establecida por el usuario, y listarlos en pantalla con el título "Lista de alumnos aprobados".

f) Copiar a un arreglo todos los alumnos reprobados, es decir, que su calificación sea menor o igual a 5, y listarlos en pantalla con el título "Lista de alumnos reprobados".

g) Desarrollar una función que calcule el promedio del grupo, de tal manera que muestre la lista completa de alumnos con nombre y calificación y que, en la parte final, muestre el promedio grupal.

h) Realizar un menú con el uso de la estructura condicional switch, de tal forma que el usuario pueda seleccionar a qué función quiere tener acceso, de todas las funciones desarrolladas en los incisos anteriores.

APÉNDICE

Ejercicio 1

1.- Dirección. 2.- Apuntadores. 3.- Contenido. 4.- Valor izquierdo. 5.- Nulo. 6.- Indirección. 7.- Operador de dirección. 8.- Variable. 9.- Expresiones. 10.- Valor

Ejercicio 2

1.- (f); 2.- (d); 3.- (i); 4.- (b); 5.- (j); 6.- (a); 7.- (g); 8.- (e); 9.- (c); 10.- (h)

Ejercicio 3

1.- F; 2.- V; 3.- V; 4.- F; 5.- F; 6.- F; 7.- V; 8.- V; 9.- V; 10.- F

Ejercicio 4

Sintaxis de declaración de apuntador

Tipo de dato * Nombre del apuntador;

Sintaxis de apuntador a apuntador (indirección múltiple)

Tipo de dato * * Nombre del apuntador;

Sintaxis de incremento de apuntadores

Apuntador = Apuntador + Número de incremento;

Sintaxis de la declaración de un apuntador de tipo entero y de la asignación de la dirección del primer elemento de un arreglo de tipo entero al apuntador

```
int *ptr ;
ptr = &arreglo[0];
```

Ejercicio 5

1.- (e); 2.- (d); 3.- (g); 4.- (b); 5.- (a); 6.- (h); 7.- (f); 8.- (c)

Ejercicio 6

1.- (F); 2.- (V); 3.- (F); 4.- (F); 5.- (V); 6.- (V); 7.- (F); 8.- (F); 9.- (V); 10.- (F)

Ejercicio 7

Los _**apuntadores**_ o _**punteros**_ son utilizados en el _**lenguaje**__ C para apuntar a una _**variable**__ y almacenar su _**dirección**__ en memoria, por este motivo se les llaman variables de dirección. Estos apuntadores utilizan el símbolo _**ampersand**_ (&) para indicar su dirección; indicamos al _**compilador**_ por medio de un símbolo _**asterisco**_ (*) que la variable es un apuntador.

En el valor _**izquierdo**__ de un apuntador, tenemos la dirección de la variable a la que apunta; y, en su valor _**derecho**_, su contenido. Pero si le asignamos un valor _**nulo**__, no puede _**apuntar**_ a alguna variable válida. Asímismo, varios apuntadores pueden apuntar a la misma dirección, en este caso utilizaríamos una expresión de _**asignación**__ para copiar dichos apuntadores.

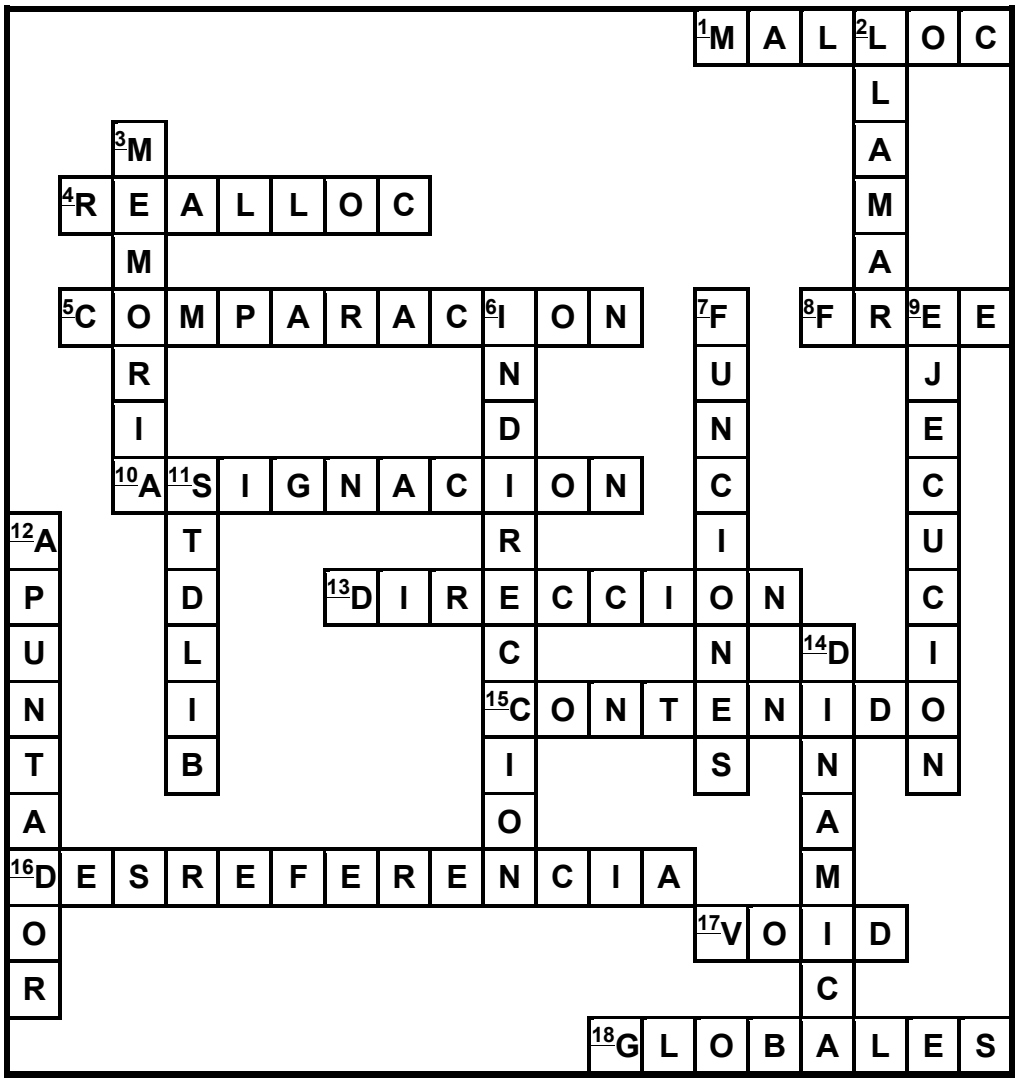
Otra de las _**operaciones**_ que se pueden realizar con apuntadores es el _**incremento**_ de la dirección a la que apuntan, al aumentar su valor. Asímismo, existe la operación opuesta, que consiste en el _**decremento**_ de la dirección a la que apuntan, al disminuirla.

También es importante decir que los apuntadores son utilizados en conjunto con los _**arreglos**_ cuando se apunta a su primer _**elemento**_ y, así, direccionar a los demás que lo conforman; tanto en una dimensión o _**vectores**_, en dos dimensiones o _**matrices**_ y en más de este número de dimensiones.

Sopa de letras

			D	I	R	E	C	C	I	O	N								
									Z			L	E	N	G	U	A	J	E
A	R	R	E	G	L	O	S		Q								O		
P									U							L			
U				A	S	T	E	R	I	S	C	O			U			C	
N			O	M					E	L	E	M	E	N	T	O		O	
T		V		P					R									M	
A		E	A	E	E				D									P	
D		C		R		R			A	P	U	N	T	A	R			I	
O		T		S	I		A			S								L	
R		O		A		A		C			I				P			A	
E		R		N			B		I			G		U				D	
S		E		D				L		O			N					O	
		S							E		N	T		A				R	
										S	E	E			C				
	A	R	R	E	G	L	O	S		R			S			I			
									O								O		
	M	A	T	R	I	C	E	S										N	
									D	E	C	R	E	M	E	N	T	O	
	D	E	R	E	C	H	O												

Crucigrama


1MALLOC
2LLAMADA
3MEMORIA
4REALLOC
5COMPARACION
6INDICE
7FUNCION
8FRACCION
9EJECUCION
10ASIGNACION
11STACK
12APUNTADOR
13DIRECCION
14DIRECCION
15CONTENIDO
16DESREFERENCIA
17VOID
18GLOBALES

Listado de ejercicios resueltos

Nota: Las soluciones a los ejercicios fueron probadas en las versiones para Windows de los compiladores Turbo C++ versión 3.0 y Dev-C++ versión 4.9.9.2.

Listado 6.1.1

```
/* ----------------------------------------------------------------
Programa: Ejercicio6-1-1.c
Objetivo: Demostrar la declaración de apuntadores y asignación de
      direcciones.
----------------------------------------------------------------*/
#include <stdio.h>

int Z, *api;

void main()
{
      Z = 43;
      api = &Z;

      clrscr();

      printf("El valor de Z es %d", *api);

      getch();

}
```

Listado 6.1.3

```
/* ----------------------------------------------------------------
Programa: Ejercicio6-1-3.c
Objetivo: Demostrar la declaración de apuntadores y asignación de
      valores de manera indirecta.
----------------------------------------------------------------*/
#include <stdio.h>

int f, Z, *api;

void main()
{
      f = 6;
      Z = 8;
      api = &f;
      Z = *api; /* Valor asignado a f indirectamente*/

      clrscr();

      printf("El nuevo valor de Z es %d", *api);

      getch();

}
```

Solución Ejercicio 6.1.5: Secuencia gráfica del intercambio.

int x = 100, y = 200;

Memoria principal

1014		
1012	**200**	y
1010	**100**	x

Área de pila

4000	aefd	←apuntador de pila
3998	aafa	
4996	a01d	
4994	alfd	

intercambio (&x, &y);

Memoria principal

Dirección		
1014		
1012	200	y
1010	100	x

Dirección	Área de pila	
4000	**1010**	F
3998	**1012**	Z
4996	a01d	var_temp
4994	alfd	←apuntador de pila

var_temp = *F; */* var_temp recibe el dato contenido en la dirección F */*

Memoria principal

Dirección			
1014			
1012	200	y	
1010	100	x	*F

Dirección	Área de pila	
4000	1010	F
3998	1012	Z
4996	**100**	var_temp
4994	alfd	←apuntador de pila

***F = *Z**; */* El contenido en 1012 se guarda en 1010 */*

Memoria principal

Dirección			
1014			
1012	200	y	*Z
1010	**200**	x	

Dirección	Área de pila	
4000	1010	F
3998	1012	Z
4996	**100**	var_temp
4994	alfd	←apuntador de pila

***Z = var_temp**;

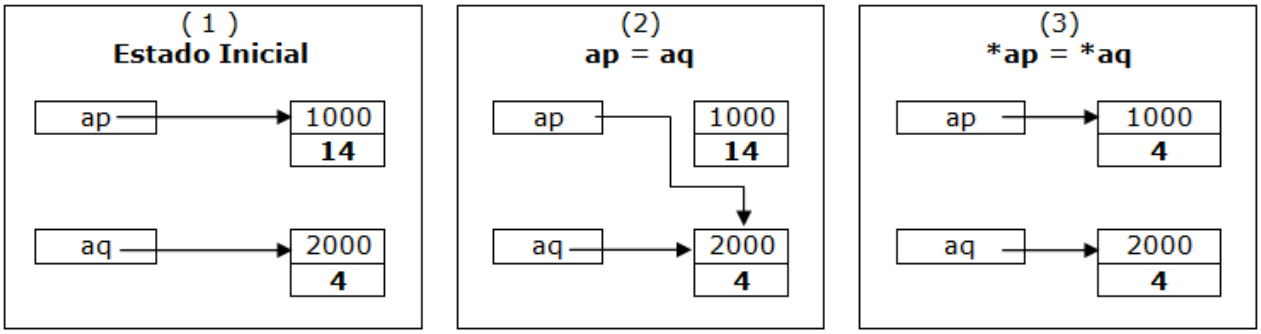
/ Colocar en la dirección 1012 el valor de var_temp y salir de la función intercambio() */*

Memoria principal

Dirección		
1014		
1012	**100**	Y
1010	200	X

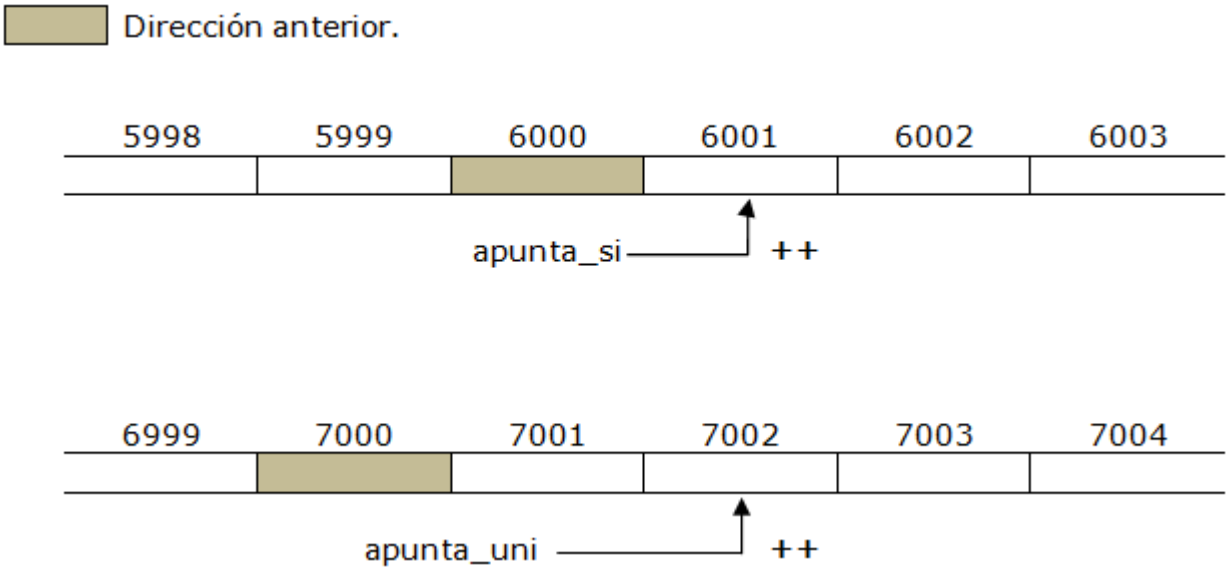
Dirección	Área de pila	
4000	1010	←apuntador de pila
3998	1012	
4996	**100**	
4994	alfd	

Solución Ejercicio 6.1.7: Secuencia gráfica de asignación.

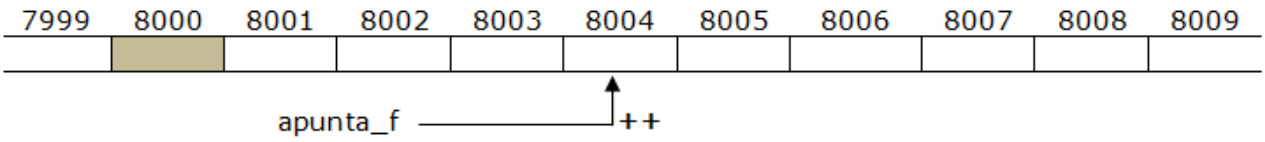


Solución ejercicio 6.1.10: Secuencia gráfica de operaciones de incremento

Explicación: El puntero **apunta_si** señala a la dirección 6001, porque el tamaño del tipo de dato al cual apunta es un **short int** (entero corto) que ocupa 1 byte de memoria.



