



**Manual de prácticas del
Laboratorio de Fundamentos
de programación**

Código:	MADO-17
Versión:	05
Página	1/187
Sección ISO	8.3
Fecha de emisión	22 / enero / 2025

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Manual de prácticas del laboratorio de Fundamentos de programación

Elaborado por:	Revisado por:	Autorizado por:	Vigente desde:
Ing. Jorge Alberto Solano Gálvez M.C. Edgar E. García Cano M.C. Laura Sandoval Montaño M.C. Cintia Quezada Reyes M.I. Tanya Itzel Arteaga Ricci Ing. María Guadalupe Morales Nava Ing. Manuel Castañeda Castañeda. Ing. Maricela Castañeda Perdomo Ing. Julio De León Razo M.T. Hugo Zúñiga Barragán	Ing. Laura Sandoval Montaño (jefe de academias) Ing. Jorge Alberto Solano Gálvez M.I. Tanya Itzel Arteaga Ricci M.C. Cintia Quezada Reyes Ing. Manuel Castañeda Castañeda Ing. Maricela Castañeda Perdomo Ing. Julio De León Razo	Dra. Rocío Alejandra Aldeco Pérez	22-enero-2025

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	2/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Índice de prácticas

No	Nombre	Página
1	La computación como herramienta de trabajo del profesional de ingeniería	3
2	GNU/Linux	23
3	Solución de problemas y Algoritmos.	36
4	Diagramas de flujo	52
5	Pseudocódigo	73
6	Entorno y fundamentos de Lenguaje C	85
7	Estructuras de selección	113
8	Estructuras de repetición	126
9	Arreglos unidimensionales	135
10	Arreglos multidimensionales	149
11	Funciones	162
12	Lectura y escritura de datos	176

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
Versión:		05	
Página		3/187	
Sección ISO		8.3	
Fecha de emisión		22 / enero / 2025	
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 01:

La computación como herramienta de trabajo del profesional de ingeniería



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	Ing. Manuel Enrique Castañeda Castañeda	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	4/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 01: La computación como herramienta de trabajo del profesional de ingeniería

Objetivo:

El alumno conocerá y utilizará herramientas de software que ofrecen las Tecnologías de la Información y Comunicación que le permitan realizar actividades y trabajos académicos de forma organizada y profesional a lo largo de la vida escolar, tales como manejo de repositorios de almacenamiento, búsquedas de información especializada y revisión de información arrojada por generadores de contenido mediante la escritura de un prompt.

Actividades:

- Realizar búsquedas de información especializada.
- Revisar y validar contenido creado por inteligencia artificial generativa.
- En casa, crear un repositorio de almacenamiento en línea.

Introducción

El uso de dispositivos de cómputo y comunicación se vuelve fundamental para el desempeño de muchas actividades, las cuales pueden ser de la vida cotidiana, académica, profesional, empresarial e inclusive de entretenimiento.

Como futuros profesionales de la ingeniería, los estudiantes de esta disciplina requieren conocer y utilizar las herramientas de las Tecnologías de la Información y Comunicación (TIC) que les apoyen tanto en sus tareas académicas como en su próxima vida profesional. De la gran gama de herramientas TIC existentes, en esta práctica nos enfocaremos en las herramientas para manejo de repositorios de almacenamiento, buscadores de información especializada en Internet y revisión de información arrojada por generadores de contenido, las cuales permitirán a los estudiantes realizar las siguientes actividades en apoyo a sus tareas académicas:

- Almacenamiento de la información de manera organizada en repositorios que sean accesibles, seguros y que la disponibilidad de la información sea las 24 horas de los 365 días del año.
- Búsqueda de información especializada en Internet.
- Revisar información que arrojen las herramientas de generación de contenidos.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión	MADO-17 05 5/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Control de Versiones

Un controlador de versiones es un sistema de software que lleva a cabo el registro de los cambios sobre uno o más archivos (sin importar el tipo de archivos) a lo largo del tiempo.

Estos sistemas permiten regresar a versiones específicas de nuestros archivos, revertir y comparar cambios, revisar quién hizo ciertas modificaciones, así como proteger nuestros archivos de errores humanos o de consecuencias no previstas o no deseadas. Además, un control de versiones nos facilita el trabajo colaborativo y nos permite tener un respaldo de nuestros archivos.

Actualmente esta herramienta es sumamente importante para los profesionistas del software, sin embargo, su uso se extiende a diseñadores, escritores o cualquiera que necesite llevar un control más estricto sobre los cambios en sus archivos.

Tipos de Sistemas de Control de Versiones

Sistema de Control de versiones Local

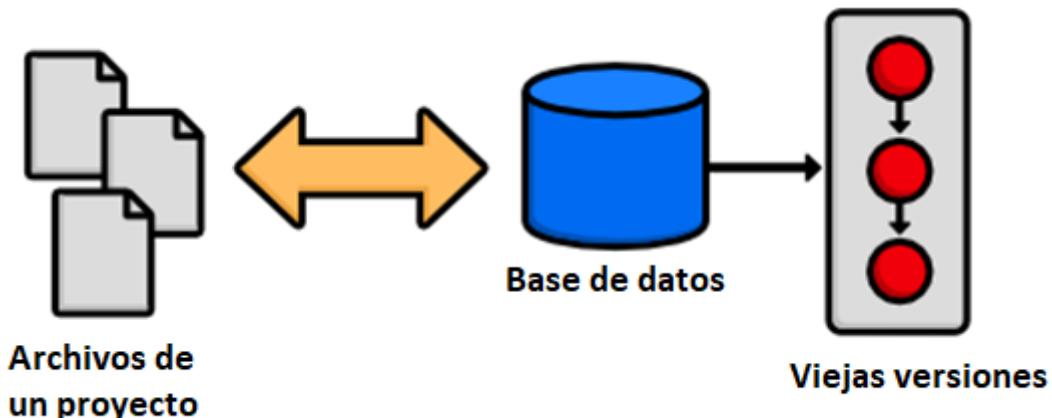


Figura 1. Control de versiones local

En estos sistemas, el registro de los cambios de los archivos se almacena en una base de datos local. (Figura 1)

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 6/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Sistema de Control de Versiones Centralizado

Estos sistemas están pensados para poder trabajar de manera colaborativa, por lo que un servidor central lleva el control de las versiones y cada usuario descarga los archivos desde ese servidor y sube sus cambios al mismo. (Figura 2)

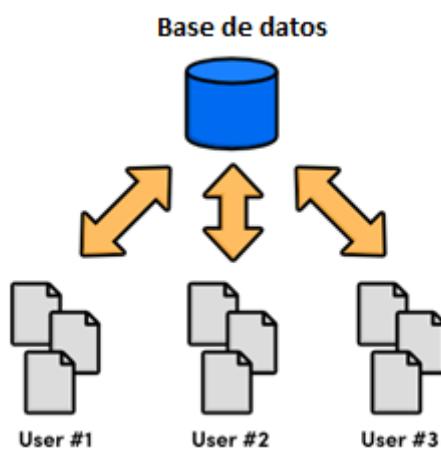


Figura 2: Control de Versiones Centralizado

Sistema de Control de Versiones Distribuido

En estos sistemas, los usuarios tienen una copia exacta del proyecto, así como todo el registro de las versiones, de esta manera si el servidor remoto falla o se corrompe, los usuarios pueden restablecer el servidor con sus propias copias de seguridad y obtener los cambios en los archivos directamente del equipo de otros usuarios. (Figura 3)

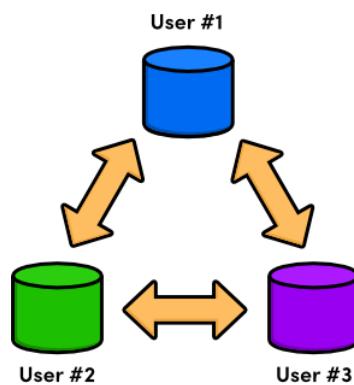


Figura 3: Control de Versiones Distribuido

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	7/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Git

Es un sistema de control de versiones de código libre, escrito en C, multiplataforma creado en 2005 por Linus Torvalds, desarrollado por la necesidad de tener un sistema de control de versiones eficiente para la elaboración del Kernel de Linux. Hoy en día es el sistema de control de versiones más usado y adoptado en el mundo.

Repositorio

Es el directorio de trabajo usado para organizar un proyecto, aquí se encuentran todos los archivos que integran nuestro proyecto, y en el caso de Git, todos aquellos que son necesarios para llevar a cabo el control de versiones.

Repositorio Local

Es aquel que se encuentra en nuestro propio equipo y solo el dueño del equipo tiene acceso a él.

Repositorio Remoto

Es aquel que está alojado en la nube, esto quiere decir, que se encuentra en un servidor externo, el cual puede ser accedido desde Internet y que nos va a permitir tener siempre a la mano nuestros archivos. Algunos de los proyectos que usan como base Git son: github.com, bitbucket.org o gitlab.com, todos ofreciendo diferentes características.