Explicación: El puntero **apunta_uni** señala a la dirección 7002, porque el tamaño del tipo de dato al cual apunta es un **unsigned int** (entero sin signo) que ocupa 2 bytes de memoria.



Explicación: El puntero **apunta_f** señala la dirección 8004, porque el tamaño del tipo de dato al cual apunta es un **float** que ocupa 4 bytes de memoria.

Listado 6.2.1.

```
/* ----------------------------------------------------------------
Programa: Ejercicio6-2-1.c
Objetivo: Uso de apuntadores y arreglos.
----------------------------------------------------------------*/
#include <stdio.h>

#define MAX_PALABRAS 4

char esp[][10] = {"primavera","verano","otoño","invierno"};
char ingles[][10] = {"spring", "summer", "autumn", "winter"};

int busca_pal(char *, char*);

void main()
{
      char  pal_esp[80];
      int c, localiza;

      printf("\n Introduzca la palabra en español: ");
      gets(pal_esp);

      for (c = 0; c < MAX_PALABRAS; c++)
      {
            localiza = busca_pal(pal_esp, esp[c]);

            if (localiza == 0) break; /* Se encontró la palabra */

      }


      if (localiza == 0)
            printf("\n %s en inglés se dice %s", pal_esp, ingles[localiza]);
      else
            printf("\n Lo siento, esa palabra no fue encontrada");

      getch();

}

/* ----------------------------------------------------------------
Función: busca_pal()
Objetivo: Comparar dos cadenas de caracteres con el uso de apuntadores.
Parámetros: Los apuntadores a las cadenas de caracteres a comparar.
Salida: Un valor entero menor a 0 si C1 es menor que C2, 0 si son iguales y
mayor que 0 si C1 es mayor que C2.
----------------------------------------------------------------*/
int busca_pal(char *C1, char *C2)
{
      while (*C1)
            if (*C1++ - *C2++) return (*--C1 - *--C2);

      if (*C2) return -1 ;

      return 0;

}
```

Listado 6.2.3a.

```
/* ----------------------------------------------------------------------
Programa: Ejercicio6-2-3a.c
Objetivo: Uso de apuntadores y arreglos.
----------------------------------------------------------------------*/
#include <stdio.h>

#define NA 10

char  alumno[NA][100];
char  aprobados[NA][100];
float calificacion[NA];
int   tot_alum;

void captura();
void localiza();
void lista_aprobados();
void imprime_lista();
void promedio_gpo();
int  busca_alum(char *, char *);

void main()
{
      captura();
}

/* ----------------------------------------------------------------------
Función: captura()
Objetivo: Leer nombres y calificaciones para 10 alumnos o menos.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------------*/
void captura()
{
      int n = 0, i;
      char *fin;

      clrscr();

      printf ("Finalizar captura con <ctrl+z> \n\n");

      printf ("Nombre del alumno: ");
      fin = gets(alumno[n]);

      while ((n < NA) && (fin != NULL))
      {
            printf ("Calificación del alumno: ");
            scanf("%f", &calificacion[n++]);

            printf ("Nombre del alumno: ");
            fin = gets(alumno[n]);

      }

      tot_alum = n;   /* Para saber cuántos alumnos se capturaron */

}
```

Listado 6.2.3c.

```
/* ----------------------------------------------------------------
Programa: Ejercicio6-2-3c.c
Objetivo: Uso de apuntadores y arreglos.
----------------------------------------------------------------*/
#include <stdio.h>

#define NA 10

char  alumno[NA][100];
char  aprobados[NA][100];
float calificacion[NA];
int   tot_alum;

void captura();
void localiza();
void lista_aprobados();
void imprime_lista();
void promedio_gpo();
int  busca_alum(char *, char *);


void main()
{
      captura();
      localiza();

}

/* ----------------------------------------------------------------
Función: localiza()
Objetivo: Pedir el nombre de un alumno y decir su calificación.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------*/
void localiza()
{
      char nom_alum[100];
      int ind, c, ubica_alum;

      clrscr();

      printf("\n Introduzca el nombre del alumno? ");
      gets(nom_alum);

      for (c = 0; c < tot_alum; c++)
      {
            ubica_alum = busca_alum(nom_alum, alumno[c]);

            if (ubica_alum == 0) break; /* Se encontró al alumno */
      }

      if (ubica_alum == 0)
             printf("\n La calificación del alumno es: %4.2f",
                  calificacion[ubica_alum]);
      else
         printf("\n Ese alumno no se encuentra en listas");
```

```
	getch();

}

/* ---------------------------------------------------------------------
Función: busca_alum()
Objetivo: Comparar dos cadenas de caracteres con el uso de apuntadores.
Parámetros: Los apuntadores a las cadenas de caracteres a comparar.
Salida: Un valor entero menor a 0 si C1 es menor que C2, 0 si son iguales y
mayor que 0 si C1 es mayor que C2.
---------------------------------------------------------------------*/
int busca_alum(char *C1, char *C2)
{
	while (*C1)
		if (*C1++ - *C2++) return (*--C1 - * --C2);

	if (*C2) return -1;

	return 0;

}
```

Listado 6.2.3e.

```
/* ----------------------------------------------------------------------
Programa: Ejercicio6-2-3e.c
Objetivo: Uso de apuntadores y arreglos.
----------------------------------------------------------------------*/
#include <stdio.h>

#define NA 10

char  alumno[NA][100];
char  aprobados[NA][100];
float calificacion[NA];
int   tot_alum, tot_aprobados;

void captura();
void localiza();
void lista_aprobados();
void imprime_lista();
void promedio_gpo();
int  busca_alum(char *, char *);


void main()
{
      captura();
      localiza();
      lista_aprobados();
      imprime_lista();

}

/* ----------------------------------------------------------------------
Función: Lista_aprobados()
Objetivo: Pedir al usuario una calificación mínima aprobatoria y pasar
      a un arreglo únicamente a los alumnos aprobados.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------------*/
void lista_aprobados()
{
      int c, sa;
      float calif;

      clrscr();

      printf("\n Introduzca la calificación mínima aprobatoria: ");
      scanf("%f", &calif);

      sa = 0;

      for (c = 0; c < tot_alum; c++)
      {
            if (calificacion[c] >= calif)
           {
                  strcpy(aprobados[sa], alumno[c]);
                  sa++;
           }
```

```
	}

	tot_aprobados = sa;

}

/* ----------------------------------------------------------------------
Función: imprime_lista()
Objetivo: Imprimir una lista de los alumnos aprobados.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------------*/
void imprime_lista()
{
	int i;

	clrscr();

	printf("\n Lista de alumnos aprobados \n ");
	printf("\n Nombre: \n \n ");

	for (i = 0; i < tot_aprobados; i++)
	{
		printf("\n %d .- %s\n ", i + 1, aprobados[i]);
	}

	getch();

}
```

Listado 6.2.3g.

```
/* ----------------------------------------------------------------------
Programa: Ejercicio6-2-3g.c
Objetivo: Uso de apuntadores y arreglos.
----------------------------------------------------------------------*/
#include <stdio.h>

#define NA 10

char  alumno[NA][100];
char  aprobados[NA][100];
float calificacion[NA];
int   tot_alum, tot_aprobados;

void captura();
void localiza();
void lista_aprobados();
void imprime_lista();
void promedio_gpo();
int  busca_alum(char *, char *);

void main()
{
      captura();
      localiza();
      lista_aprobados();
      imprime_lista();
      promedio_gpo();
}


/* ----------------------------------------------------------------------
Función: promedio_gpo()
Objetivo: Obtener una lista del grupo y su promedio general.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------------*/
void promedio_gpo()
{
      int i;
      float prom, suma_calif;

      clrscr();
      printf("\n Lista del grupo \n ");
      printf("\n Nombre: \t \t \t Calificación\n \n ");

      suma_calif = 0;
      for (i = 0; i < tot_alum; i++) {
            printf("\n %d .- %s \t \t \t %4.2f\n ", i + 1, alumno[i],
                  calificacion[i]);
            suma_calif += calificacion[i];
      }

      prom = suma_calif / tot_alum;
      printf("\n El promedio general del grupo es: %4.2f", prom);
      getch();
}
```

Séptima parte

Estructuras de datos

7. Estructuras de datos

7.1. Tipos de datos compuestos

Ejercicio 1

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Estructuras de datos	Operador punto	Miembros	Apuntadores
Estructuras	struct	Uniones	Inicializadores
struct carro	Operador flecha		

Enunciados:

1.- A los elementos que forman las estructuras se les llama _____.

2.- Las _____ son el conjunto de datos de tipos diferentes.

3.- Los _____ ayudan a completar una estructura por medio de una lista de datos.

4.- Se usa el _____ para hacer referencia a un miembro de una estructura, mediante el uso de _____.

5.- Hacemos referencia a un miembro de una unión por medio de un _____.

6.- _____ es la palabra reservada del lenguaje C para declarar una estructura.

7.- El ejemplo _____ define la estructura con su nombre de etiqueta.

8.- Las _____ son un bloque de memoria que almacena simultáneamente datos de variables de tipos diferentes.

9.- Las _____ son un conjunto de elementos de datos.

Ejercicio 2

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis la letra correspondiente al inciso que indique la relación correcta.

a) Etiqueta de una estructura	( )	1.- Son parecidas a las estructuras pero almacenan simultáneamente varias variables en el mismo bloque de memoria.
b) Uniones	( )	2.- Permite dar otro nombre a un tipo de datos básico o definido por el usuario.
c) typedef	( )	3.- Ejemplo de declaración de una estructura con su nombre de etiqueta.
d) Variable de estructura	( )	4.- Son estructuras que contienen un miembro apuntador que señala a otra estructura de su mismo tipo.
e) carro.color	( )	5.- Término que se le da al hecho de que un apuntador señale a otro apuntador.
f) struct nodo	( )	6.- Ejemplo de acceso a un miembro de una variable de tipo estructura.
g) *main( )*	( )	7.- Su nombre define un tipo de estructuras y se emplea para declarar las variables de dicho tipo.
h) Indirección múltiple	( )	8.- Almacenan datos de tipos diferentes.
i) Estructuras autorreferenciadas	( )	9.- Las estructuras se definen antes de…
j) Estructuras	( )	10.- Variable que podemos definir después de la llave de cierre de la definición de una estructura y que contendrá los mismos miembros que esta define.

Ejercicio 3

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Los elementos de las estructuras, a diferencia de los arreglos, tienen su propio nombre. ( )
2. En el lenguaje C está descartado pasar una estructura a una función. ( )
3. Los miembros de una estructura que apuntan a otra estructura del mismo tipo, siempre tienen el nombre *autorreferenciados*. ( )
4. Es posible declarar varias estructuras del mismo tipo. ( )
5. Es imposible hacer referencia a un miembro de una unión por medio del operador flecha (**->**). ( )
6. Para el lenguaje C es inadmisible la anidación de estructuras. ( )
7. Es posible utilizar apuntadores en estructuras de datos. ( )
8. El lenguaje C impide que una unión sea un miembro de una estructura o viceversa. ( )
9. Podemos pasar elementos de una estructura por valor o por referencia a una función. ( )
10. Podemos asignar valores a los miembros de una estructura, uno por uno. ( )

Ejercicio 4

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Listas enlazadas abiertas	Operador punto	union moto	Apuntadores
Estructuras dinámicas	union	Dirección	Inicializadores
Uniones	Operador flecha		

Enunciados:

1.- Las _____son un conjunto de datos que almacenan todos sus elementos de forma simultánea en un bloque de memoria a partir de la misma dirección.

2.- Las estructuras permiten crear elementos de datos que pueden aumentar o disminuir de tamaño en tiempo de ejecución, por eso a estos elementos se les denomina _____.

3.- Los _____ ayudan a rellenar una estructura por medio de una lista de datos.

4.-Se usa el _____ para hacer referencia a un miembro de una unión mediante el uso de _____.

5.- Hacemos referencia a un miembro de una unión por medio de un _____.

6.- _____ es la palabra reservada del lenguaje C para declarar una unión.

7.- El ejemplo _____ define una unión con su etiqueta.

8.- Las uniones tienen elementos que comparten la misma _____ en memoria.

9.- Las _____ son el conjunto de elementos de datos alineados en una fila. Cada elemento del conjunto cuenta con un apuntador direccionado al miembro siguiente y, el apuntador del último elemento, tendrá un valor NULL.

Ejercicio 5

Instrucciones: Para cada una de las afirmaciones siguientes escriba la letra "V", dentro de los paréntesis en la columna de la derecha, si la afirmación es verdadera o, escriba la letra "F", si la afirmación es falsa.

1. Los miembros de una unión comparten la misma dirección de memoria. ( )
2. Es posible anidar una unión dentro de una estructura. ( )
3. Las pilas son estructuras de tamaño fijo en memoria. ( )
4. Una unión cambia dinámicamente su tamaño en memoria. ( )
5. Una lista enlazada es una serie de objetos, llamados nodos, que contienen elementos de datos y apuntadores, los cuales se direccionan al nodo siguiente, en un acomodo de fila. ( )
6. Las pilas son una estructura exclusiva del lenguaje C. ( )
7. Se inicializa el miembro de una unión solo cuando se va a utilizar. ( )
8. Las estructuras dinámicas de datos aumentan y disminuyen su tamaño en memoria durante la ejecución del programa. ( )
9. La palabra `union` se puede utilizar como nombre de variable en el lenguaje C. ( )
10. La asignación de memoria de una unión y una estructura son iguales. ( )

Ejercicio 6

Instrucciones: Relacione correctamente cada uno de los enunciados enlistados en la columna de la izquierda con uno solo de los enunciados en la columna de la derecha. Escriba dentro de los paréntesis, la letra correspondiente al inciso que indique la relación correcta.

a) LIFO	( )	1.- Es la característica de las listas donde el último nodo en entrar es el primero en salir.
b) Listas doblemente enlazadas	( )	2.- Este proceso crea un nuevo nodo con su información y se coloca un apuntador que direccione a otro nodo de la lista.
c) Nodo	( )	3.- Tipo de lista donde se inserta un nodo en un extremo y se extrae por el otro.
d) Estructuras de datos dinámicas	( )	4.- Tipo de lista que cuenta con dos apuntadores, uno apunta al nodo anterior y el otro al posterior. Pueden recorrerse en ambos sentidos.
e) Listas circulares	( )	5.- Tienen la característica principal de tener la asignación de memoria de forma variable.
f) FIFO	( )	6.- Elemento de la lista que contiene la información y uno o varios apuntadores, que direcciona a otros elementos de su mismo tipo.
g) Borrar un nodo	( )	7.- Lista en la cual se insertan y extraen los nodos por el mismo extremo.
h) Insertar un nodo	( )	8.- En este proceso se elimina el nodo y se devuelve el espacio de memoria con la función free( )
i) Pila	( )	9.- Característica de las listas donde el primer nodo en entrar es el primero en salir
j) Cola	( )	10.- Tipo de lista donde el último extremo se enlaza con el primero.

Ejercicio 7

Instrucciones: Lea con atención los enunciados siguientes y compleméntelos de forma correcta. Para tal fin, escriba una de las respuestas, que se proporcionan a continuación, en el espacio en blanco señalado por la línea dentro de cada uno de los enunciados.

Respuestas:

Diferentes	Reservada	Primeros	FIFO
Colas	Arreglos	struct	Anidación
Tamaño	LIFO	Estructura	Dinámicas
Miembros	Flecha	Nodos	Apuntadores
Uniones	Entrelazadas	Anterior	Doblemente

Enunciados:

Las variables de _____ pueden estar declaradas después de la llave de cierre que contiene el código de la estructura, sus variables internas pueden ser de _____ tipos; las estructuras están definidas por la palabra _____ _____, llamando a sus elementos _____ de la estructura, a los que hacemos referencia por medio del operador _____, además de permitir contener _____, a través de los cuales se pueden crear estructuras de datos que varían su _____ en memoria.

Las estructuras, a diferencia de los _____, permiten que sus miembros pueden ser de diferentes tipos, estando permitida la _____ de estructuras, es decir, una dentro de otra.

Las _____ son parecidas a las estructuras, con la diferencia de que almacenan de forma simultánea en el mismo bloque de memoria, a todos los miembros que las componen.

Las estructuras _____ de datos llaman a sus elementos _____, teniendo entre estas estructuras a las pilas, o estructuras _____; en las cuales los últimos nodos en entrar son los _____ en salir; y las _____ o estructuras _____, en las cuales el primer nodo en entrar es el primero en salir; ambas estructuras están _____ internamente. También existen estructuras _____ enlazadas, en las que cada nodo se conecta con el posterior y con el _____.

Sopa de letras

En la sopa de letras siguiente halle las palabras utilizadas como respuestas en el ejercicio anterior:

E	A	D	E	R	T	U	J	H	T	G	F	S	D	R	T	Y	J	K	A
N	D	A	R	F	D	S	R	T	U	F	L	M	G	I	E	D	R	J	P
T	E	E	D	I	F	E	R	E	N	T	E	S	E	Y	S	S	U	H	U
R	R	S	R	U	O	P	L	Ñ	H	G	C	D	G	T	Y	P	O	D	N
E	T	T	S	E	T	R	U	J	F	F	H	S	D	E	U	R	K	S	T
L	Y	Y	A	R	R	E	G	L	O	S	A	A	E	W	O	I	U	E	A
A	U	U	S	R	Y	S	W	I	A	S	D	X	Y	D	P	M	G	Y	D
Z	M	I	E	Y	T	E	R	F	I	F	O	E	I	O	Ñ	E	S	T	O
A	I	O	R	F	O	R	T	O	Y	G	D	S	P	Y	O	R	W	D	R
D	E	P	T	N	,Ñ	V	U	O	K	V	F	X	A	C	K	O	A	O	E
A	M	Ñ	Y	O	O	A	N	I	D	A	C	I	O	N	H	S	Q	B	S
S	B	E	U	D	R	D	V	D	S	U	V	L	S	S	E	S	D	L	D
D	R	O	O	A	E	A	O	S	E	R	A	A	W	I	S	D	I	E	F
E	O	I	P	U	O	J	G	S	Y	S	N	Z	Y	T	A	R	N	M	G
R	S	R	X	E	J	T	E	D	F	T	Y	O	K	H	T	R	A	E	T
T	A	M	A	Ñ	O	A	T	Y	U	N	I	O	N	E	S	F	M	N	J
A	D	E	R	T	Y	U	O	J	T	Y	U	J	N	B	V	F	I	T	H
U	I	H	D	S	T	R	U	C	T	B	U	I	G	R	D	Q	C	E	T
A	N	T	E	R	I	O	R	X	D	R	T	Y	G	V	H	U	A	E	R
S	T	E	V	Y	U	I	E	S	T	R	U	C	T	U	R	A	S	K	I

Crucigrama

HORIZONTALES

2.- A las estructuras _____, se les llama así por la capacidad de variar la cantidad de memoria requerida en tiempo de ejecución.

6.- Las _____ son un vector que contiene elementos llamados nodos, formado por datos y apuntadores.

7.- Los _____ permiten manipular una variable a través de su dirección en memoria.

9.- Las _____ son un tipo de lista caracterizado porque los elementos se almacenan en filas, entrando por un extremo y saliendo por el otro.

11.- Se les llama listas _____ ya que los nodos se conectan entre sí, por medio de sus apuntadores.

12.- Es la abreviación de la característica del arreglo tipo cola.

13.- En las listas _____ cada elemento apunta al siguiente y, el último, apunta a NULL.

16.- Lugar donde se ejecuta el programa, ubica cada una de sus partes por medio de su dirección.

17.- Operador que permite hacer referencia a un miembro de una unión, con ella se pueden utilizar apuntadores.

18.- Variable de un solo subíndice o una dimensión.

VERTICALES

1.- En las listas _____ el último nodo se enlaza al primero.

3.- Las _____ son un bloque de memoria donde se almacenan simultáneamente datos de tipos diferentes.

4.- Palabra reservada usada para definir las estructuras.

5.- La diferencia principal entre un arreglo y una estructura es que los miembros de esta última pueden tener un tamaño…

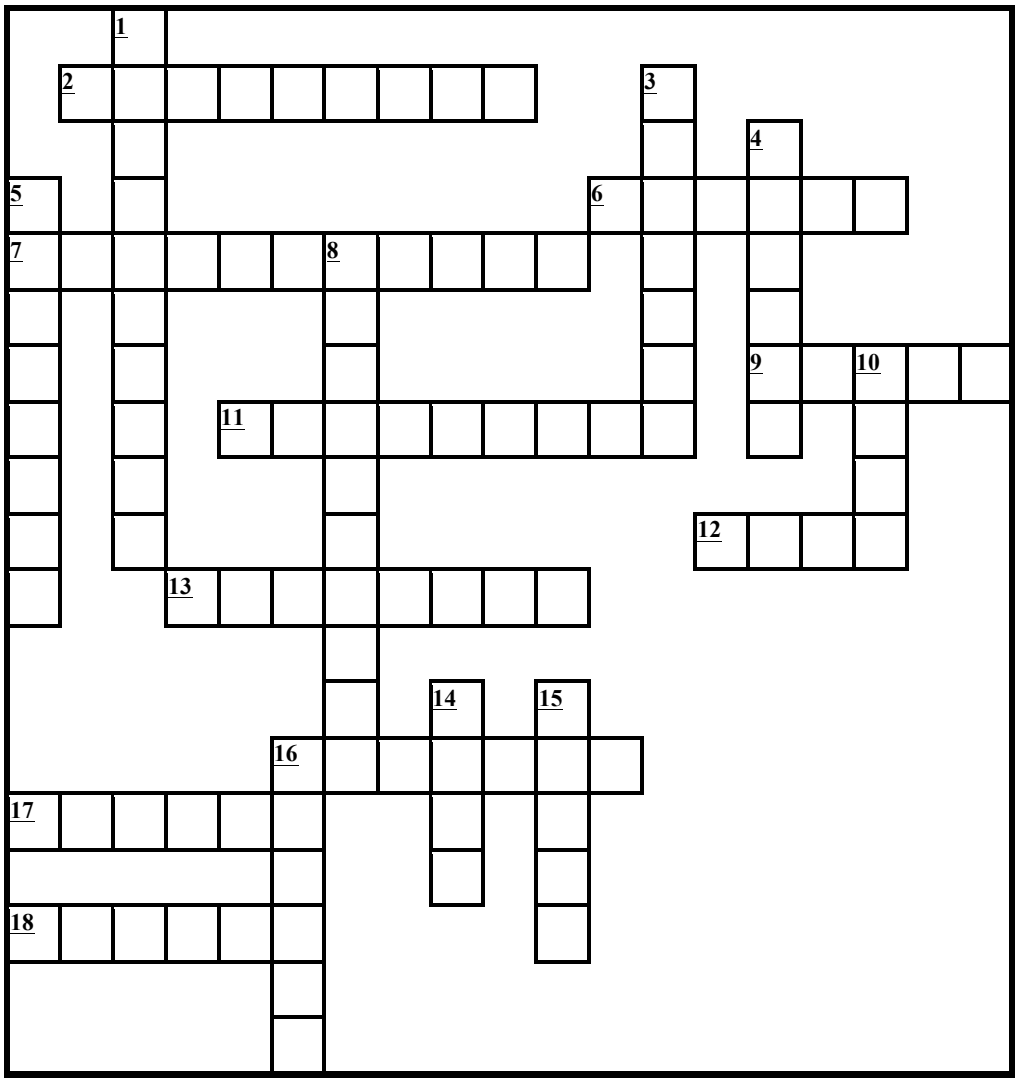
8.- Las listas _____ enlazadas tienen en sus nodos dos apuntadores, permitiendo recorrerse en ambos sentidos.

10.- Abreviatura característica de la pila.

14.- Elemento de las listas formado por los datos y uno o varios apuntadores.

15.- En las listas tipo _____ solo se permite añadir y leer en el extremo superior.

16.- Conjunto de elementos formados en filas y columnas.



7.2. Ejercicios prácticos

Para simplificar la revisión de este documento, se aclara que los ejercicios están alternados, es decir, en el apéndice de este apartado se presenta la solución de un ejercicio y se propone que el siguiente sea resuelto por el lector.

El significado de la nomenclatura utilizada, por ejemplo "Ejercicio 2.4.3", tiene la explicación siguiente: el primer número corresponde al apartado; el segundo corresponde al tema específico; y, el tercero, es el número de ejercicio, es decir, apartado 2, tema 4 y ejercicio o programa número 3.

Por otra parte, hay dos tipos de prácticas, las prácticas *normales* y las de *paso a paso*. Para las prácticas normales se recomienda utilizar los ejercicios propuestos para que sean resueltos por el lector y, para las prácticas paso a paso, guiarse con los ejercicios resueltos, solo se debe seguir el procedimiento siguiente:

1. Encender el equipo de cómputo.
2. Abrir sesión del ambiente gráfico Windows.
3. Ejecutar la aplicación que se tenga instalada para editar los programas escritos en lenguaje C.
4. Transcribir el programa en cuestión.
5. Guardar el archivo que contiene el programa transcrito.
6. Compilar el programa.
7. Iniciar un proceso de depuración del programa (corrección de errores) hasta cerciorarse de que el programa cumpla con el objetivo establecido.
8. Imprimir el resultado de la ejecución o ejecuciones del programa (paso opcional).

Estructuras y uniones

Ejercicio 7.1.1: En el ejercicio 5.3.3 se planteaba guardar el nombre y puntuación de unas concursantes en un certamen de belleza y fue resuelto utilizando dos arreglos, pues bien, lo que aquí se solicita es que se declare una estructura llamada *certamen* donde se guarden ambos elementos o componentes. Por último, declare dos variables llamadas *eliminadas* y *finalistas* del tipo estructura *certamen*.

Ejercicio 7.1.2: En el ejercicio 5.3.4 se formuló un problema sobre los concursos de *reality show* y en específico se pedía usar arreglos para guardar los nombres de los participantes y la votación. Ahora se solicita que se declare una estructura llamada *concurso* donde se guarden ambos datos. Por último declare un arreglo llamado *participantes* tipo estructura *concurso,* donde se puedan almacenar los 16 "cantantes" inscritos al concurso.

Ejercicio 7.1.3: Declare una estructura llamada *fecha* con los siguientes elementos: *Día*, de tipo entero; *mes*, de tipo entero; y, *año*, también del tipo entero. Después declare dos variables, una llamada

***f_nacimiento* y otra *f_hoy*, del tipo estructura *fecha*. Por último, asigne valores a ambas variables e imprímalas en pantalla con el formato: dd – mm – aa y dd / mm / aa, respectivamente.**

Ejercicio 7.1.4: Declare una estructura llamada *teléfono* con los siguientes elementos: *Código* del tipo entero (sirve para almacenar la clave del país), *LADA* del tipo entero (usado para almacenar la clave o lada nacional), *Número* del tipo entero (utilizado para guardar el número telefónico). Después, declare dos variables, una llamada *t_casa* y otra mi_*cel* del tipo estructura *teléfono*. Por último, asigne como valores para ambas variables dos números telefónicos, puede ser el teléfono de tu escuela y en la otra variable tu número de teléfono celular o el de algún compañero, e imprímelos en pantalla con este formato: 99 – (999) 999-9999. Donde el 9 (nueve) representa cualquier número.

Ejercicio 7.1.5: Una empresa cuenta con un grupo máximo de 10 vendedores. De ellos es necesario llevar un registro de su nombre, sueldo base, total de ventas y comisión. Con el uso de arreglos de estructuras haga las declaraciones necesarias de tal forma que el programa pida el nombre del vendedor, las ventas realizadas y calcule su comisión a raíz de un 15% sobre su total de ventas. Los datos deberán ser guardados dentro del arreglo. Por último, genere una lista donde aparezca el nombre del vendedor, sus ventas y su comisión correspondiente. Al final, informe el monto total que se requerirá para pagar la nómina generada por los vendedores. Una aclaración adicional es que el sueldo base son 30 salarios mínimos a la quincena (suponer que el salario mínimo está en $50.00 pesos).

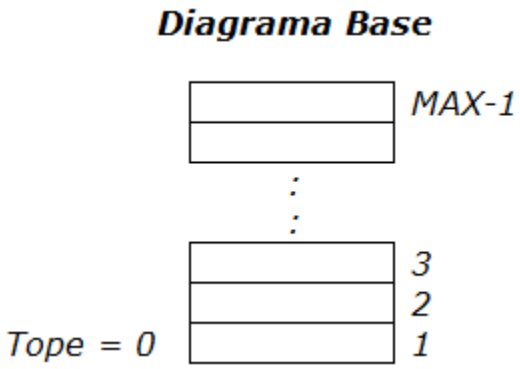
Ejercicio 7.1.6: En el ejercicio 6.2.3 inciso A se solicitaba una lista de 10 alumnos y su calificación respectiva. Intente crear una estructura con ambos componentes y usando arreglos de estructuras haga las declaraciones correspondientes de tal forma que el programa pida tanto el nombre como la calificación y se almacenen correctamente con este tipo de datos estructura. Además, en el inciso G de ese mismo ejercicio se plantea desarrollar una función que calcule el promedio del grupo, de tal manera que muestre la lista completa de alumnos con nombre y calificación y que, en la parte final, muestre el promedio grupal. Intente resolver esa función pero con el manejo de arreglos de estructuras.

Ejercicio 7.1.7: Una empresa dedicada al envío de correspondencia de recibos de cobro de servicios (como agua, luz, teléfono, entre otros) tiene problemas para hacer llegar los sobres, porque en ocasiones el domicilio que está dado de alta corresponde a un edificio con departamentos o condominios. Se pide que mediante el empleo de estructuras y uniones se cree una declaración que contemple ambos casos, es decir, cuando se trate de un domicilio particular o cuando se trate con un domicilio que tenga número interior. Los elementos sugeridos son: nombre, dirección (calle y número), colonia, ciudad, código postal y que contemple nombre del edificio y número interior en casos donde

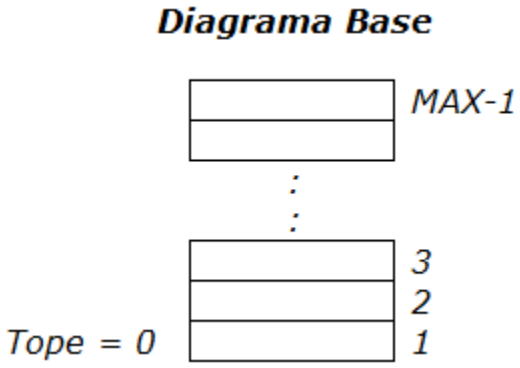
se requiera. A manera de ejemplo asignar dos domicilios, uno particular y otro para un departamento, e imprimirlos en pantalla con el formato que se maneja en los sobres.

Ejercicio 7.2.1: Mediante la ayuda de esquemas o diagramas representa gráficamente una PILA vacía.

Ejercicio 7.2.2: Con la ayuda del esquema o diagrama que aparece a continuación, representa gráficamente una PILA con los elementos siguientes: primero, segundo y tercero. Indica la posición final de la variable *Tope*.



Ejercicio 7.2.3: Con la ayuda del esquema o diagrama que aparece a continuación, representa gráficamente una PILA llena con los siguientes elementos: 111, 222, 333, … 999 e indica cómo se mueve la variable tope.