Github

Es una plataforma de almacenamiento para control de versiones y colaboración. que permite almacenar nuestros repositorios de una forma fácil y rápida, además nos da herramientas para el mejor control del proyecto, posibilidad de agregar colaboradores, notificaciones, herramientas gráficas y mucho más. Actualmente Github cuenta con más de 31 millones de usuarios haciéndola la plataforma más grande de almacenamiento de código en el mundo.

Operaciones en un repositorio

Agregar

Esta operación agrega archivos en nuestro repositorio para ser considerados en el nuevo estado guardado del proyecto. Por lo general son los archivos creados o que tienen nuevas modificaciones.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	8/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Commit

Esta operación se encarga de registrar los archivos agregados para generar un nuevo estado (o versión) en nuestro repositorio, un *commit* puede registrar uno o más archivos, y van acompañados de una explicación de lo que agregamos o cambiamos.

Ramas (Branches)

Nuestro repositorio se puede ver como un árbol, donde la rama principal (generalmente llamada *master*) contiene nuestro trabajo revisado y funcionando. Una rama es una bifurcación de otra rama en la cual podemos realizar nuevas modificaciones y pruebas sin afectar los archivos que ya funcionan, una vez que hayamos terminado las nuevas modificaciones sobre esa rama, se puede fusionar (*merge*) con la rama padre y ésta tendrá los nuevos cambios ya aprobados.

Almacenamiento en la nube

El almacenamiento en la nube (o *cloud storage*, en inglés) es un modelo de servicio en el cual los datos de un sistema de cómputo se almacenan, se administran y se respaldan de forma remota, normalmente en servidores que están en la nube y que son administrados por el proveedor del servicio. Estos datos se ponen a disposición de los usuarios a través de una red, como lo es Internet.

Google Drive, OneDrive, iCloud o Dropbox son algunos espacios de almacenamiento en la nube. Además, Google Drive (Google) y OneDrive (Outlook) cuentan con herramientas que permiten crear documentos de texto, hojas de cálculo y presentaciones, donde el único requisito es tener una cuenta de correo de dichos proveedores.

Este tipo de herramientas hace posible editar un documento y compartirlo con uno o varios contactos, de tal manera que todos pueden trabajar grupalmente en un solo documento.

Por lo tanto, los documentos creados pueden ser vistos, editados, compartidos y descargados en cualquier sistema operativo, ya sea Windows, Mac OS o Linux, y en cualquier dispositivo con capacidad de procesamiento como teléfonos inteligentes, tabletas y computadoras. (Figura 4)

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	9/187
		Sección ISO	8.3
Facultad de Ingeniería	Fecha de emisión		
	22 / enero / 2025		
Área/Departamento:		Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			



Figura 4. Dispositivos de conexión a la nube

[Google Forms](#)

Google Drive cuenta con una aplicación para recolectar información usando formularios (Forms), una particularidad de la hoja de cálculo. Se puede generar una serie de preguntas que pueden ser mandadas y contestadas por un grupo de personas. También proporciona un resumen con gráficas de los datos obtenidos del formulario (Figura 5).

New forms features

What do you think about the new Forms features?

	This will change my life	Gee whiz, finally!	Pretty cool	Meh	I dislike change
Grid question type	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bi-Di input support	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Improved results summary charts	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Sign-in to view	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Pre-populate via parameter	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figura 5. Google Forms

[OneNote](#)

Por otro lado, a través de OneDrive de Microsoft se puede utilizar la aplicación OneNote. El editor OneNote es muy amigable para realizar apuntes como si se ocupara una libreta de papel, pero con la diferencia de que todo se queda guardado en la nube.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 10/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

[Dropbox](#)

Dropbox es una herramienta que sirve para almacenar cualquier tipo de archivo digital en Internet. Para utilizarlo es necesario contar con una cuenta de correo para darse de alta en el sitio. Una vez realizado el registro se puede acceder al sitio, ya sea por medio de su interfaz web o descargando la aplicación que puede ser instalada en cualquier sistema operativo (teléfonos inteligentes, tabletas y computadoras).

Dropbox cuenta con aplicaciones de Microsoft Office Online para editar documentos. Los documentos también pueden ser compartidos con otros usuarios, ya sea compartiendo la carpeta que los contiene o por medio de un enlace.

Buscadores de Internet Académicos

En el contexto académico, los buscadores especializados se han convertido en herramientas indispensables para localizar información confiable, revisada por pares y relevante para investigaciones o actividades escolares. A continuación, se describen algunos de los principales motores de búsqueda académica y sus aplicaciones.

[Google Scholar \(Google Académico\)](#)

Especializado en artículos de revistas científicas, tesis, libros y otros recursos académicos. Permite filtrar por año, tipo de documento y citas relacionadas. Ideal para estudiantes que buscan trabajos de investigación de diversas disciplinas. (Figura 6).

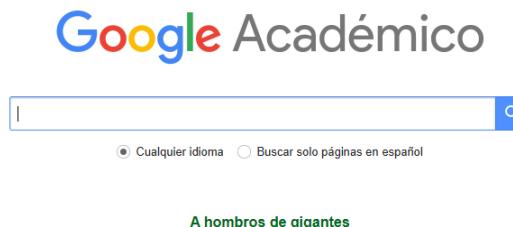


Figura 6. Google Scholar

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17
		Versión: 05
		Página 11/187
		Sección ISO 8.3
		Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

Microsoft Academic

Proporciona acceso a literatura académica y métricas de impacto. Ofrece visualizaciones y análisis de redes de citación (Figura 7).

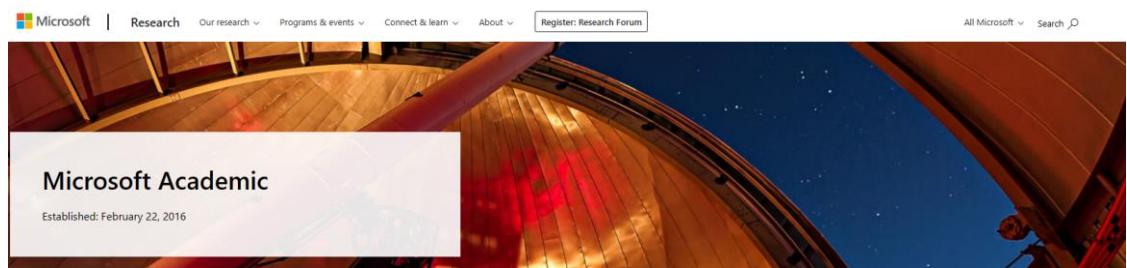


Figura 7. Microsoft Academic

ScienceDirect y SpringerLink

Repositorios de revistas y libros de editoriales líderes en ciencia y tecnología. Proporcionan acceso a publicaciones de alto impacto (Figura 8).

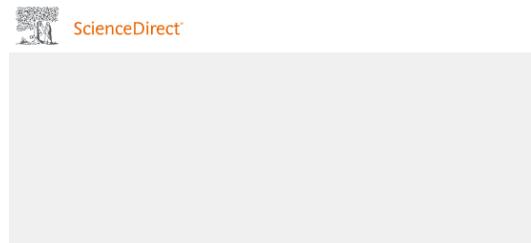


Figura 8. ScienceDirect

ResearchGate

Plataforma que conecta investigadores de diversas disciplinas. Permite descargar artículos, interactuar con autores y acceder a documentos compartidos directamente (Figura 9).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17
		Versión: 05
		Página 12/187
		Sección ISO 8.3
		Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

ResearchGate

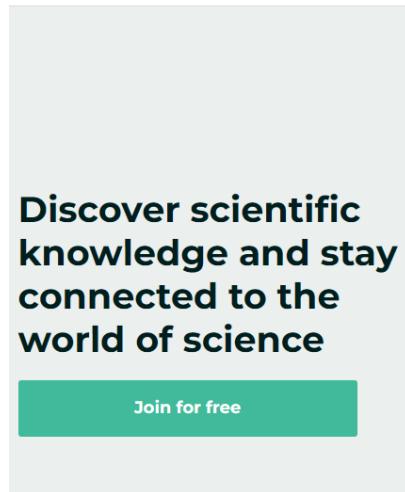


Figura 9. ScienceDirect

BASE (Bielefeld Academic Search Engine)

Uno de los buscadores académicos más completos con acceso a recursos de bibliotecas y repositorios. Incluye documentos de acceso abierto.