Ejercicio 7.2.4: Mediante el apoyo del esquema o diagrama del ejercicio 7.2.2, representa gráficamente la creación de una PILA, con los días de la semana.

Ejercicio 7.2.5: Mediante la ayuda de esquemas o diagramas representa gráficamente como quedaría la implementación, con una PILA, de la bandeja de entrada para mensajes de texto de un celular. La representación consiste en dos diagramas donde ejemplifiques cómo se mueve el tope después de leer y borrar los últimos 2 mensajes recibidos, a partir del diagrama base.

Diagrama Base

Tope=6	SMS-7/Ago	MAX-1
	SMS-7/Ago	5
	SMS-6/Ago	4
	SMS-21/Ju1	3
	SMS-20/Jun	2
	SMS-19/Jul	1
	SMS-19/Jun	0

Ejercicio 7.2.6: A continuación se presenta una pila con el primer cuatrimestre del año. Tu tarea es representar gráficamente cómo quedaría la implementación de una PILA, al agregar 2 meses del año para que se convierta en semestre. La representación consiste en dos diagramas donde ejemplifiques cómo se mueve el tope a partir del diagrama base.

Diagrama Base

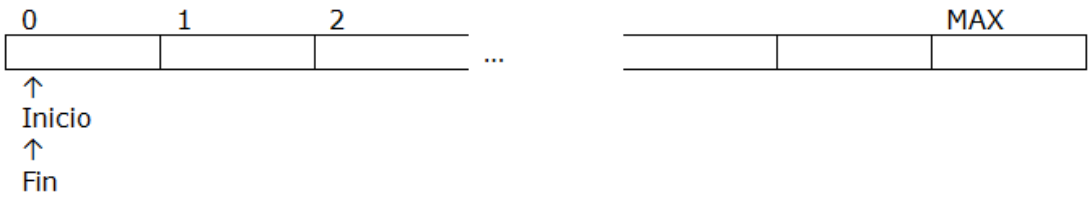
		5
		4
Tope->	Abril	3
	Marzo	2
	Febrero	1
	Enero	0

Colas

Ejercicio 7.2.7: Mediante la ayuda de esquemas o diagramas representa gráficamente una COLA vacía.

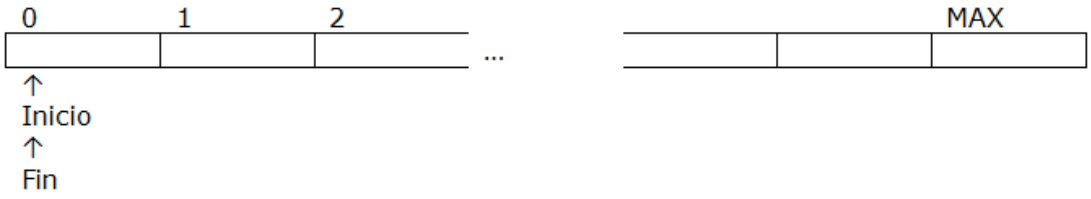
Ejercicio 7.2.8: Con la ayuda del esquema o diagrama que aparece a continuación, representa gráficamente una COLA con los elementos siguientes: primero, segundo, tercero. Indica la posición final de las variables *inicio* y *fin*.

Diagrama Base



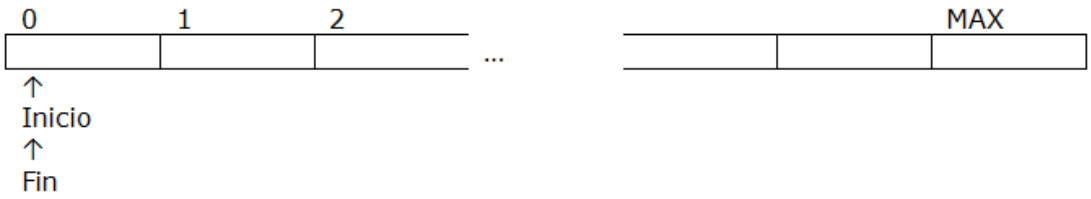
Ejercicio 7.2.9: Con la ayuda del esquema o diagrama que aparece a continuación, representa gráficamente una COLA llena con los elementos siguientes: UNO, DOS, TRES, … INFINITO. Indica cómo se mueven las variables Inicio y Fin.

Diagrama Base



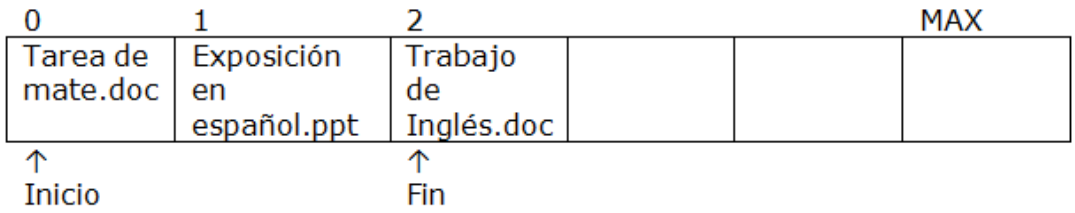
Ejercicio 7.2.10: Con la ayuda del esquema o diagrama que aparece a continuación, representa gráficamente una COLA que tenga como elementos los nombres de las estaciones del año. Indica la posición final de las variables *inicio* y *fin*.

Diagrama Base



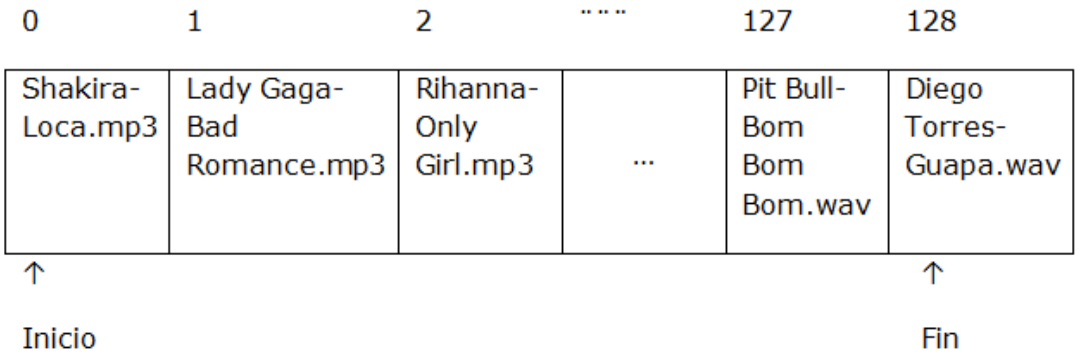
Ejercicio 7.2.11: Mediante la ayuda de esquemas o diagramas representa gráficamente cómo quedaría una COLA de impresión, la cual tiene tres trabajos de impresión y se envía un archivo más llamado *gráfica de física.xls*. La representación consiste en dos diagramas donde ejemplifiques cómo se mueve el tope a partir del diagrama base.

Diagrama Base



Ejercicio 7.2.12: A continuación se representa, mediante una cola, el reproductor Windows Media con una lista de reproducción. Tu tarea es representar gráficamente cómo quedaría la implementación de una COLA, después de reproducirse 2 canciones. La representación consiste en dos diagramas donde ejemplifiques cómo se mueve el *inicio* y *fin* a partir del diagrama base.

Diagrama Base

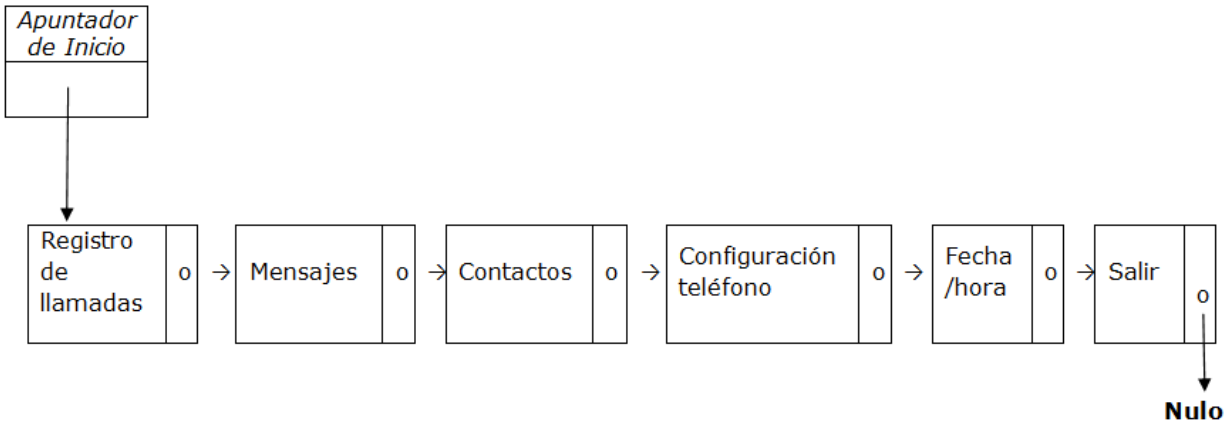


Listas

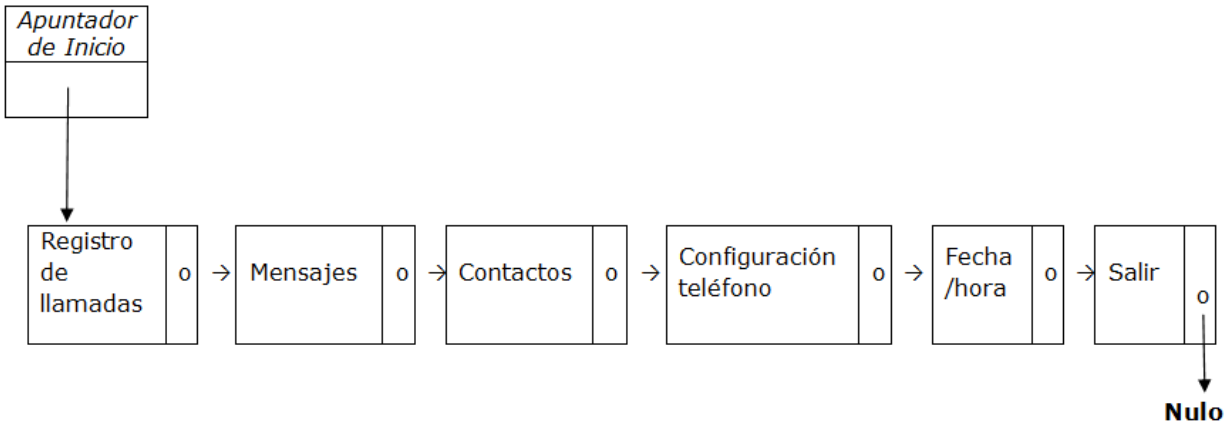
Ejercicio 7.2.13: Mediante la ayuda de esquemas o diagramas representa gráficamente los elementos necesarios para construir una lista.

Ejercicio 7.2.14: En un inicio los celulares contaban con menús y listas de opciones para su manejo. El inconveniente que presentaban era que si se requería la última opción, era necesario recorrer la lista completa. Si, un momento después, se requería utilizar la primera opción, era necesario recorrer las opciones a la inversa hasta llegar a la primera. El planteamiento es: ¿cómo se resolvió esa desventaja,

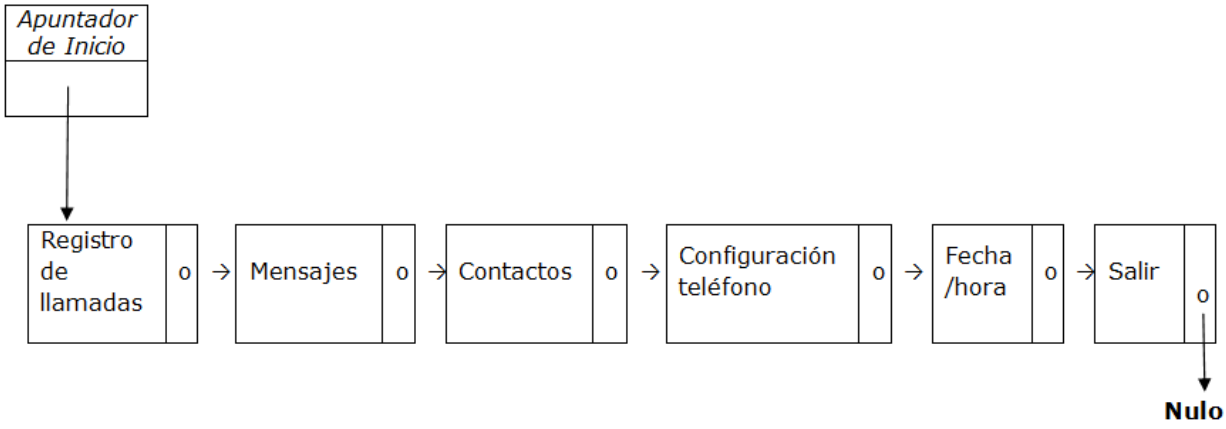
de tal manera que se pudiera pasar de la última opción a la primera? En concreto, aquí se supone que en inicio los menús los trabajaban con listas enlazadas, entonces ¿cómo modificarías el diagrama siguiente de una lista enlazada para que se volviera una lista circular?



Ejercicio 7.2.15: Conforme avanzó la tecnología, los celulares reorganizaron sus menús y algunas opciones las reacomodaron, tratando de simular ese avance. Representa mediante una lista enlazada cómo suprimir la opción Fecha/hora, pensando en que se agregó dentro de la configuración del teléfono.



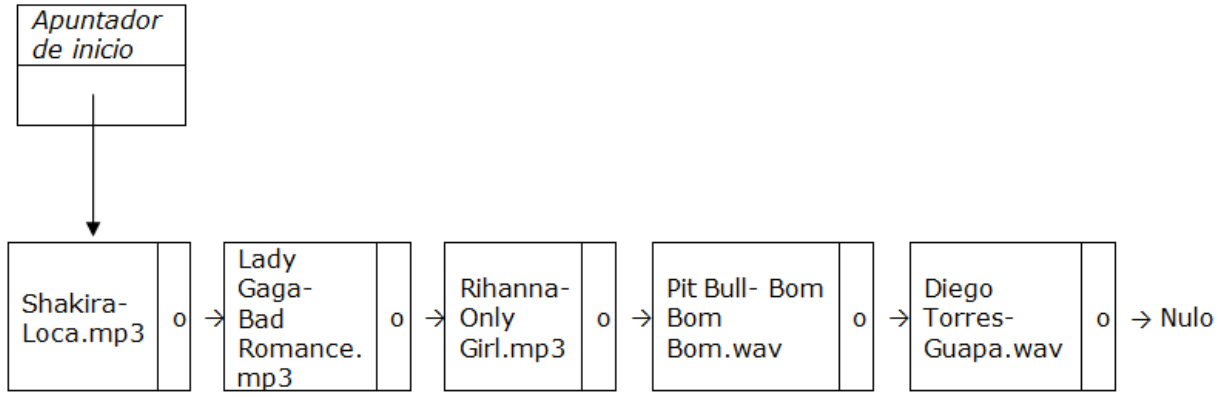
Ejercicio 7.2.16: En ese mismo desarrollo tecnológico, los celulares fueron incorporando funciones que les permitían ser más útiles. Representa en otro esquema de lista enlazada, de qué manera se incorporaría la opción *Agenda* entre las opciones de *Contactos* y *Configuración teléfono*.



Ejercicio 7.2.17: Mediante la ayuda de esquemas o diagramas representa gráficamente los elementos necesarios para construir una lista doblemente ligada.