Figura 10. Bielefeld Academic Search Engine

Repositorio UNAM

Plataforma de acceso abierto que concentra trabajos académicos, tesis, artículos y recursos de investigación de la Universidad Nacional Autónoma de México. Disponible en: [https://repositorio.unam.mx/ \(Figuras 11 y 12\).](https://repositorio.unam.mx/)

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 13/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		



Figura 11. Repositorio UNAM



Figura 12. Repositorio UNAM, índices de contenido

Introducción a herramientas de IA para análisis de datos, generación de contenido y automatización de tareas

La Inteligencia Artificial (IA) ha revolucionado la forma en que manejamos, analizamos y generamos contenido, facilitando tareas que antes eran arduas y demandaban mucho tiempo. En el ámbito de la ingeniería, la IA permite automatizar procesos, realizar análisis predictivos y optimizar decisiones basadas en grandes volúmenes de datos. Las herramientas de IA han llegado a ser fundamentales para realizar tareas de manera más eficiente y precisa, mejorando significativamente la productividad y la innovación.

De aquí la importancia de que se exploraren diversas herramientas de IA que faciliten el análisis de datos, la creación automática de contenido y la automatización de tareas repetitivas. A través de prácticas específicas, se entenderá cómo integrar estas herramientas en proyectos académicos y profesionales, utilizando plataformas como Google Colab, OpenAI, y otros servicios de IA en la nube.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	14/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

La IA permite crear contenido de texto, imágenes, audio y video de forma automática, facilitando tareas como la redacción de documentos, diseño de materiales y generación de ideas.

Herramientas a explorar:

ChatGPT (OpenAI): Generación de texto para resúmenes, explicaciones, propuestas, etc.

Ideogram: Generación de imágenes a partir de descripciones textuales.

Google Docs con complementos de IA: Ayuda para la redacción y edición de documentos.

Gamma: Diseño gráfico asistido por IA para crear presentaciones y gráficos.

La importancia de usar un buen PROMPT

Todas las herramientas de IA generativas como las arriba mencionadas, requieren de un prompt que es la entrada o instrucción que se le proporciona a la herramienta para generar una respuesta o crear un contenido específico. En otras palabras, es el texto, la pregunta o el conjunto de indicaciones que el usuario proporciona para guiar a la IA en la producción de resultados, como generar texto, imágenes, música, entre otros.

Algunas recomendaciones para escribir un buen prompt son:

1. Ser específico,
2. Dar contexto,
3. Especificar el público objetivo (formal, casual, amigable, Universitario) ,
4. Si es posible dar ejemplos

Importante: siempre que se obtenga contenido a través de estas herramientas se deberá realizar una verificación de la precisión de la información antes de incluirla en trabajos académicos y profesionales. Además, es conveniente revisar las referencias de dónde tomó la información.

Herramientas para la detección del uso de Inteligencia Artificial en contenido generado

La rápida adopción de herramientas basadas en Inteligencia Artificial (IA) para la generación de texto, imágenes y otros contenidos ha planteado nuevos retos en términos de autenticidad, originalidad y ética. En respuesta, han surgido herramientas diseñadas para identificar contenido generado por IA, como ZeroGPT, Originality.ai, y otras. Estas herramientas son esenciales en contextos educativos, empresariales y de investigación, donde la validación de la originalidad y la integridad de los contenidos es fundamental.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	15/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Las herramientas de detección de IA utilizan algoritmos avanzados para analizar patrones en el contenido que podrían indicar que ha sido generado por modelos GPT:

ZeroGPT

Analiza texto para determinar la probabilidad de que haya sido generado por IA. Proporciona porcentajes de "autenticidad" en el contenido.

Originality.ai

Detecta texto generado por IA y verifica el plagio. Frecuentemente utilizado por educadores, editores y profesionales del contenido.

GPTZero

Orientado a educadores para identificar el uso de IA en tareas académicas. Ofrece un análisis detallado basado en puntuaciones de "perplejidad" y "aleatoriedad".

Actividad en casa

Creación de cuenta en github.com

Para comenzar a utilizar github, se debe hacer lo siguiente: abrimos en cualquier navegador web la dirección <https://github.com>. Damos click en “Sign Up” para crear una cuenta. (Figura 13)

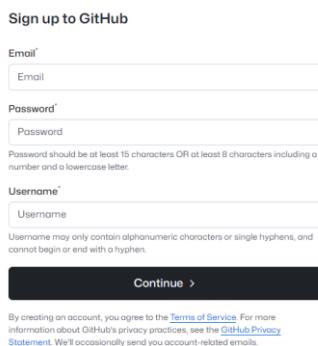
The image shows the GitHub homepage with a dark blue background. At the top, there is a navigation bar with links for Product, Solutions, Resources, Open Source, Enterprise, and Pricing. Below the navigation bar, the main headline reads "Build and ship software on a single, collaborative platform" in large, bold, white font. Underneath the headline, a subtext says "Join the world's most widely adopted AI-powered developer platform." At the bottom, there are three buttons: a white button with "Enter your email", a green button with "Sign up for GitHub", and another green button with "Try GitHub Copilot".

Figura 13. Página de inicio Github

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 16/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

Escribimos un usuario propio, un correo, una contraseña y damos click en “Create an account” “Sign up”, esperamos el correo de verificación, y verificamos nuestra cuenta. (Figura 14).

Ingresar nombre, correo, resolver un rompecabezas y crear la cuenta



The screenshot shows the 'Sign up to GitHub' form. It has three input fields: 'Email', 'Password', and 'Username'. Each field has a placeholder and a small explanatory text below it. Below the fields is a large black 'Continue >' button. At the bottom, there is a small text block with links to the 'Terms of Service' and 'GitHub Privacy Statement'.

Figura 14. Crear cuenta

Responder las siguientes preguntas: ¿Qué tipo de trabajo haces principalmente?, ¿Cuánta experiencia en programación tienes? y ¿Para qué planeas usar GitHub?, con esto se termina la configuración, ahora se debe verificar la cuenta mediante el correo electrónico ingresado

Creando nuestro primer repositorio

Damos click en el botón de “new” (Figura 15).

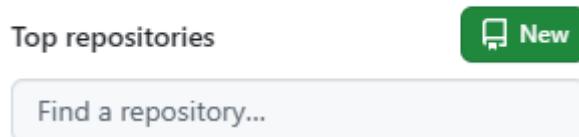
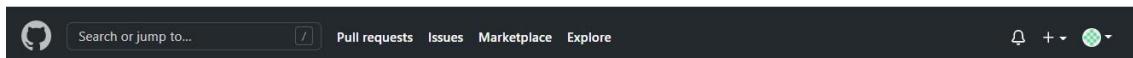


Figura 15. Iniciar proyecto

En este paso se crea el repositorio, le damos un nombre (practica1_fdp), una descripción e inicializamos un README; posteriormente damos click a “Create repository” (Figura 16).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 17/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner * <input style="border: 1px solid #ccc; padding: 2px; width: 150px; height: 20px; border-radius: 5px; margin-bottom: 5px;" type="text" value="fimecc123"/>	Repository name * <input style="width: 100%; height: 100%; border: none; outline: none;" type="text" value=""/>
Great repository names are short and memorable. Need inspiration? How about bookish-octo-enigma?	
Description (optional) <input style="width: 100%; height: 30px; border: 1px solid #ccc; border-radius: 5px;" type="text" value=""/>	
<input checked="" type="radio"/>  Public Anyone on the internet can see this repository. You choose who can commit.	
<input type="radio"/>  Private You choose who can see and commit to this repository.	

Figura 16. Crear nuevo repositorio

Creación de archivos en nuestro repositorio

Damos click en el botón de “Create new file” (Figura 17).

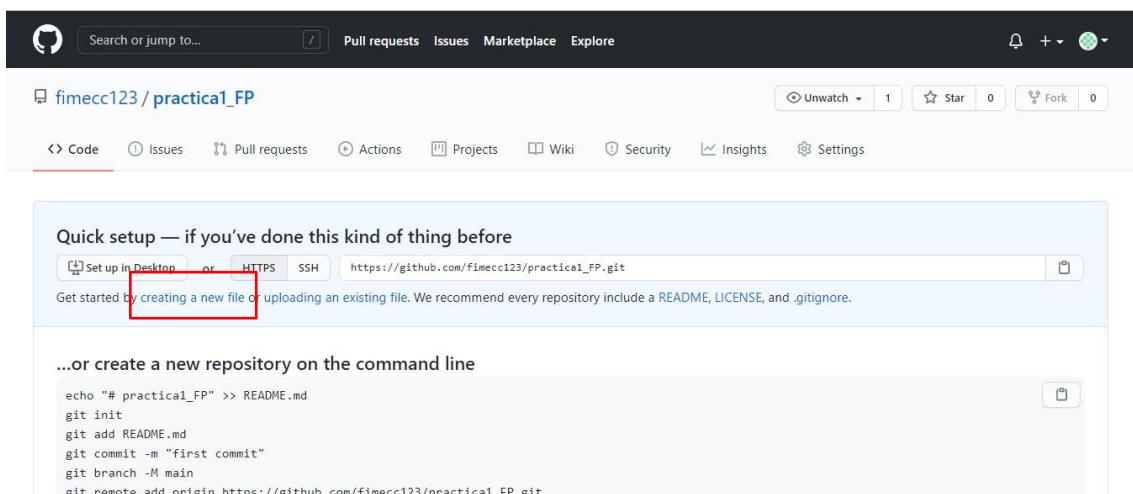


Figura 17. Crear nuevo archivo

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 18/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