Ejercicio 7.2.18: De los últimos obstáculos que se presentaron en cuanto a la incorporación de funciones para celulares fueron los reproductores de música, ya que si usaban listas enlazadas simples o circulares, solo se podía avanzar en un sentido, es decir, escuchar la canción siguiente. Esto servía para ver imágenes, pero no para regresar a escuchar la canción anterior. Modifica el esquema que se te presenta de tal forma que se pueda tocar tanto la canción siguiente como la melodía previa.

Diagrama Base: *Lista doblemente enlazada.*



Ejercicio 7.2.19: Agenda.- El último ejercicio presentado en esta obra consiste en la implementación, como un *súper ejercicio*, de una agenda para administración de contactos. Se pide, también, que la información perteneciente a dichos contactos se administre mediante una lista doblemente ligada (cuya implementación también deberá desarrollarse), manejando memoria asignada dinámicamente. Para ello las tareas a realizar son:

a) Presentar un menú en pantalla que permita elegir entre las acciones siguientes a realizar sobre la agenda:
   1. Altas.- Agregar un contacto a la agenda.
   2. Bajas.- Eliminar un contacto de la agenda.
   3. Consultas.- Consultar la información de un contacto específico.
   4. Listado.- Listar en pantalla la información de todos los contactos en la agenda.
   5. Salir.- Finalizar el programa.

   - No se maneja una opción de modificación a la información de los contactos en la agenda, dado que dicha operación se puede substituir con una acción de baja y luego una de alta.

b) La información a manejar para cada contacto consiste en los datos particulares siguientes:
   1. Nombre(s) de pila – Cadena de 31 caracteres como máximo, incluyendo el '\0'.
   2. Apellido(s) – Cadena de 31 caracteres como máximo, incluyendo el '\0'.
   3. Número de teléfono celular – Cadena de 16 caracteres como máximo, incluyendo el '\0'.
   4. Correo electrónico – Cadena de 31 caracteres como máximo, incluyendo el '\0'.

c) Los contactos deberán insertarse en la lista doblemente ligada en orden ascendente, con respecto al nombre completo de cada contacto (empezando por los apellidos y luego los nombres de pila).

d) No se permite la inserción de un contacto duplicado. Además, para las operaciones de eliminación y consulta individual de un contacto, se deberá indicar si éste no existiera aún en la agenda.

e) Bajo ninguna circunstancia debe olvidar liberar la memoria dinámica asignada durante la ejecución del programa (llamando a la función *free()*, como complemento de la función *malloc()*).

f) Por su parte, la implementación de la lista doblemente ligada deberá contar con, al menos, las operaciones de inserción (en forma ordenada y ascendente), eliminación y búsqueda de un elemento de la lista.

g) Maneje de forma genérica el contenido de cada nodo, es decir, los campos de la estructura autorreferenciada de cada nodo deben consistir en un apuntador void al contenido y los apuntadores al nodo siguiente y al nodo previo. Así mismo, el manejo del contenido en los nodos también deberá emplear asignación dinámica de memoria.

APÉNDICE

Ejercicio 1

1.- Miembros. 2.- Estructuras. 3.- Inicializadores. 4.- Operador flecha / Apuntadores. 5.- Operador punto. 6.- struct. 7.- struct carro. 8.- Uniones. 9.- Estructuras de datos.

Ejercicio 2

1.- (b); 2.- (c); 3.- (f); 4.- (i); 5.- (h); 6.- (e); 7.- (a); 8.- (j); 9.-(g); 10.-(d)

Ejercicio 3

1.- (V); 2.- (F); 3.- (F); 4.- (V); 5.- (F); 6.- (F); 7.- (V); 8.- (F); 9.- (V); 10.- (V)

Ejercicio 4

1.- Uniones. 2.- Estructuras dinámicas. 3.- Inicializadores. 4.- Operador flecha / Apuntadores. 5.- Operador punto. 6.- union. 7.- union moto. 8.- Dirección. 9.- Listas enlazadas abiertas.

Ejercicio 5

1.- (V); 2.- (V); 3.- (F); 4.- (F); 5.- (V); 6.- (F); 7.- (V); 8.- (V); 9.- (F); 10.- (F)

Ejercicio 6

1.- (a); 2.- (h); 3.- (j); 4.- (b); 5.- (d); 6.- (c); 7.- (i); 8.- (g); 9.- (f); 10.- (e)

Ejercicio 7

Las variables de **_estructura__** pueden estar declaradas después del paréntesis que contiene el código de la estructura, sus variables internas pueden ser de **_diferentes_** tipos; las estructuras están definidas por la palabra **_reservada__** **_struct_**, llamando a sus elementos **_miembros__** de la estructura, a los que hacemos referencia por medio del operador **_flecha_**, además de permitir contener **_apuntadores_**, a través de los cuales se pueden crear estructuras de datos que varían su **_tamaño_** en memoria.

Las estructuras, a diferencia de los **_arreglos_**, permiten que sus miembros puedan ser de diferentes tipos, estando permitida la **_anidación_** de estructuras, es decir, una dentro de otra.

Las **_uniones__** son parecidas a las estructuras, con la diferencia de que almacenan de forma simultánea en el mismo bloque de memoria, a todos los miembros que las componen.

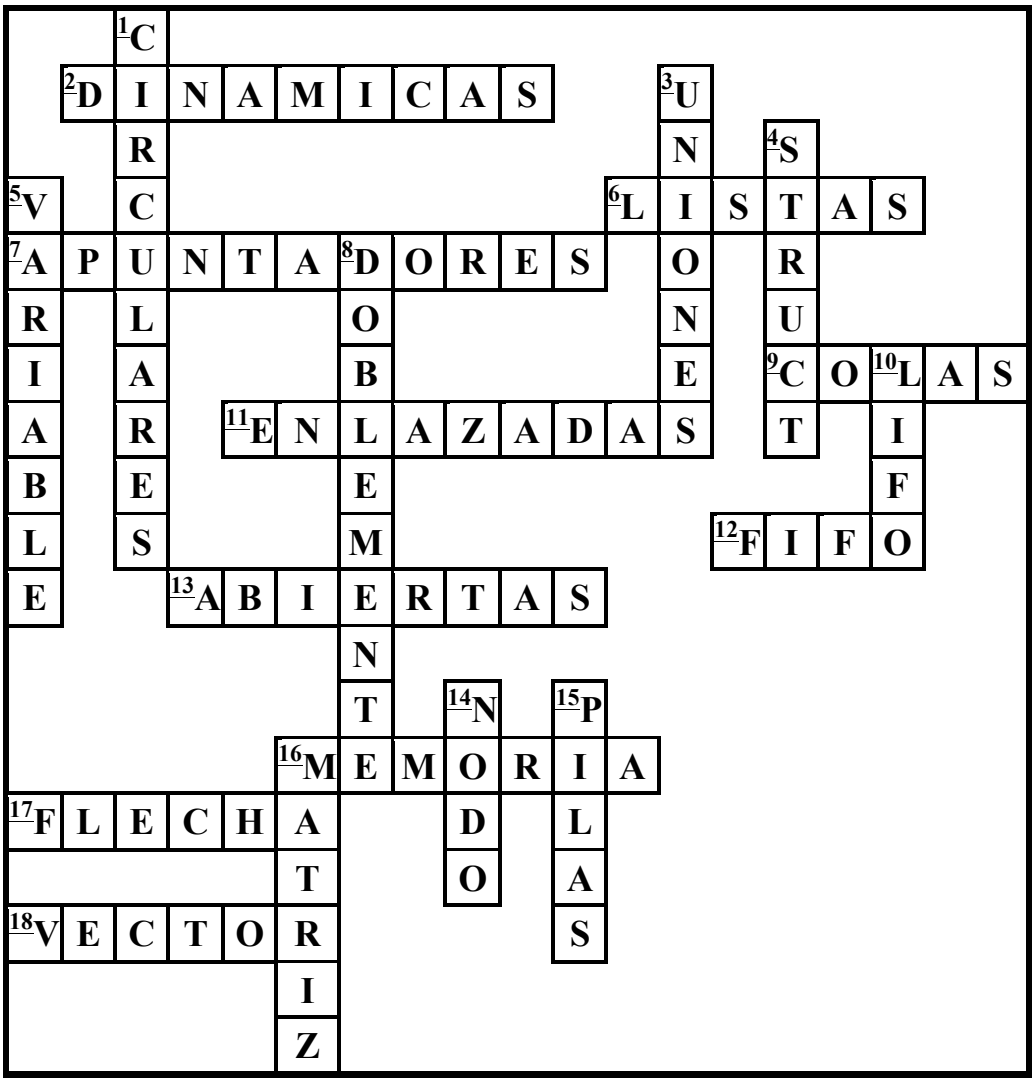
Las estructuras **_dinámicas_** de datos llaman a sus elementos **_nodos_**, teniendo entre estas estructuras a las pilas, o estructuras **_LIFO_**; en las cuales los últimos nodos en entrar son los **_primeros_** en salir; y las **_colas_** o estructuras **_FIFO_**, en las cuales el primer nodo en entrar es el primero en salir; ambas estructuras están **_entrelazadas_** internamente. También existen

estructuras _**doblemente**_ enlazadas, en las que cada nodo se conecta con el posterior y con el _**anterior**_.

Sopa de letras

E											F							A
N											L							P
T			D	I	F	E	R	E	N	T	E	S						U
R											C					P		N
E						R					H					R		T
L			A	R	R	E	G	L	O	S	A					I		A
A						S		I								M		D
Z	M					E		F	I	F	O					E		O
A	I					R		O								R	D	R
D	E			N		V								C		O	O	E
A	M				O	A	N	I	D	A	C	I	O	N		S	B	S
S	B					D						L					D	L	
	R					A	O				A						I	E	
	O							S		S							N	M	
	S																A	E	
T	A	M	A	Ñ	O				U	N	I	O	N	E	S		M	N	
																	I	T	
				S	T	R	U	C	T								C	E	
A	N	T	E	R	I	O	R										A		
							E	S	T	R	U	C	T	U	R	A	S		

Crucigrama


1C
2DINAMICAS
3UNA
4S
5VARIABLES
6LISTAS
7APUNTADORES
8DOBLE
9COLAS
10LIFO
11ENLAZADAS
12FIFO
13ABIERTAS
14NODO
15PILAS
16MEMORIA
17FLECHA
18VECTORIZ

Listado de Ejercicios RESUELTOS

Nota: Las soluciones a los ejercicios fueron probadas en las versiones para Windows de los compiladores Turbo C++ versión 3.0 y Dev-C++ versión 4.9.9.2.

Listado 7.1.1.

```
/* ----------------------------------------------------------------
Programa: Ejercicio7-1-1.c
Objetivo: Declaración de estructuras.
----------------------------------------------------------------*/
#include <stdio.h>

struct certamen
{
      char nombre[30];
      float puntuacion;
};

struct certamen eliminadas, finalistas;
```

Listado 7.1.3.

```
/* ----------------------------------------------------------------
Programa: Ejercicio7-1-3.c
Objetivo: Declaración y asignación de datos a estructuras.
----------------------------------------------------------------*/
#include <stdio.h>

struct fecha
{
      int dia;
      int mes;
      int aho;
};

struct fecha f_nacimiento, f_hoy;

void main()
{
      f_nacimiento.aho = 1968;
      f_nacimiento.mes = 6;
      f_nacimiento.dia = 19;
      f_hoy.dia = 04;
      f_hoy.mes = 8;
      f_hoy.aho = 2011;

      clrscr();

      printf(" Tu fecha de nacimiento es: %d / %d / %d", f_nacimiento.dia,
            f_nacimiento.mes, f_nacimiento.aho);
      printf("\n La fecha de hoy es: %d - %d - %d", f_hoy.dia,  f_hoy.mes,
            f_hoy.aho);

      getch();

}
```

Listado 7.1.5

```
/* ----------------------------------------------------------------
Programa: Ejercicio7-1-5.c
Objetivo: Usar arreglos de estructuras.
----------------------------------------------------------------*/
#include <stdio.h>

#define NV 10
#define SM 50
#define PC 15

void captura();
void lista_vendedores ();

struct vendedor
{
      char  nombre[100];
      int   sdo_base;
      float tot_vtas;
      float comision;
};

struct vendedor vendedores[NV];

void main()
{
      captura();
      lista_vendedores();
}

/* ----------------------------------------------------------------
Función: captura()
Objetivo: Leer nombres, ventas, sueldo base y comisión para NV
      vendedores o menos.
Parámetros: Ninguno.
Salida: Ninguna.
----------------------------------------------------------------*/
void captura()
{
      struct vendedor aux;
      int n = 0, i;
      char *fin;

      clrscr();

      printf ("\n Finalizar captura con <ctrl+z> \n\n");
      printf ("\n Nombre del vendedor: ");

      fin = gets(aux.nombre);

      while ((n < NV) && (fin != NULL))
      {
            printf ("\n Total de ventas: ");
```

```
            scanf("%f", &aux.tot_vtas);

            aux.comision = aux.tot_vtas * PC / 100.0;
            aux.sdo_base = 30 * SM;
            vendedores[n] = aux;
            n++;

            printf ("\n Nombre del vendedor: ");
            fin = gets(aux.nombre);
      }
}

/* ---------------------------------------------------------------------
Función: lista_vendedores()
Objetivo: Presentar en pantalla una lista de vendedores y su nómina.
Parámetros: Ninguno.
Salida: Ninguna.
---------------------------------------------------------------------*/
void lista_vendedores ()
{
      int i;
      float sueldos = 0.0;

      printf("\n Nomina de los vendedores \n ");
      printf("\n Nombre: \t \t \t Sueldo \t Comisión");


      for (i = 0; i < NV; i++)
      {
            printf("\n %s \t \t \t \t  %d \t \t %8.2f", vendedores[i].nombre,
                  vendedores[i].sdo_base, vendedores[i].comision);

            sueldos += vendedores[i].sdo_base + vendedores[i].comision;
      }

      printf("\n La nómina total de los vendedores es de $%8.2f", sueldos);

      getch();

}
```

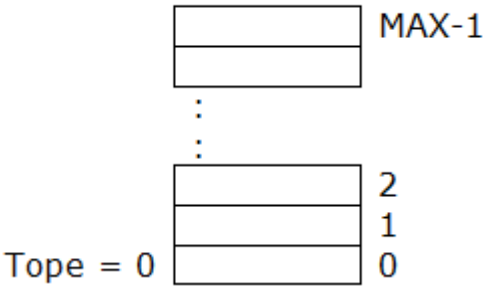
Listado 7.1.7

```
/* ----------------------------------------------------------------
Programa: Ejercicio7-1-7.c
Objetivo: Utilizar estructuras, uniones y su anidación.
----------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>

#define PARTICULAR   1
#define DEPARTAMENTO 2

struct dom_particular
{
      char direccion[30];
      char colonia[30];
      char ciudad[30];
      long int cp;
};

struct dom_departamento
{
      char nom_edificio[30];
      int  num_depto;
};

union domicilio
{
      struct dom_particular dp;
      struct dom_departamento dd;
};

struct contacto
{
      char nombre[30];
      int tipo_dom;
      union domicilio dom;
};


void imprime(struct contacto c);


void main()
{
      struct contacto gerente, presidente;

      /* Domicilio particular */
      strncpy(gerente.nombre, "Filho Enrique Borjas", 30);
      gerente.tipo_dom = PARTICULAR;
      strncpy(gerente.dom.dp.direccion, "Niños Héroes N° 35", 30);
      strncpy(gerente.dom.dp.colonia, "Chapultepec Norte", 30);
      strncpy(gerente.dom.dp.ciudad, "Morelia, Michoacán", 30);
      gerente.dom.dp.cp = 58270;
```

```
	/* Domicilio de departamento */
	strncpy(presidente.nombre, "Luis Gerardo Zavala", 30);
	presidente.tipo_dom = DEPARTAMENTO;
	strncpy(presidente.dom.dd.nom_edificio, "Torre G", 20);
	presidente.dom.dd.num_depto = 43;

	clrscr();

	imprime(gerente);
	printf("\n\n");
	imprime(presidente);

	getch();
}

/* ---------------------------------------------------------------------
Función: imprime()
Objetivo: Imprimir el domicilio de un contacto, ya sea particular o
	departamento.
Parámetros: Los datos del contacto cuyo domicilio se imprimirá.
Salida: Ninguna.
---------------------------------------------------------------------*/
void imprime(struct contacto c)
{
	printf(" %s ", c.nombre);

	if (c.tipo_dom == PARTICULAR)
	{
		printf("\n %s ", c.dom.dp.direccion);
		printf("\n %s ", c.dom.dp.colonia);
		printf("\n %li %s \n", c.dom.dp.cp, c.dom.dp.ciudad);
	}
	else
		printf("\n %s - Departamento %d \n",
			c.dom.dd.nom_edificio, c.dom.dd.num_depto);
}
```