Crearemos un archivo llamado Datos, y en la primera línea agregaremos nuestro nombre (Figura 18)

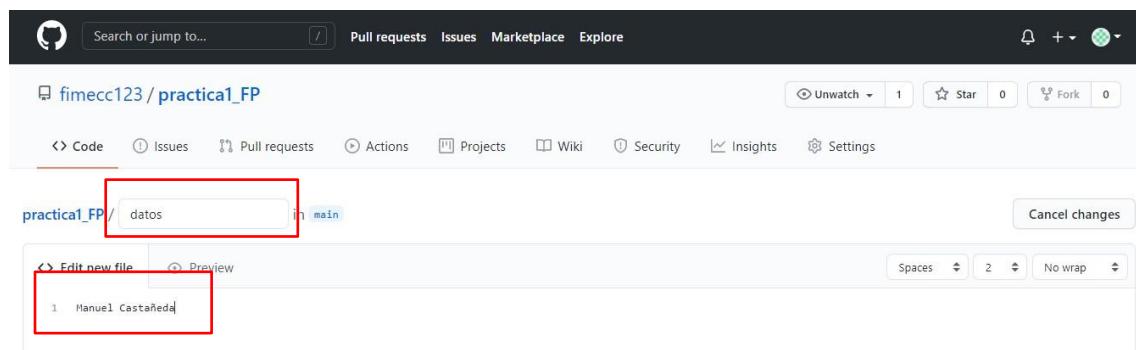


Figura 18. Modificación de archivo nuevo

En la sección de *Commit new file*, haremos una explicación del archivo creado, posteriormente damos click al botón de *Commit new file* (Figura 19)

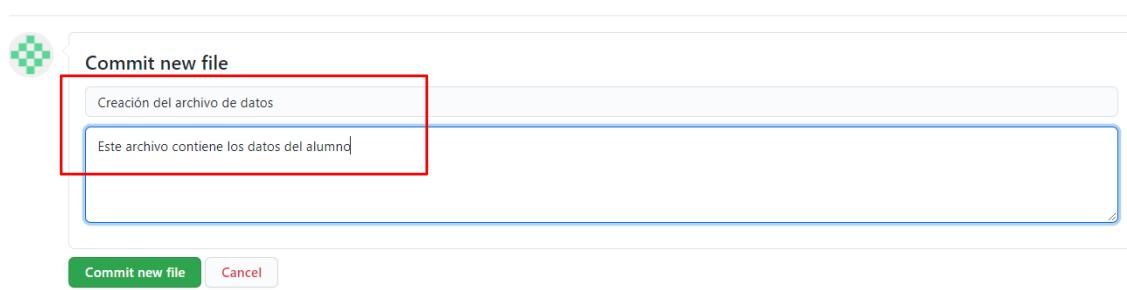
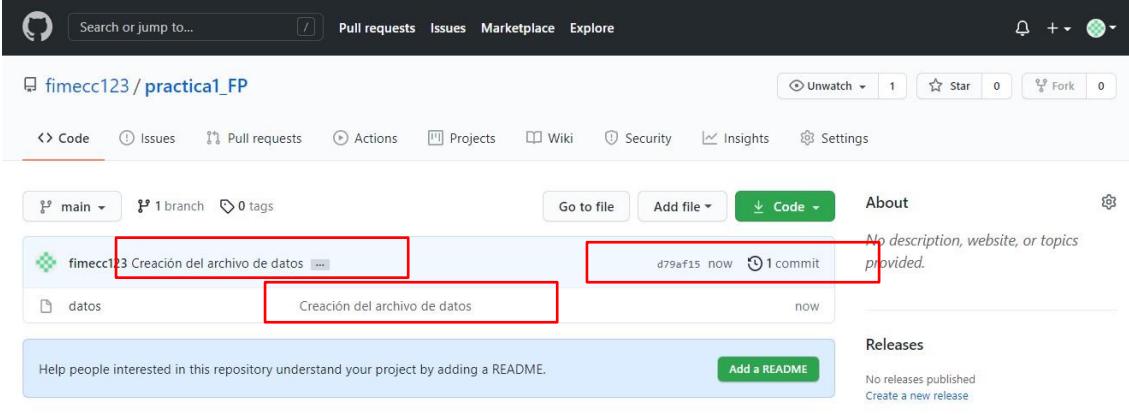


Figura 19. Commit nuevo archivo

Con esto habremos creado un nuevo archivo en nuestro repositorio, la acción de hacer *commit* es indicarle al Control de versiones que hemos terminado una nueva modificación, dando una breve explicación Al momento de hacer el *commit*, nuestro proyecto se encuentra en un nuevo estado. En la pantalla principal del repositorio se puede ver la lista de archivos en nuestro repositorio con la explicación del *commit* que agregó o modificó a ese archivo (Figura 20).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 19/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

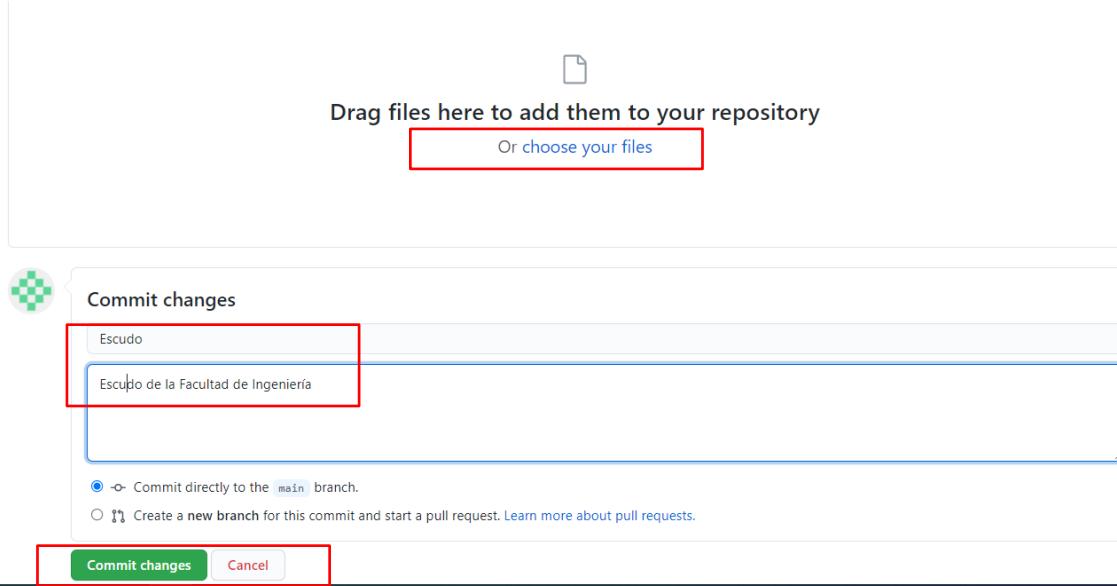


The screenshot shows a GitHub repository page. At the top, there's a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. Below the header, the repository name 'fimecc123 / practica1_FP' is shown, along with 'Unwatch' (1), 'Star' (0), and 'Fork' (0) buttons. The main content area shows a commit history. One commit, 'Creación del archivo de datos' by user 'fimecc123' at hash 'd79af15' (now) with '1 commit', is highlighted with a red box. The commit message itself is also highlighted with a red box. Other parts of the interface like the code editor, file list, and repository details are visible but not highlighted.

Figura 20. Confirmación de la modificación del archivo

Subiremos dos imágenes locales (escudo de la facultad y de la universidad) a nuestro repositorio, dando click en el botón de “Upload files”

Seleccionamos los dos archivos de nuestro equipo y hacemos el commit, explicando los archivos agregados (Figura 21).



The screenshot shows a GitHub commit dialog box. At the top, it says 'Commit changes'. Below that, there are two file attachments: 'Escudo' and 'Escudo de la Facultad de Ingeniería', both highlighted with a red box. Underneath the files, there are two radio button options: one for committing directly to the 'main' branch (selected) and another for creating a new branch and starting a pull request. At the bottom of the dialog are 'Commit changes' and 'Cancel' buttons, with the 'Commit changes' button also highlighted with a red box.

Figura 21. Cargar archivos al proyecto

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 20/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

Como se observa, un commit puede ser de uno o más archivos.

Modificando un archivo

 click en el archivo “Datos” y posteriormente hacemos click en el botón  Damos con forma de lápiz

Agregamos en la siguiente línea nuestro número de cuenta y en una línea nueva nuestro correo. Hacemos el *commit* explicando qué cambios hicimos. (Figura 22)

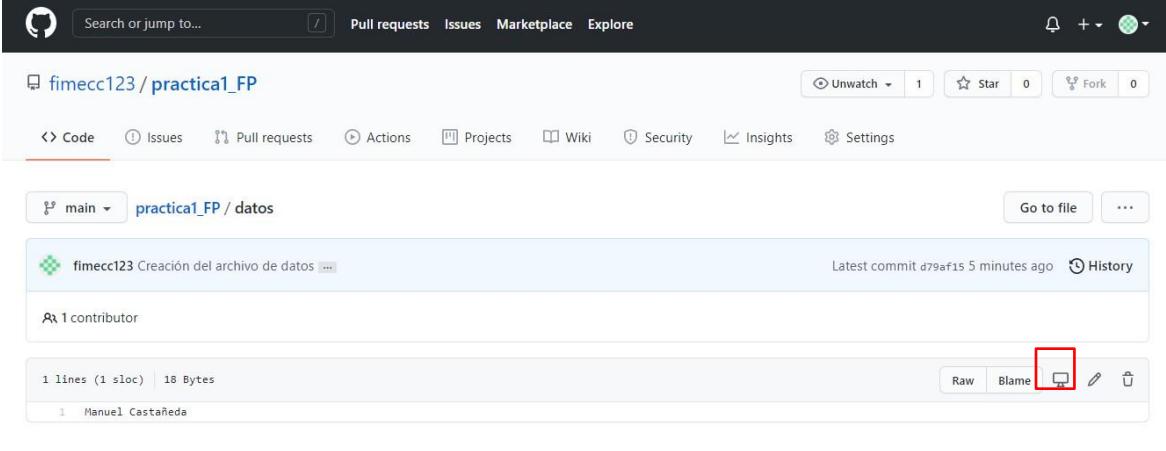


Figura 22. Editar archivo

Revisando la historia de nuestro repositorio

En la página principal del repositorio dar click a los *commits*, en este momento debe ser 4.

En esta sección se pueden revisar los cambios y estados en nuestro repositorio, Analizar qué pasa al darle click al nombre de cada *commit*.

Se pueden observar las modificaciones o adiciones que se hicieron en el *commit*. Git guarda cada estado de nuestros archivos, de esta manera siempre podemos acceder a versiones específicas.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 21/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Dar click al botón



En esta sección se puede observar el estado total del repositorio al momento de un *commit* específico. Es como una máquina del tiempo, ¡puedes regresar a versiones anteriores!

Actividad:

1. Realizar el reporte de la práctica actual.
2. Subir el archivo al repositorio creado y registrar el cambio con el *commit* “Reporte práctica 1”.
3. Mandar el link del repositorio al profesor.
Ejemplo de link: (Figura 23)

GitHub, Inc. (US) | https://github.com/labdefundamentos/practica1_fdp

Figura 23. Link del repositorio

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	22/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Referencias

1. <http://rypress.com/tutorials/git>
2. <https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de- versiones>
3. <https://www.dropbox.com/>
4. <https://scholar.google.com/>
5. <https://www.microsoft.com/en-us/research/project/academic/>
6. <https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/springerlink>
7. <https://www.researchgate.net/>
8. <https://www.base-search.net/>

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	23/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 02: GNU/Linux



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	Ing. Manuel Enrique Castañeda Castañeda	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	24/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 02: GNU/Linux

Objetivo:

El alumno identificará al sistema operativo como una parte esencial de un sistema de cómputo. Explorará un sistema operativo GNU/Linux con el fin de conocer y utilizar sus comandos básicos.

Actividades:

- Iniciar sesión en un sistema operativo GNU/Linux y abrir una “terminal”
- Utilizar los comandos básicos para navegar por el sistema de archivos.
- Emplear comandos para manejo de archivos.

Introducción

El Sistema Operativo es el conjunto de programas y datos que administra los recursos tanto de hardware (dispositivos) como de software (programas y datos) de un sistema de cómputo y/o comunicación. Además, funciona como interfaz entre la computadora y el usuario o aplicaciones.

En la actualidad existen diversos sistemas operativos; por ejemplo, para equipos de cómputo están Windows, Linux, Mac OS entre otros. Para el caso de dispositivos móviles se encuentran Android, IOS, Windows Phone, etcétera. Cada uno de ellos tiene diferentes versiones y distribuciones que se ajustan a los diversos equipos de cómputo y comunicación en los que trabajan.

Los componentes de un sistema operativo, de forma general, son:

- Gestor de memoria,
- Administrador y planificador de procesos,
- Sistema de archivos y
- Administración de E/S.

Comúnmente, estos componentes se encuentran en el kernel o núcleo del sistema operativo.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 25/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

En cuanto a la Interfaz con el usuario, las hay de tipo texto y de tipo gráfico. En la actualidad, es común trabajar con la interfaz gráfica ya que facilita mucho seleccionar la aplicación a utilizar; inclusive esta selección se hace “tocando la pantalla” (técnica touch).

Sin embargo, cuando se desarrollan proyectos donde se elaborarán documentos y programas es necesario el uso de dispositivos de entrada y salida (hardware) y aplicaciones en modo texto (software).

Sistema Operativo Linux

Linux es un sistema operativo tipo Unix de libre distribución para computadoras personales, servidores y estaciones de trabajo.

El sistema está conformado por el núcleo (kernel) y un gran número de programas y bibliotecas. Muchos programas y bibliotecas han sido posibles gracias al proyecto GNU, por lo mismo, se conoce a este sistema operativo como GNU/Linux.

Software libre

Un software libre es aquel que se puede adquirir de manera gratuita, es decir, no se tiene que pagar algún tipo de licencia a alguna casa desarrolladora de software por el uso de éste.

Además, que un software sea libre implica también que el software viene acompañado del código fuente, es decir, se pueden realizar cambios en el funcionamiento del sistema si así se desea.

Linux se distribuye bajo la Licencia Pública General de GNU por lo tanto, el código fuente tiene que estar siempre accesible y cualquier modificación o trabajo derivado debe tener esta licencia.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 26/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Licencia GNU

La Licencia Pública General de GNU o GNU General Public License (GNU GPL) es una licencia creada por la Free Software Foundation en 1989 y está orientada principalmente a proteger la libre distribución, modificación y uso de software.

Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

Kernel de GNU/Linux

El kernel o núcleo de linux se puede definir como el corazón del sistema operativo. Es el encargado de que el software y el hardware del equipo se puedan comunicar. Sus componentes son los que se mencionaron en la introducción de esta práctica.



Figura 1: Capas que componen al sistema operativo GNU/Linux.

De la Figura 1, se puede observar que entre el kernel y las aplicaciones existe una capa que permite al usuario comunicarse con el sistema operativo y en general con la computadora, a través de programas que ya vienen instalados con la distribución de Linux (Debian, Ubuntu, Fedora, etc.) y trabajan ya sea en modo gráfico o en modo texto. Uno de estos programas es el Shell.

La estructura de Linux para el almacenamiento de archivos es de forma jerárquica; por lo que la carpeta o archivo base es “root” (raíz) la cual se representa con una diagonal (/). De este archivo raíz, parten todos los demás. Los archivos pueden ser carpetas (directorios), de datos, aplicaciones, programas, etc. (Figura 2)



Manual de prácticas del Laboratorio de Fundamentos de programación

Código:	MADO-17
Versión:	05
Página	27/187
Sección ISO	8.3
Fecha de emisión	22 / enero / 2025

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

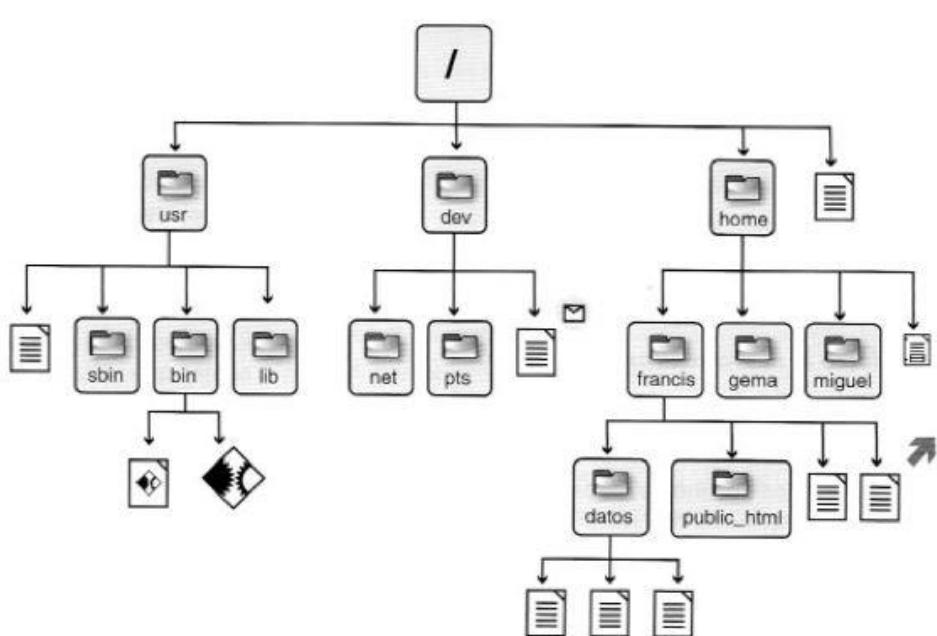


Figura 2: Una parte del sistema de archivos jerárquico en GNU/Linux.