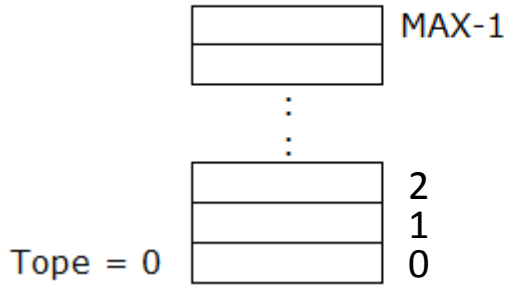
Pilas

Solución ejercicio 7.2.1:

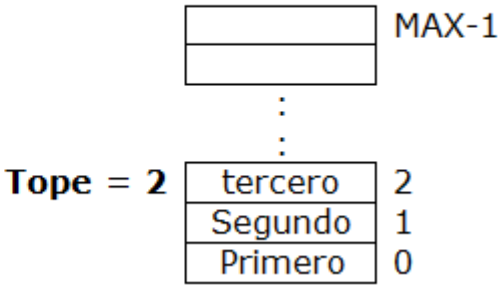


Solución ejercicio 7.2.2:

Diagrama Base

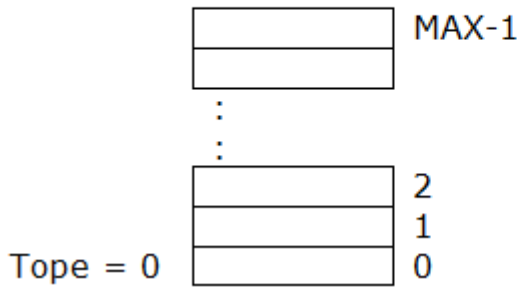


Solución:

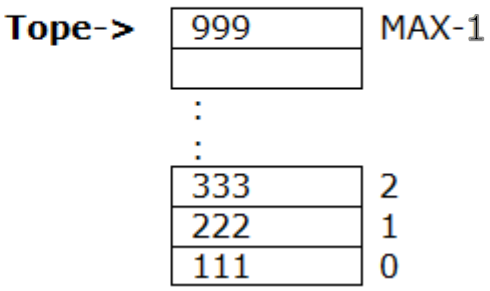


Solución ejercicio 7.2.3

Diagrama Base



Solución:



Solución ejercicio 7.2.5:

Solución:

Diagrama Base

Tope=6	SMS-7/Ago	MAX-1
	SMS-7/Ago	5
	SMS-6/Ago	4
	SMS-21/Ju1	3
	SMS-20/Jun	2
	SMS-19/Jul	1
	SMS-19/Jun	0

Diagrama 1

	SMS-7/Ago	MAX-1
Tope=5	SMS-7/Ago	5
	SMS-6/Ago	4
	SMS-21/Ju1	3
	SMS-20/Jun	2
	SMS-19/Jul	1
	SMS-19/Jun	0

Diagrama 2

	SMS-7/Ago	MAX-1
	SMS-7/Ago	5
Tope=4	SMS-6/Ago	4
	SMS-21/Jul	3
	SMS-20/Jul	2
	SMS-19/Jul	1
	SMS-19/Jul	0

Solución ejercicio 7.2.6:

Solución:

Diagrama Base

		5
		4
Tope->	Abril	3
	Marzo	2
	Febrero	1
	Enero	0

Diagrama 1

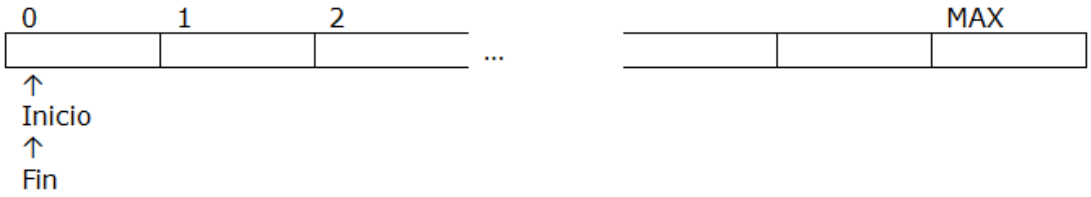
		5
Tope->	**Mayo**	4
	Abril	3
	Marzo	2
	Febrero	1
	Enero	0

Diagrama 2

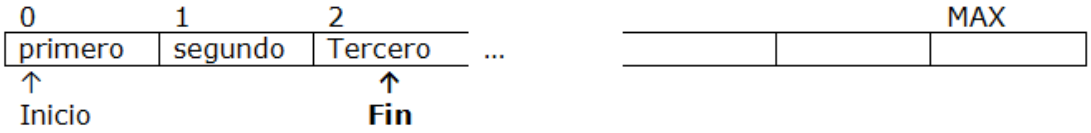
Tope->	**Junio**	5
	Mayo	4
	Abril	3
	Marzo	2
	Febrero	1
	Enero	0

Colas

Solución ejercicio 7.2.7:

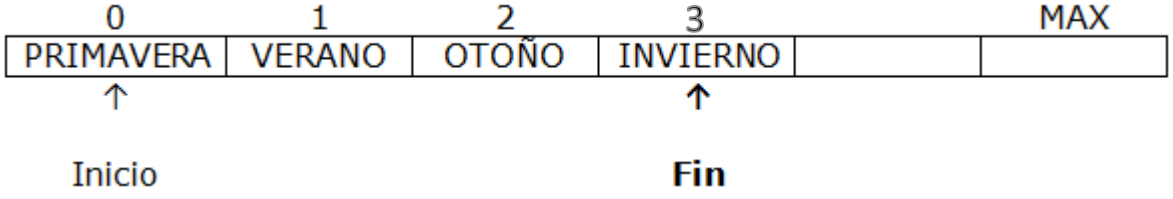


Solución ejercicio 7.2.8**:**

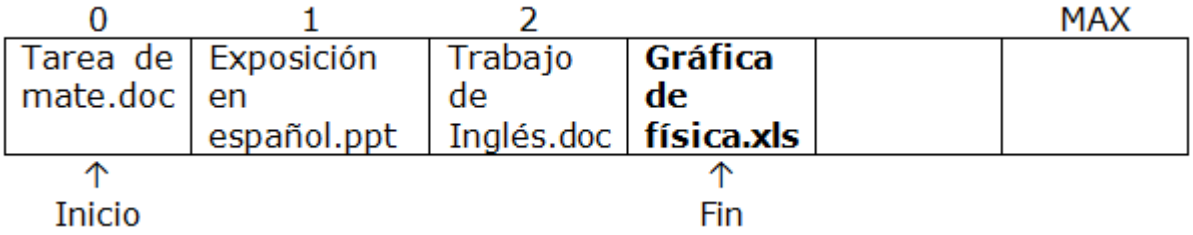


Solución ejercicio 7.2.9.

Solución ejercicio 7.2.10



Solución ejercicio 7.2.11:



Solución ejercicio 7.2.12**:**

Diagrama 1

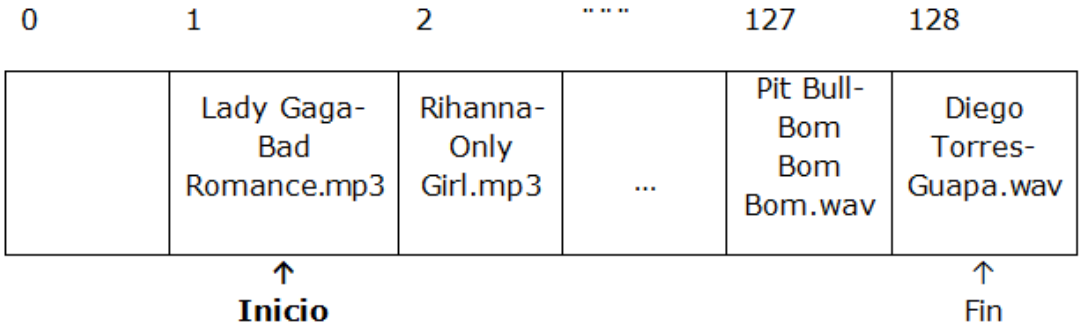
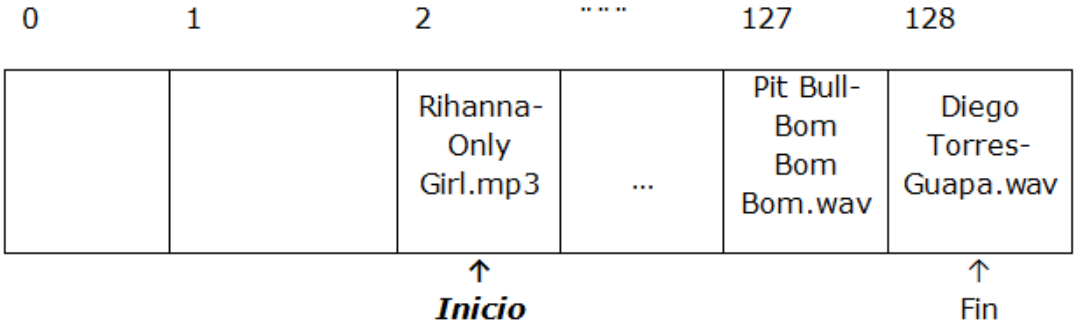
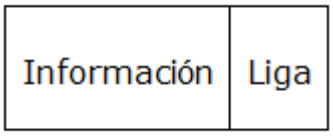


Diagrama 2

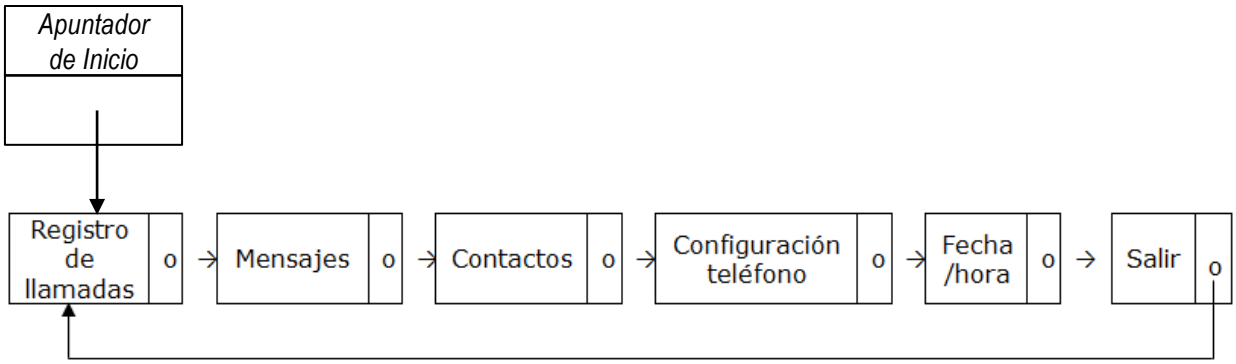


Listas

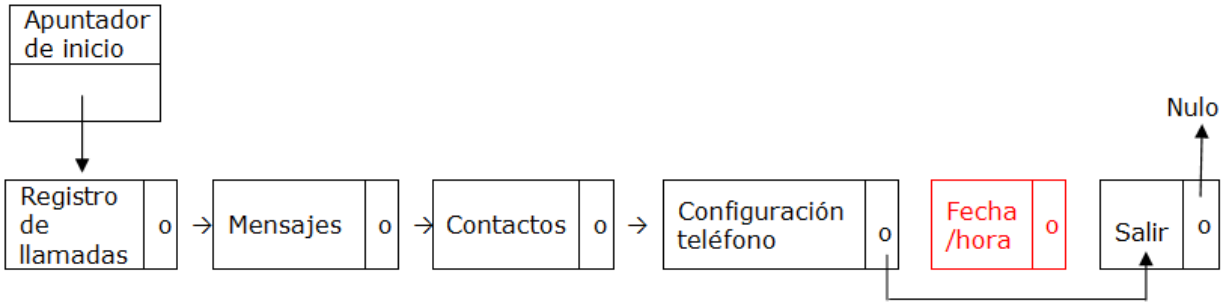
Solución ejercicio 7.2.13:



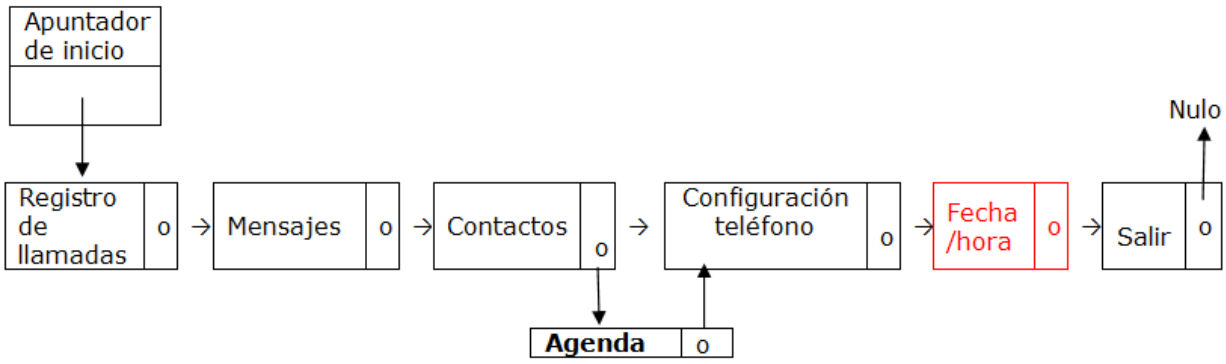
Solución ejercicio 7.2.14**:**



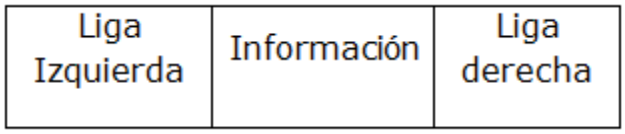
Solución ejercicio 7.2.15:



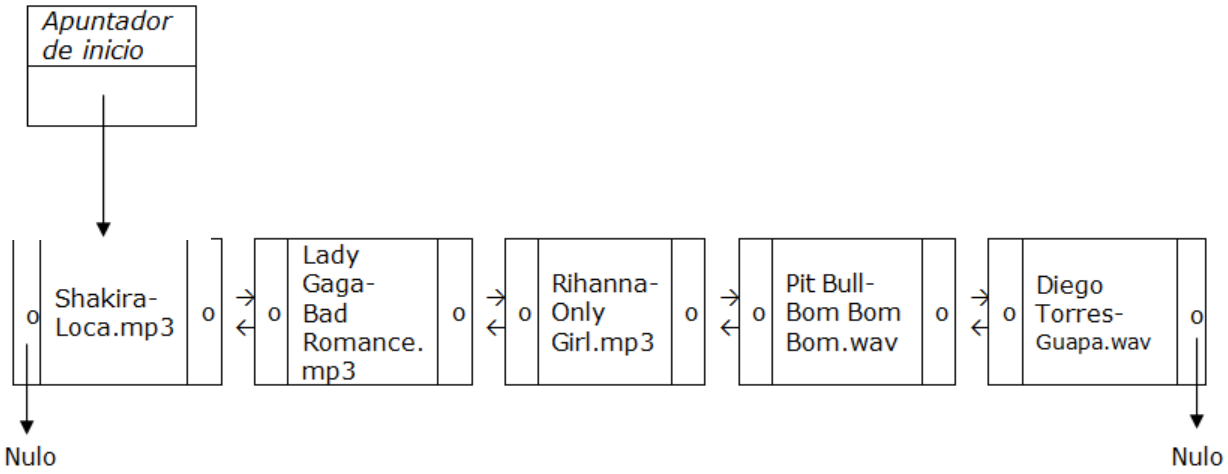
Solución ejercicio 7.2.16:



Solución ejercicio 7.2.17:



Solución ejercicio 7.2.18**:** *Lista doblemente enlazada.*



Solución ejercicio 7.2.19: **Agenda**.- La solución de este ejercicio, dada su laboriosidad, emplea los archivos que a continuación se enlistan:

1. `Ejercicio-Agenda.h`: Definiciones y prototipos de las funciones del programa de administración de una agenda de contactos, como prueba para la implementación de listas doblemente ligadas.
2. `Ejercicio-Agenda.c`: Programa de administración de una agenda de contactos, como prueba para la implementación de listas doblemente ligadas.
3. `Ejercicio-Listas-Ligadas.h`: Definiciones y prototipos de las funciones de la implementación de listas doblemente ligadas.
4. `Ejercicio-Listas-Ligadas.c`: Implementación de las listas doblemente ligadas.