Interfaz de línea de comandos (CLI) o shell de GNU/Linux

El Shell de GNU/Linux permite introducir órdenes (comandos) y ejecutar programas en el sistema operativo. Todas las órdenes de UNIX/Linux son programas que están almacenados en el sistema de archivos y a los que llamamos comandos, por lo tanto, todo en GNU/Linux se puede controlar mediante comandos.

Comandos básicos

Para trabajar en Linux utilizando comandos, se debe abrir una "terminal" o "consola" que es una ventana donde aparece la "línea de comandos" en la cual se escribirá la orden o comando. La terminal permite un mayor grado de funciones y configuración de lo que queremos hacer con una aplicación o acción en general respecto a un entorno gráfico.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 28/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

El proceso de abrir una terminal varía dependiendo del entorno gráfico. Por lo general hay un área de “aplicaciones” donde se selecciona *terminal* o *consola*. (Figura 3)

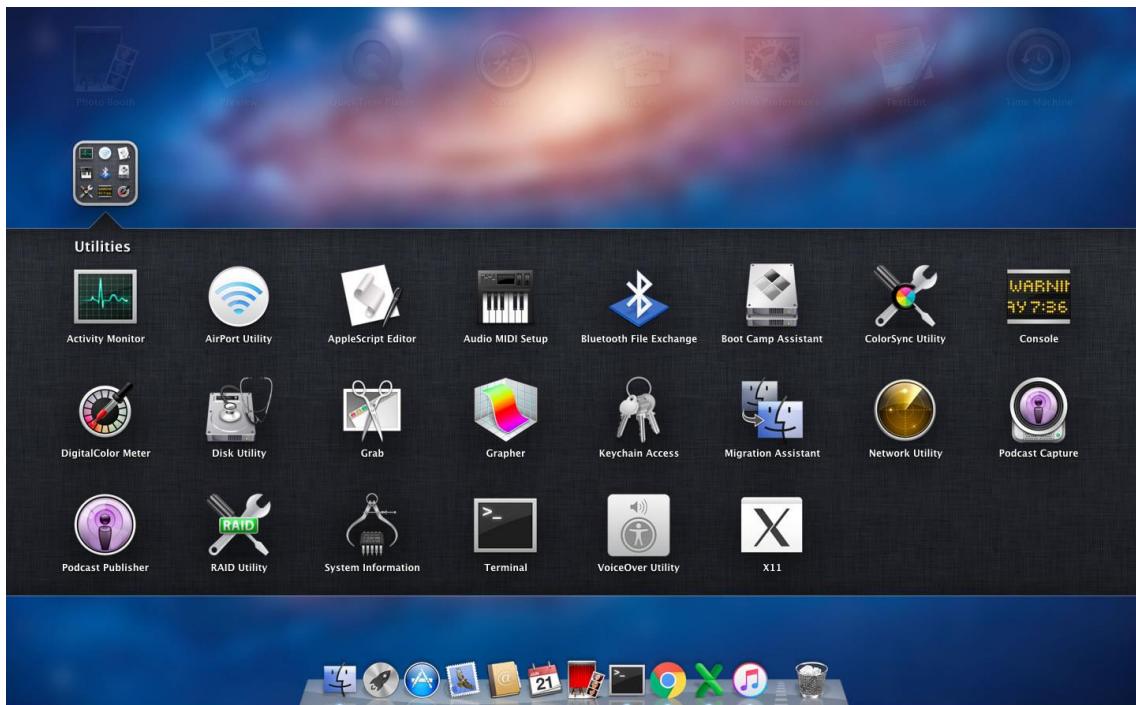


Figura 3: Utilidades entorno gráfico

O bien en el ícono de aplicaciones en la línea de “buscar” escribir “terminal” si es que no está a la vista el ícono de terminal.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	29/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Una vez teniendo una terminal abierta, estamos listos para introducir comandos. (Figura 4)

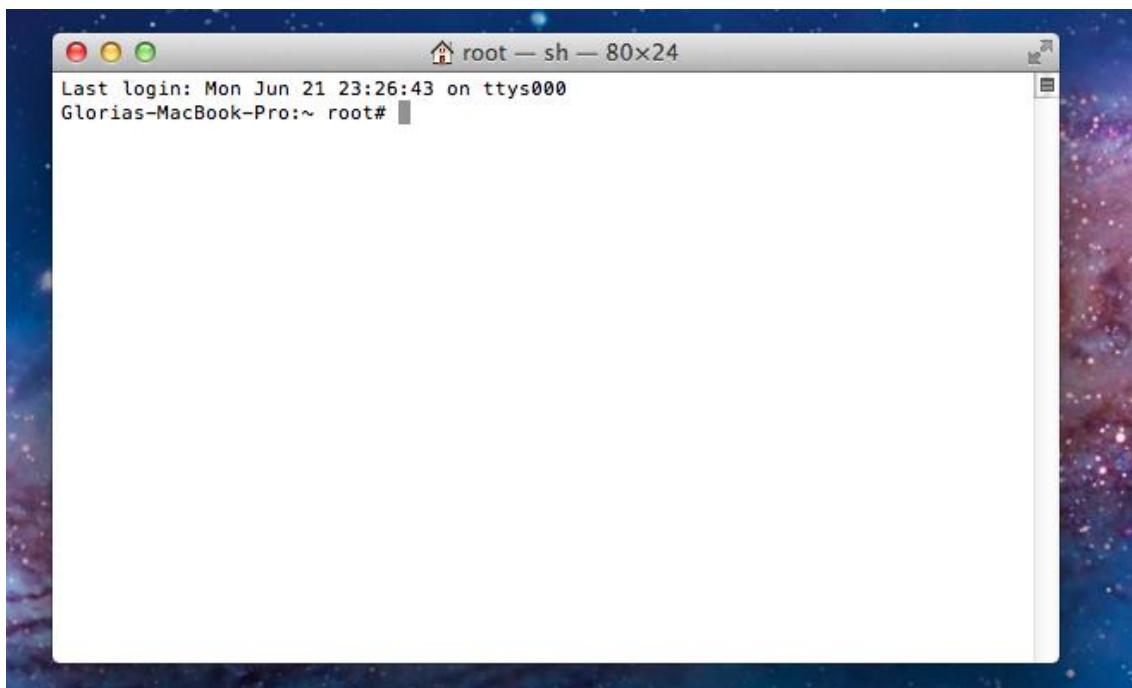


Figura 4: Terminal

La sintaxis que siguen los comandos es la siguiente:

comando [-opciones] [argumentos]

Esto es, el nombre del comando, seguido de algunas banderas (opciones) para modificar la ejecución de este y, al final, se puede incluir un argumento (ruta, ubicación, archivo, etcétera) dependiendo del comando. Tanto las opciones como los argumentos son opcionales.

Ejemplo (comando ls)

El comando *ls* permite listar los elementos que existen en alguna ubicación del sistema de archivos de Linux. Por defecto lista los elementos que existen en la ubicación actual; Linux nombra la ubicación actual con un punto (.) por lo que

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 30/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

`ls`

y

`ls .`

realizan exactamente lo mismo.

El comando `ls` realiza acciones distintas dependiendo de las banderas que utilice, por ejemplo, si se utiliza la opción `l` se genera un listado largo de la ubicación actual:

`ls -l`

Es posible listar los elementos que existen en cualquier ubicación del sistema de archivos, para ello hay que ejecutar el comando especificando como argumento la ubicación donde se desean listar los elementos. Si queremos ver los archivos que se encuentran en la raíz, usamos:

`ls /`

Para ver los usuarios del equipo local, revisamos el directorio `home` que parte de la raíz (/):

`ls /home`

Tanto las opciones como los argumentos se pueden combinar para generar una ejecución más específica:

`ls -l /home`

GNU/Linux proporciona el comando `man`, el cual permite visualizar la descripción de cualquier comando, así como la manera en la que se puede utilizar.

`man ls`

Antes de revisar otros comandos, es importante aprender a “navegar” por el sistema de archivos de Linux en modo texto. Basándonos en la Figura 2 de esta práctica, si deseamos ver la lista de los archivos del directorio `usr`, podemos escribir el comando:

`ls /usr`

Esto es, el argumento se inicia con / indicando que es el directorio raíz, seguido de `usr` que es el nombre del directorio. Cuando especificamos la ubicación de un archivo partiendo de la raíz, se dice que estamos indicando la “ruta absoluta” del archivo.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	31/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Existe otra forma de especificar la ubicación de un archivo, esto es empleando la “ruta relativa”.

Si bien el punto (.) es para indicar la ubicación actual, el doble punto (..) se utiliza para referirse al directorio “padre”. De esta forma si deseamos listar los archivos que dependen de mi directorio padre se escribe el siguiente comando:

```
ls ..  
o  
ls ../
```

Se pueden utilizar varias referencias al directorio padre para ir navegando por el sistema de archivos, de tal manera que se realice la ubicación de un archivo a través de una ruta relativa. De la Figura 2, si nuestra cuenta depende de *home*, la ruta relativa para listar los archivos del directorio *usr* es:

```
ls ../../usr
```

Con los primeros dos puntos se hace referencia al directorio *home*, con los siguientes dos puntos se refiere al directorio raíz, y finalmente se escribe el nombre del directorio *usr*.

Ejemplo (comando touch)

El comando *touch* permite crear un archivo de texto, su sintaxis es la siguiente:

```
touch nombre_archivo[.ext]
```

En GNU/Linux no es necesario agregar una extensión al archivo creado, sin embargo, es recomendable hacerlo para poder identificar el tipo de archivo creado.

Ejemplo (comando mkdir)

El comando *mkdir* permite crear una carpeta, su sintaxis es la siguiente:

```
mkdir nombre_carpeta
```

Para crear una carpeta en nuestra cuenta, que tenga como nombre “tareas” se escribe el siguiente comando:

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	32/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

```
mkdir tareas
```

Ejemplo (comando cd)

El comando *cd* permite ubicarse en una carpeta, su sintaxis es la siguiente:

```
cd nombre_carpeta
```

Por lo que si queremos situarnos en la carpeta “tareas” creada anteriormente, se escribe el comando:

```
cd tareas
```

Ahora, si deseamos situarnos en la carpeta de inicio de nuestra cuenta, que es la carpeta padre, escribimos el comando:

```
cd ..
```

Ejemplo (comando pwd)

El comando *pwd* permite conocer la ubicación actual(ruta), su sintaxis es la siguiente:

```
pwd
```

Ejemplo (comando find)

El comando *find* permite buscar un elemento dentro del sistema de archivos, su sintaxis es la siguiente:

```
find . -name cadena_buscar
```

Al comando *find* hay que indicarle en qué parte del sistema de archivos va a iniciar la búsqueda. En el ejemplo anterior la búsqueda se inicia en la posición actual (uso de *.*). Además, utilizando la bandera *-name* permite determinar la cadena a buscar (comúnmente es el nombre de un archivo).

Si queremos encontrar la ubicación del archivo *tareas*, se escribe el siguiente comando:

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	33/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada	

```
find . -name tareas
```

Ejemplo (comando clear)

El comando *clear* permite limpiar la consola o terminal, su sintaxis es la siguiente:

```
clear
```

Ejemplo (comando cp)

El comando *cp* permite copiar un archivo, su sintaxis es la siguiente:

```
cp archivo_origen archivo_destino
```

Si queremos una copia del archivo *datos.txt* con nombre *datosViejos.txt* en el mismo directorio, entonces se escribe el comando

```
cp datos.txt datosViejos.txt
```

Ahora, si requerimos una copia de un archivo que está en la carpeta padre en la ubicación actual y con el mismo nombre, entonces podemos emplear las rutas relativas de la siguiente forma:

```
cp ../archivo_a_copiar .
```

Es muy importante indicar como archivo destino al punto (.) para que el archivo de copia se ubique en el directorio actual.

Ejemplo (comando mv)

El comando *mv* mueve un archivo de un lugar a otro, en el sistema de archivos; su sintaxis es la siguiente:

```
mv ubicación_origen/archivo ubicación_destino
```

El comando mueve el archivo desde su ubicación origen hacia la ubicación deseada(destino).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	34/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Si queremos que un archivo que está en la carpeta padre, reubicarlo en el directorio actual y con el mismo nombre, entonces podemos emplear las rutas relativas de la siguiente forma:

```
mv ../../archivo_a_reubicar .
```

Este comando también puede ser usado para cambiar el nombre de un archivo, simplemente se indica el nombre actual del archivo y el nuevo nombre:

```
mv nombre_actual_archivo nombre_nuevo_archivo
```

Ejemplo (comando rm)

El comando *rm* permite eliminar un archivo o un directorio, su sintaxis es la siguiente:

```
rm nombre_archivo
rm nombre_carpetas
```

Cuando la carpeta que se desea borrar contiene información, se debe utilizar la bandera *-f* para forzar la eliminación. Si la carpeta contiene otras carpetas, se debe utilizar la opción *-r*, para realizar la eliminación recursiva.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	35/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía

- Óscar Vicente Huguet Soriano, Sonia Doménech Gómez. Introducción a Linux. [Figura 1]. Consulta: Junio de 2015. Disponible en:
http://mural.uv.es/oshuso/81_introduccin_a_linux.html
- Pablo Delgado. Integración de sistemas. Linux y su sistema gestor de ficheros (descripciones). [Figura 2]. Consulta agosto de 2016. Disponible en:
<http://todobytes.es/2014/09/integracion-de-sistemas-linux-y-su-sistema-gestor-de-ficheros-descripciones/>

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	36/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 03: Solución de problemas y Algoritmos.



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes Ing. Maricela Castañeda Perdomo	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	37/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 03: Solución de problemas y Algoritmos.

Objetivo:

El alumno elaborará algoritmos correctos y eficientes en la solución de problemas siguiendo las etapas de Análisis y Diseño pertenecientes al Ciclo de vida del software.

Actividades:

- A partir del enunciado de un problema, identificar el conjunto de entrada y el conjunto de salida.
- Elaborar un algoritmo que resuelva un problema determinado (dado por el profesor), identificando los módulos de entrada, de procesamiento y de salida.

Introducción

Un problema informático se puede definir como el conjunto de instancias al cual corresponde un conjunto de soluciones, junto con una relación que asocia para cada instancia del problema un subconjunto de soluciones (posiblemente vacío).

Para poder solucionar un problema nos apoyamos en la Ingeniería de Software que de acuerdo con la IEEE se define como "La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software". Por lo que el uso y establecimiento de principios de ingeniería sólidos, son básicos para obtener un software que sea económicamente fiable y funcione eficientemente.

La Ingeniería de Software provee métodos que indican cómo generar software. Estos métodos abarcan una amplia gama de tareas:

- Planeación y estimación del proyecto.
- Análisis de requerimientos del sistema y software.
- Diseño de la estructura de datos, la arquitectura del programa y el procedimiento algorítmico.
- Codificación.
- Pruebas y mantenimiento (validación y verificación).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 38/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ciclo de vida del software

La ISO (International Organization for Standardization) en su norma 12207 define al ciclo de vida de un software como:

“Un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando desde la definición hasta la finalización de su uso”

La Figura 1 muestra las actividades, también nombradas *etapas*, que se realizan en el ciclo de vida del software.



Figura 1: Ciclo de vida del software.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 39/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Solución de problemas

Dentro del ciclo de vida del software, en el análisis se busca comprender la necesidad, es decir, entender el problema.

El análisis es el proceso para averiguar qué es lo que requiere el usuario del sistema de software (análisis de requisitos). Esta etapa permite definir las necesidades de forma clara y concisa (especificación de requisitos).

Por lo tanto, la etapa del análisis consiste en conocer qué es lo que está solicitando el usuario. Para ello es importante identificar dos grandes conjuntos dentro del sistema: el conjunto de entrada y el conjunto de salida. En la Figura 2 se muestra de manera esquemática los componentes del sistema.

El **conjunto de entrada** está compuesto por todos aquellos datos que pueden alimentar al sistema.

El **conjunto de salida** está compuesto por todos los datos que el sistema regresará como resultado del proceso. Estos datos se obtienen a partir de los datos de entrada.

La unión del conjunto de entrada y el conjunto de salida forman lo que se conoce como el dominio del problema, es decir, los valores que el problema puede manejar.

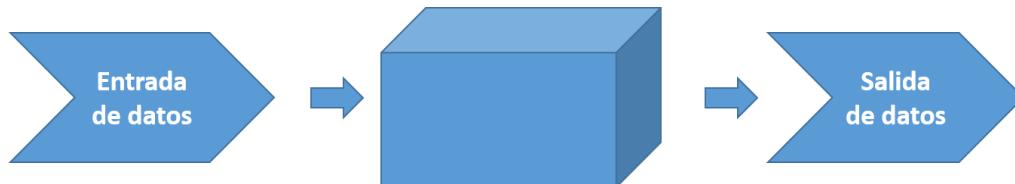


Figura 2. Sistema

La etapa de análisis es crucial para la creación de un software de calidad, ya que si no se entiende qué es lo que se desea realizar, no se puede generar una solución. Sin embargo, es común caer en ambigüedades debido al mal entendimiento de los requerimientos iniciales.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 40/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Ejemplo 1

PROBLEMA: Determinar si un número dado es positivo o negativo.