La implementación en el lenguaje de programación C, de la solución a este ejercicio, maneja técnicas que se deben conocer y dominar, a fin de crear código reutilizable y bien estructurado. Se recomienda encarecidamente que estudie la solución detenidamente, que investigue las partes que aún no llegue a comprender y que, a futuro, las aplique en sus propios proyectos. Algunas de las técnicas referidas son:

- Uso de archivos de cabecera, para definiciones y prototipos de función.

- Uso de la palabra clave `typedef` para definición de sinónimos de tipos. Por ejemplo, la estructura autorreferenciada de los nodos de la lista doblemente ligada se define de esta forma:

```
typedef struct nodo *Nodoptr;

typedef struct nodo
{
    void *contenido;
    Nodoptr sig;
    Nodoptr prev;
} Nodo;
```

- Lo cual define a la palabra `Nodoptr` como un tipo, sinónimo del tipo `struct nodo*`, es decir, que a partir de este momento cualquier variable que pretenda ser del tipo apuntador a una estructura nodo, se podrá declarar empleando `struct nodo *variable` o `Nodoptr variable`. Esto se hace con la finalidad de dar mayor claridad al código fuente. De la misma manera se define el tipo `Nodo` como un sinónimo del tipo `struct nodo`.

- La técnica empleada más complicada de entender, quizás, sean los apuntadores a función. Éstos no son más que apuntadores comunes que indican la ubicación del código ejecutable de una función en la memoria. Por este motivo y para generalizar lo más posible el código de implementación de las listas doblemente ligadas, se abstraen las funciones que dependen del tipo específico del contenido insertado en los nodos de la lista. Por ejemplo, si el contenido en los nodos es de tipo cadena de caracteres, la función de comparación a utilizar sería la función de biblioteca *strcmp()* pero, si el contenido es del tipo `struct contacto`, la función de comparación debe implementarse específicamente para este tipo de estructuras, y cuyo prototipo pudiera ser `int compara_contactos(struct contacto c1, struct contacto c2)`. O, también, como el planteamiento del problema establece que el contenido de los nodos también debe manejarse con memoria asignada dinámicamente, el prototipo pudiera ser `int compara_contactos(struct contacto *ap_c1, struct contacto *ap_c2)`, es decir, los parámetros recibidos son apuntadores a estructuras de tipo contacto.

- Como puede observarse en el prototipo de la función de comparación definida para estructuras de tipo `contacto`, también se emplea la técnica de seguir el formato de la función de biblioteca *strcmp()*, es decir, la función de comparación devolverá un valor < 0 si el primer contacto es menor que el segundo, devolverá un valor == 0 si los contactos son iguales y devolverá un valor > 0, si el primer contacto es mayor que el segundo. La definición de <, == y > se implementa en la propia función, esto es, tomando como ejemplo la estructura contacto pudiéramos decir que un contacto es menor que otro si

su nombre es menor (lexicográficamente hablando). O podría tomarse un campo de edad del contacto y hacer la comparación con base en él. Cabe señalar que para generalizar la implementación de la lista, el prototipo de la función debe manejar apuntadores al tipo `void` y no al tipo `struct contacto`. Por esta razón, el apuntador a la función de comparación que reciben las funciones de la lista ligada es: `int (*comparar)(void *, void*)`, lo que significa que `comparar` es una variable apuntador a una función y, dicha función, recibe dos parámetros de tipo apuntador a `void`. Además, la función devuelve un valor entero (lo que arriba se comenta: <, == o > 0). Se deben observar los paréntesis alrededor del nombre de la variable apuntador a función, sin ellos, dicha variable se convertiría en una función propiamente dicha y no un apuntador.

```
/* ------------------------------------------------------------------
Programa: Ejercicio-Agenda.h
Objetivo: Definiciones y prototipos de las funciones del programa
    de administración de una agenda de contactos, como prueba
    para la implementación de listas doblemente ligadas.
/* ------------------------------------------------------------------
#ifndef AGENDA_H
#define AGENDA_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Opciones del menú */
#define SALIR     0
#define ALTAS     1
#define BAJAS     2
#define CONSULTAS 3
#define LISTADO   4

typedef struct contacto *Contactoptr;

typedef struct contacto
{
    char nombre[31];
    char apellidos[31];
    char celular[16];
    char correo[31];
} Contacto;

int  menu(void);
Contactoptr captura(void);
Contactoptr captura_info_id(void);
void imprime_contacto(Contactoptr c);
int  comp_nombre(Contactoptr c1, Contactoptr c2);
void clrscr(void);

#endif // AGENDA_H
```

```
/* ------------------------------------------------------------------
Programa: Ejercicio-Agenda.c
Objetivo: Programa de administración de una agenda de contactos,
    como prueba para la implementación de listas doblemente
    ligadas.
------------------------------------------------------------------*/
#include "agenda.h"
#include "listas_dobles.h"

int main()
{
    int opcion = -1;

    Nodoptr agenda = NULL;
    Contactoptr c = NULL;
    Contactoptr eliminado = NULL;
    Contactoptr encontrado = NULL;

    do
    {
        opcion = menu();

        switch (opcion)
        {
        case ALTAS:
            c = captura();

            /* Los casts:
                    (void *) c
                    (int (*)(void *, void*)) comp_nombre
               se requieren porque las funciones de la lista
               doblemente ligada reciben parámetros genéricos
               (tipo void), mientras que c y comp_apellido no lo
               son, la conversión sin cast generaría errores de
               compilación. Lo mismo pasa para las demás
               operaciones de la lista ligada */
            ins_orden(&agenda, (void *) c,
                      (int (*)(void *, void*)) comp_nombre);
            break;
        case BAJAS:
            c = captura_info_id();
            eliminado = (Contactoptr) elimina(&agenda, (void *) c,
                    (int (*)(void *, void*)) comp_nombre);

            if (NULL == eliminado)
                mensaje("\n\nEl contacto no existe.");
            else
            {
                imprime_contacto(eliminado);
                mensaje("\n\n...este fue el contacto eliminado.");

                /* Liberar la memoria asignada al contacto*/
                free(eliminado);
                eliminado = NULL;
```

```
            } /* else */
            break;
        case CONSULTAS:
            c = captura_info_id();
            encontrado = (Contactoptr) busca(agenda, (void *) c,
                    (int (*)(void *, void*)) comp_nombre);

            if (NULL == encontrado)
                mensaje("\n\nEl contacto no existe.");
            else
            {
                imprime_contacto(encontrado);
                mensaje("\n\n...datos del contacto solicitado.");

            } /* else */
            break;
        case LISTADO:
            clrscr();
            lista(agenda, (void (*)(void *)) imprime_contacto);
            break;
        default:
            break;

        } /* switch */

    } while (opcion != SALIR);

    /* Liberar la lista */
    destruye(&agenda);

    return 0;

} /* main() */


/* ---------------------------------------------------------------------
Función: menu()
Objetivo: Presentar el menú de opciones de la agenda de contactos.
Parámetros: Ninguno.
Salida: La opción elegida del menú.
---------------------------------------------------------------------*/
int menu(void)
{
    int opcion = SALIR;

    do
    {
        clrscr();
        printf("*** A G E N D A ***\n");
        printf("\n1.- Altas");
        printf("\n2.- Bajas");
        printf("\n3.- Consultas");
        printf("\n4.- Listado");
```

```
        printf("\n\n0.- Salir");

        if ((opcion < SALIR) || (opcion > LISTADO))
            printf("\n\nOpcion incorrecta, intente de nuevo...");

        printf("\n\nOpcion-> ");
        scanf("%d", &opcion);

    } while((opcion < SALIR) || (opcion > LISTADO));

    return opcion;

} /* menu() */


/* ------------------------------------------------------------------
Función: captura()
Objetivo: Solicitar la información correspondiente a un contacto.
Parámetros: Ninguno.
Salida: El apuntador al registro en memoria dinámica con la
    información del contacto capturado.
------------------------------------------------------------------*/
Contactoptr captura(void)
{
    Contactoptr cp = (Contactoptr) malloc(sizeof(Contacto));

    if (NULL == cp)
    {
        printf("\n\ncaptura(): No hay memoria disponible.\n\n");
        exit(1);

    } /* if */

    clrscr();
    printf("Capturando informacion...");

    /* El número después del carácter %, en scanf(), indica el
       número máximo de caracteres a leer. El texto [^\n] le
       indica a scanf() que admita todos los caracteres menos el
       carácter '\n', es decir el ENTER (por omisión, scanf() no
       lee espacios en blanco, con esto sí). La función
       _flushall() limpia todos los búfers del programa,
       incluyendo el del teclado, que es el que interesa en este
       caso (para que el programa no lea "basura" dejada en el
       búfer del teclado) */
    printf("\n\nNombre: ");
    _flushall();
    scanf("%30[^\n]s", cp->nombre);

    printf("\nApellidos: ");
    _flushall();
    scanf("%30[^\n]s", cp->apellidos);

    printf("\nTelefono celular: ");
```

```
    _flushall();
    scanf("%15[^\n]s", cp->celular);

    printf("\nCorreo electronico: ");
    _flushall();
    scanf("%30[^\n]s", cp->correo);

    return cp;

} /* captura() */


/* ---------------------------------------------------------------
Función: captura_info_id()
Objetivo: Solicitar la información mínima de identificación
    correspondiente a un contacto.
Parámetros: Ninguno.
Salida: El apuntador al registro en memoria dinámica con la
    información del contacto capturado.
---------------------------------------------------------------*/
Contactoptr captura_info_id(void)
{
    Contactoptr cp = (Contactoptr) malloc(sizeof(Contacto));

    if (NULL == cp)
    {
        printf("\n\ncaptura_info_id(): No hay memoria disponible.\n\n");
        exit(1);

    } /* if */

    clrscr();
    printf("Capturando informacion de identificacion...");

    printf("\n\nNombre: ");
    _flushall();
    scanf("%30[^\n]s", cp->nombre);

    printf("\nApellidos: ");
    _flushall();
    scanf("%30[^\n]s", cp->apellidos);

    /* Datos no necesarios para identificar a un contacto */
    strcpy(cp->celular, "");
    strcpy(cp->correo, "");

    return cp;

} /* captura_info_id() */
```

```
/* ------------------------------------------------------------------
Función: imprime_contacto()
Objetivo: Imprime la información correspondiente a un contacto.
Parámetros: c - Apuntador al registro en memoria dinámica con la
    información del contacto a imprimir en pantalla.
Salida: Ninguna.
------------------------------------------------------------------*/
void imprime_contacto(Contactoptr c)
{
    if (NULL == c)
        printf("\n\nimprime_contacto(): Apuntador a contacto nulo.\n\n");

    printf("\n\nNombre: %s", c->nombre);
    printf("\nApellidos: %s", c->apellidos);
    printf("\nTelefono celular: %s", c->celular);
    printf("\nCorreo electronico: %s", c->correo);

} /* imprime_contacto() */


/* ------------------------------------------------------------------
Función: comp_nombre()
Objetivo: Comparar los registros de dos contactos, en memoria
    dinámica, con base en sus nombres completos, empezando por
    los apellidos.
Parámetros: c1 y c2 - Los registros de los contactos a comparar.
Salida:
    Un valor  < 0 si c1 < c2
    Un valor == 0 si c1 == c2
    Un valor  > 0 si c1 > c2
------------------------------------------------------------------*/
int comp_nombre(Contactoptr c1, Contactoptr c2)
{
    char nombre_completo1[61];
    char nombre_completo2[61];

    /* Concatenar el nombre completo del contacto 1 */
    strcpy(nombre_completo1, c1->apellidos);
    strcat(nombre_completo1, " ");
    strcat(nombre_completo1, c1->nombre);

    /* Concatenar el nombre completo del contacto 2 */
    strcpy(nombre_completo2, c2->apellidos);
    strcat(nombre_completo2, " ");
    strcat(nombre_completo2, c2->nombre);

    return strcmp(nombre_completo1, nombre_completo2);

} /* comp_nombre() */
```

```
/* ---------------------------------------------------------------------
Función: clrscr()
Objetivo: Limpiar la pantalla.
Parámetros: Ninguno.
Salida: Ninguna.
---------------------------------------------------------------------*/
void clrscr(void)
{
    system("cls");

} /* clrscr() */
```

```
/* ------------------------------------------------------------------
Programa: Ejercicio-Listas-Ligadas.h
Objetivo: Definiciones y prototipos de las funciones de la
    implementación de listas doblemente ligadas.
------------------------------------------------------------------*/
#ifndef LISTAS_DOBLES_H
#define LISTAS_DOBLES_H

typedef struct nodo *Nodoptr;

typedef struct nodo
{
    void *contenido;
    Nodoptr sig;
    Nodoptr prev;
} Nodo;

void *busca(Nodoptr l, void *contenido, int (*comparar)(void *, void*));
Nodoptr crea_nodo(void *contenido);
void *elimina(Nodoptr *l, void *contenido, int (*comparar)(void *, void*));
Nodoptr nodo_elimina(Nodoptr l, void *contenido,
                     int (*comparar)(void *, void*));
void ins_inicio(Nodoptr *l, void *contenido);
void ins_orden(Nodoptr *l, void *contenido, int (*comparar)(void *, void*));
void nodo_ins_orden(Nodoptr nodo_ref, Nodoptr n,
                    int (*comparar)(void *, void*));
void lista(Nodoptr l, void (*imprimir)(void *));
void destruye(Nodoptr *l);
void mensaje(char *m);

#endif // LISTAS_DOBLES_H
```

```c
/* ------------------------------------------------------------------
Programa: Ejercicio-Listas-Ligadas.c
Objetivo: Implementación de listas doblemente ligadas.
------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include "listas_dobles.h"

/* ------------------------------------------------------------------
Función: busca()
Objetivo: Buscar recursivamente el contenido de un nodo en una
    lista doblemente ligada.
Parámetros:
    l         - La lista en donde se buscará.
    contenido - Apuntador al contenido a buscar.
    comparar  - Apuntador a una función de comparación que permita
        encontrar el contenido buscado en los nodos.
Salida: El apuntador al contenido encontrado, o NULL si no se
    encontró.
------------------------------------------------------------------*/
void *busca(Nodoptr l, void *contenido,
            int (*comparar)(void *, void*))
{
    int res = 0;

    if (NULL == l) return NULL;

    res = (*comparar)(contenido, l->contenido);

    if (0 == res) return l->contenido;
    if (res < 0) return NULL;

    /* res > 0 */
    return busca(l->sig, contenido, comparar);

} /* busca() */


/* ------------------------------------------------------------------
Función: crea_nodo()
Objetivo: Crear e inicializar un nodo de la lista doblemente ligada.
Parámetros: contenido - Apuntador al contenido del nodo a crear.
Salida: El apuntador al nodo creado.
------------------------------------------------------------------*/
Nodoptr crea_nodo(void *contenido)
{
    Nodoptr n;

    n = (Nodoptr) malloc(sizeof(Nodo));

    if (NULL == n)
    {
        mensaje("\n\ncrea_nodo(): No hay memoria disponible.\n\n");
        exit(1);
```

```
    } /* if */

    n->contenido = contenido;
    n->sig = NULL;
    n->prev = NULL;

    return n;

} //* crea_nodo() */


/* ------------------------------------------------------------------
Función: elimina()
Objetivo: Eliminar recursivamente el nodo de una lista doblemente
    ligada con un contenido específico.
Parámetros:
    l         - La lista en donde se eliminará.
    contenido - Apuntador al contenido a eliminar.
    comparar  - Apuntador a una función de comparación que permita
        encontrar el contenido a eliminar.
Salida: El apuntador al contenido eliminado, o NULL si no se
    encontró. Importante: Es responsabilidad de quien llama a
    esta función, el liberar la memoria correspondiente al
    contenido del nodo eliminado así como la del propio nodo.
------------------------------------------------------------------*/
void *elimina(Nodoptr *l, void *contenido,
              int (*comparar)(void *, void*))
{
    Nodoptr np = NULL;
    int res = 0;
    void *c_eliminado = NULL;

    if (NULL == *l) return NULL;

    res = (*comparar)(contenido, (*l)->contenido);
    if (0 == res)
    {
        np = *l;
        *l = (*l)->sig;
        np->sig = NULL;

        c_eliminado = np->contenido;
        free(np); /* No olvidar liberar la memoria del nodo */

        return c_eliminado;

    } /* if */

    if (res < 0) return NULL;

    /* res > 0 */
    np = nodo_elimina((*l)->sig, contenido, comparar);
```

```
    if (NULL == np) return NULL;

    c_eliminado = np->contenido;
    free(np); /* No olvidar liberar la memoria del nodo */

    return c_eliminado;

} /* elimina() */


/* ------------------------------------------------------------------
Función: nodo_elimina()
Objetivo: Función de apoyo para eliminar recursivamente el nodo
    de una lista doblemente ligada con un contenido específico.
Parámetros:
    l         - Apuntador a un nodo que, posiblemente, se
        eliminará de la lista doblemente ligada.
    contenido - Apuntador al contenido a eliminar.
    comparar  - Apuntador a una función de comparación que permita
        encontrar el contenido a eliminar.
Salida: El apuntador al contenido eliminado, o NULL si no se
    encontró.
------------------------------------------------------------------*/
Nodoptr nodo_elimina(Nodoptr l, void *contenido,
                     int (*comparar)(void *, void*))
{
    int res = 0;

    if (NULL == l) return NULL;

    res = (*comparar)(contenido, l->contenido);

    if (0 == res)
    {
        if (l->prev != NULL)
        {
            l->prev->sig = l->sig;

        } /* if */

        if (l->sig != NULL)
        {
            l->sig->prev = l->prev;

        } /* if */

        l->sig = NULL;
        l->prev = NULL;

        return l;

    } /* if */

    if (res < 0) return NULL;
```

```
    /* res > 0 */
    return nodo_elimina(l->sig, contenido, comparar);

} /* nodo_elimina() */


/* ------------------------------------------------------------------
Función: ins_inicio()
Objetivo: Insertar un contenido en un nodo y, éste, al inicio de
    una lista doblemente ligada.
Parámetros:
    l         - La lista en donde se insertará.
    contenido - Apuntador al contenido a insertar.
Salida: Ninguna.
------------------------------------------------------------------*/
void ins_inicio(Nodoptr *l, void *contenido)
{
    Nodoptr np = NULL;

    np = crea_nodo(contenido);

    if (NULL == (*l)) *l = np;
    else
    {
        np->sig = *l;
        (*l)->prev = np;
        *l = np;

    } /* else */

} /* ins_inicio() */


/* ------------------------------------------------------------------
Función: ins_orden()
Objetivo: Insertar un contenido en un nodo y, éste, ordenadamente
    en una lista doblemente ligada.
Parámetros:
    l         - La lista en donde se insertará.
    contenido - Apuntador al contenido a insertar.
    comparar  - Apuntador a una función de comparación que permita
        insertar ordenadamente el contenido dado.
Salida: Ninguna.
------------------------------------------------------------------*/
void ins_orden(Nodoptr *l, void *contenido, int (*comparar)(void *, void*))
{
    Nodoptr np = NULL;
    int res = 0;

    np = crea_nodo(contenido);
```

```
    if (NULL == *l) *l = np;
    else
    {
        res = (*comparar)(np->contenido, (*l)->contenido);

        if (0 == res) mensaje("\n\nins_orden(): Elemento repetido.");
        else if (res < 0)
        {
            np->sig = *l;
            (*l)->prev = np;
            *l = np;

        }
        else /* res > 0 */
        {
            if (NULL == (*l)->sig)
            {
                (*l)->sig = np;
                np->prev = *l;
                /* np->sig = NULL; */

            }
            else nodo_ins_orden((*l)->sig, np, comparar);

        } /* else res > 0 */

    } /* else *l != NULL */

} /* ins_orden() */


/* ---------------------------------------------------------------------
Función: nodo_ins_orden()
Objetivo: Función de apoyo para insertar recursiva y ordenadamente
    un nodo en una lista doblemente ligada.
Parámetros:
    nodo_ref - Apuntador a un nodo de referencia antes del cual,
        posiblemente, se inserte un nodo en una lista doblemente
        ligada.
    np       - Apuntador al nodo que se pretende insertar
        ordenadamente
    comparar - Apuntador a una función de comparación que permita
        insertar ordenadamente un nodo dado.
Salida: Ninguna.
---------------------------------------------------------------------*/
void nodo_ins_orden(Nodoptr nodo_ref, Nodoptr np,
                    int (*comparar)(void *, void*))
{
    int res = 0;

    res = (*comparar)(np->contenido, nodo_ref->contenido);
    if (0 == res) mensaje("\n\nnodo_ins_orden(): Elemento repetido.");
    else if (res < 0)
    {
```

```
        np->prev = nodo_ref->prev;
        np->sig = nodo_ref;
        nodo_ref->prev->sig = np;
        nodo_ref->prev = np;

    }
    else /* res > 0 */
    {
        if (NULL == nodo_ref->sig)
        {
            np->sig = NULL;
            np->prev = nodo_ref;
            nodo_ref->sig = np;

        }
        else nodo_ins_orden(nodo_ref->sig, np, comparar);

    } /* else res > 0 */

} /* nodo_ins_orden() */


/* ---------------------------------------------------------------------
Función: lista()
Objetivo: Mostrar en pantalla el contenido de cada nodo de una
    lista doblemente ligada.
Parámetros: .
    l - Apuntador a la lista cuyo contenido de sus nodos se va a
        listar.
    imprimir - Apuntador a una función que permita imprimir en
        pantalla el contenido de cada nodo de la lista.
Salida: Ninguna.
---------------------------------------------------------------------*/
void lista(Nodoptr l, void (*imprimir)(void *))
{
    Nodoptr np = NULL;

    printf("Contenido de la lista:");

    for (np = l; np != NULL; np = np->sig)
        (*imprimir)(np->contenido);

    mensaje("\n\nFin de la lista.");

} /* lista() */
```

```
/* ---------------------------------------------------------------
Función: destruye()
Objetivo: Destruir una lista doblemente ligada, es decir, libera
    la memoria correspondiente a los contenidos de sus nodos así
    como la de los propios nodos.
Parámetros: .
    l - Apuntador a la lista que se destruirá.
Salida: Ninguna.
---------------------------------------------------------------*/
void destruye(Nodoptr *l)
{
    Nodoptr np_sig = NULL;
    Nodoptr np = NULL;

    if (NULL == *l) return;

    /* Resguardar el resto de la lista */
    np_sig = (*l)->sig;

    /* Liberar contenido (y el propio nodo) del nodo inicial */
    free((*l)->contenido);
    free(*l);
    *l = NULL;

    np = np_sig;
    while (np != NULL)
    {
        /* Resguardar el resto de la lista */
        np_sig = np->sig;

        /* Liberar contenido y el propio nodo */
        free(np->contenido);
        free(np);

        /* Preparar iteración siguiente */
        np = np_sig;

    } /* while */
} /* destruye() */

/* ---------------------------------------------------------------
Función: mensaje()
Objetivo: Imprimir un mensaje en pantalla y esperar a que se
    presione una tecla.
Parámetros: m - El mensaje a imprimir.
Salida: Ninguna.
---------------------------------------------------------------*/
void mensaje(char *m)
{
    printf("%s\n\nPresione una tecla para continuar...\n\n", m);

    if ('\n' == getchar()) getchar();

} /* mensaje() */
```

Octava parte

Referencias

Alcock, Donald. *Illustrating C.* New York: Cambridge University Press, 1992.

Barry, Paul, y David Griffiths. *Head First Programming.* Sebastopol: O'Reilly Media, Inc., 2009.

Bell, Douglas. *Software Engineering for Students: A Programming Approach.* 4th. Edition. Pearson Education Limited, 2005.

Brassard, G., y P. Bratley. *Fundamentos de algoritmia.* Prentice-Hall, 1997.

Cairó Battistutti, Osvaldo. *Metodología de la programación: Algoritmos, diagramas de flujo y programas.* 3a. Ediciòn. México, D. F.: Alfaomega Grupo Editor, S. A. de C. V., 2005.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, y Clifford Stein. *Introduction to Algorithms.* 3rd. Edition. Cambridge: MIT Press, 2009.

Darnell, Peter A., y Philip E. Margolis. *C, A Software Engineering Approach.* 3rd. Edition. New York: Springer-Verlag New York, Inc., 1996.

Deitel, Harvey M., y Paul J. Deitel. *Cómo programar en C/C++ y Java.* 4a. edición. Naucalpan de Juárez: Pearson Educación de México, S.A. de C.V., 2004.

Deitel, Paul J., y Harvery M. Deitel. *C++11 for Programmers.* 2nd. Edition. Boston: Pearson Education, Inc., 2014.

Deitel, Paul J., y Harvey M. Deitel. *C How to Program.* 7th. Edition. New Jersey: Pearson Education, Inc., 2013.

Gookin, Dan. *C for Dummies.* Hoboken: Wiley Publishing, Inc., 2004.

Gottfried, Byron S. *Schaum's Outline of Theory and Problems of Programming with C.* 2nd. Edition. McGraw-Hill, 1996.

Goyal, Arunesh. *Moving from C to C++.* New York: Apress, 2013.

Griffiths, David, y Dawn Griffiths. *Head First C.* Sebastopol: O'Reilly Media, Inc., 2012.

Hanly, Jeri R., y Elliot B. Koffman. *Problem Solving and Problem Design in C.* 7th. Edition. New Jersey: Pearson Education, Inc., 2013.

Haskins, David. *C Programming in Linux.* David Haskins & Ventus Publishing ApS, 2009.

Horton, Ivor. *Beginning C: From Novice to Professional.* 4th. Edition. Berkeley: Apress, 2006.

ISO/IEC 14882:2003. *Programming Languages - C++.* The C++ Standard, 2003.

ISO/IEC 14882:2011. *Programming Languages - C++.* The C++ Standard, 2011.

ISO/IEC 9899:1990. *Programming Languages - C.* The C Standard, 1989.

ISO/IEC 9899:1999. *Programming Languages - C.* The C Standard, 1999.

ISO/IEC 9899:2011. *Programming Languages - C.* The C Standard, 2011.

Jalote, Pankaj. *A Concise Introduction to Software Engineering.* Springer-Verlag London Limited, 2008.

Jensen, Ted. *A Tutorial on Pointers and Arrays in C.* Redwood City: Dominio público, 2003.

—. *Tutorial sobre apuntadores y arreglos en C.* Redwood City: Dominio público, 2000.

Kalicharan, Noel. *Advanced Topics in C: Core Concepts in Data Structures.* New York: Apress, 2013.

Kernighan, Brian W., y Dennis M. Ritchie. *The C Programming Language.* 2nd. Edition. New Jersey: Prentice-Hall, 1988.

Kernighan, Brian W., y Rob Pike. *The Practice of Programming.* Addison Wesley Longman, Inc., 1999.

King, K. N. *C Programming, A Modern Approach.* 2nd. Edition. W. W. Norton & Company, 2008.

Kochan, Stephen G. *Programming in C.* 3rd. Edition. Indiana: Sam's Publishing, 2005.

Liberty, Jesse, y Bradley Jones. *Sam's Teach Yourself C++ in 21 Days.* 5th. Edition. Sam's Publishing, 2005.

Oualline, Steve. *Practical C Programming.* 3rd. Edition. Sebastopol: O'Reilly Media, Inc., 2011.

Perry, Greg. *Absolute Beginner's Guide to C.* 2nd. Edition. Indiana: Sam's Publishing, 1994.

Perry, Greg, y Dean Miller. *C Programming Absolute Beginner's Guide.* 3rd. Edition. Pearson Education, Inc., 2014.

Pilone, Dan, y Russ Miles. *Head First Software Development.* Sebastopol: O'Reilly Media, Inc., 2008.

Plauger, P. J. *The Standard C Library.* Prentice-Hall, 1992.

Prata, Stephen. *C Primer Plus.* 6th. Edition. Pearson Education, Inc., 2014.

Pressman, Roger S., y Bruce R. Maxim. *Software Engineering: A Practitioner's Approach.* 8th. Edition. McGraw-Hill, 2014.

Prinz, Peter, y Tony Crawford. *C in a Nutshell.* Sebastopol: O'Reilly Media, Inc., 2006.

Prinz, Peter, y Ulla Kirch-Prinz. *C Pocket Reference.* Sebastopol: O'Reilly Media, Inc., 2002.

Reek, Kenneth A. *Pointers on C.* Addison-Wesley Longman, Inc., 1998.

Reese, Richard. *Understanding and Using C pointers.* Sebastopol: O'Reilly Media, Inc., 2013.

Robertson, Lesley Anne. *Simple Program Design: A Step-by-Step Approach.* 4th. Edition. Boston: Thomson Learning, Inc., 2004.

Sedgewick, Robert, y Kevin Wayne. *Algorithms.* 4th. Edition. Boston: Pearson Education, Inc., 2011.

Sommerville, Ian. *Software Engineering.* 9th. Edition. Boston: Pearson Education, Inc., 2011.

Stroustrup, Bjarne. *Programming: Principles and Practice using C++.* 2nd. Edition. Boston: Pearson Education, Inc., 2014.

Tondo, Clovis L., y Scott E. Gimpel. *The C Answer Book.* 2nd. Edition. Prentice-Hall International, Inc., 1989.

Vine, Michael. *C Programming for the Absolute Beginner.* 2nd. Edition. Boston: Thomson Learning, Inc., 2008.