RESTRICIONES: El número no puede ser cero.

DATOS DE ENTRADA: El conjunto de datos de entrada E está compuesto por el conjunto de los números reales, excepto el cero.

$$E \subset \mathbb{R}^1, \text{ donde} \\ \text{num} \in E \text{ de } (-\infty, \infty) - \{0\}$$

NOTA: \mathbb{R}^1 representa al conjunto de números reales de una dimensión.

DATOS DE SALIDA: El conjunto de salida S está compuesto por dos valores mutuamente excluyentes.

Un posible conjunto de salida son los valores enteros 0 o 1, donde 0 indica que el valor es positivo y 1 indica el valor es negativo.

$$\text{res} = 0, \text{ si num } (0, \infty), \text{ res} = 1, \text{ si num } (-\infty, 0)$$

Otro posible conjunto de datos de salida son los valores booleanos o lógicos *Verdadero* o *Falso*, donde *Verdadero* indica que el valor es positivo y *Falso* indica que el valor es negativo; o viceversa, *Verdadero* indica que el valor es negativo y *Falso* indica que el valor es positivo.

La Figura 3 muestra los componentes del sistema para el ejemplo 1.

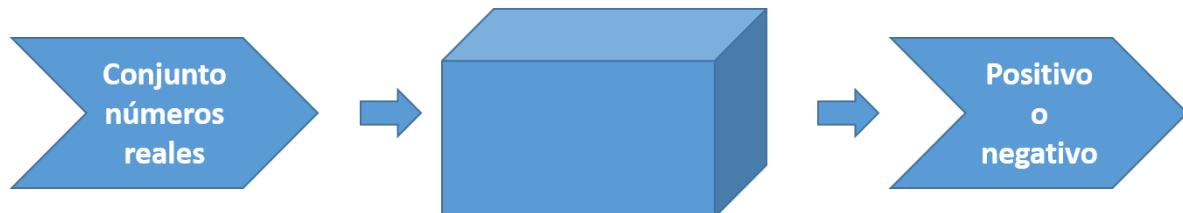


Figura 3. Ejemplo 1

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 41/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Ejemplo 2

PROBLEMA: Obtener el mayor de dos números diferentes dados.

RESTRICIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: El conjunto de entrada E está dividido en dos subconjuntos E y E'. El primer número (num1) puede adquirir cualquier valor del conjunto de los números reales ($E = (-\infty, \infty)$), sin embargo, el conjunto de entrada del segundo número (num2) es un subconjunto de E, es decir, E' está compuesto por el conjunto de los números reales excepto num1 ($E' = (-\infty, \infty) - \{num1\}$).

$$\begin{aligned} E, E' &\subset \mathbb{R}^1, \text{ donde} \\ \text{num1} &\in E \text{ de } (-\infty, \infty), \\ \text{num2} &\in E' \text{ de } (-\infty, \infty) - \{\text{num1}\} \end{aligned}$$

DATOS DE SALIDA: El conjunto de datos de salida S que puede tomar el resultado r está compuesto por el conjunto de los números reales.

$$S \subset \mathbb{R}^1, \text{ donde } r \in S \text{ de } (-\infty, \infty)$$

La Figura 4 representa los componentes del sistema del ejemplo 2.

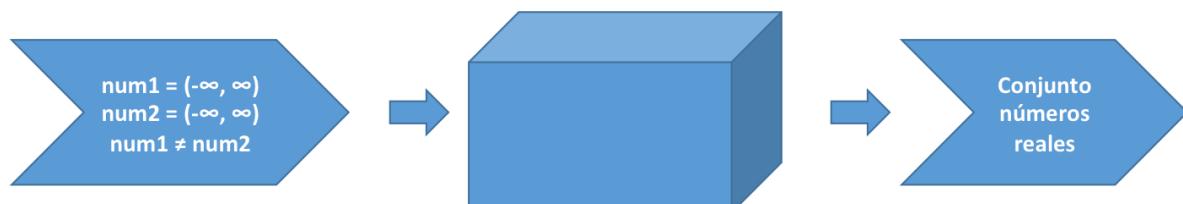


Figura 4. Ejemplo 2

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 42/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Ejemplo 3

PROBLEMA: Obtener el factorial de un número dado. El factorial de un número está dado por el producto de ese número por cada uno de los números anteriores hasta llegar a 1. El factorial de 0 (0!) es 1:

$$n! = n * (n-1)!$$

RESTRICIONES: El número de entrada debe ser entero positivo o cero. No puede ser negativo.

DATOS DE ENTRADA: El conjunto de entrada E está dado por el conjunto de los números naturales o por el cero.

$$\begin{aligned} E &\subset \mathbb{N}^1, \text{ donde} \\ \text{num} &\in E \text{ de } [1, \infty) \cup \{0\} \end{aligned}$$

DATOS DE SALIDA: El conjunto de salida S está conformado por el conjunto de los números naturales.

$$\begin{aligned} S &\subset \mathbb{N}^1; \text{ donde} \\ \text{res} &\in S \text{ de } [1, \infty) \end{aligned}$$

Los componentes del sistema de este ejemplo se muestran en la Figura 5.

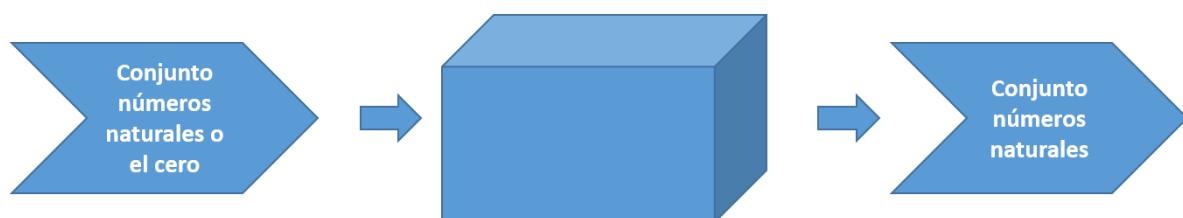


Figura 5. Ejemplo 3

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	43/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Algoritmos

Una vez realizado el análisis, es decir, ya que se entendió qué es lo que está solicitando el usuario y ya identificado el conjunto de entrada y el conjunto de salida, se puede proceder al diseño de la solución, esto es, a la generación del algoritmo.

Dentro del ciclo de vida del software, la creación de un algoritmo se encuentra en la etapa de diseño. Ver figura 1.

Durante el diseño se busca proponer una o varias alternativas viables para dar solución al problema y con base en esto tomar la mejor decisión para iniciar la construcción.

Un problema matemático es computable si éste puede ser resuelto, en principio, por un dispositivo computacional.

La teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo.

Un algoritmo se define como un conjunto de reglas, expresadas en un lenguaje específico, para realizar alguna tarea en general, es decir, un conjunto de pasos, procedimientos o acciones que permiten alcanzar un resultado o resolver un problema. Estas reglas o pasos pueden ser aplicados un número ilimitado de veces sobre una situación particular.

Un algoritmo es la parte más importante y durable de las ciencias de la computación debido a que éste puede ser creado de manera independiente tanto del lenguaje como de las características físicas del equipo que lo va a ejecutar.

Las principales características con las que debe cumplir un algoritmo son:

- Preciso: Debe indicar el orden de realización de paso y no puede tener ambigüedad.
- Definido: Si se sigue dos veces o más se obtiene el mismo resultado.
- Finito: Tiene fin, es decir tiene un número determinado de pasos.
- Correcto: Cumplir con el objetivo.
- Debe tener al menos una salida y ésta debe de ser perceptible
- Debe ser sencillo y legible
- Eficiente: Realizarlo en el menor tiempo posible

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página: 44/187 Sección ISO: 8.3 Fecha de emisión: 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

- Eficaz: Que produzca el efecto esperado

Por tanto, un buen algoritmo debe ser correcto (cumplir con el objetivo) y eficiente (realizarlo en el menor tiempo posible), además de ser entendible para cualquier persona.

Las actividades a realizar en la elaboración de un algoritmo para obtener una solución a un problema de forma correcta y eficiente se muestran en la Figura 6.

Resultados del análisis del problema

- Con qué datos se cuenta, cuáles son necesarios como valores de entrada, qué restricciones deben considerarse, cómo debe ser la salida para que el problema se resuelva.

Construcción del algoritmo

- Se refiere a la descripción detallada de los pasos que deben seguirse para resolver el problema.

Verificación del algoritmo

- Consiste en el seguimiento del mismo, empleando datos que son representativos del problema que se desea resolver (esto se conoce como prueba de escritorio).

Figura 6. Elaboración de un algoritmo

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	45/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	
		22 / enero / 2025	
			Área/Departamento:
			Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Un algoritmo consta de 3 módulos básicos (Figura 7):

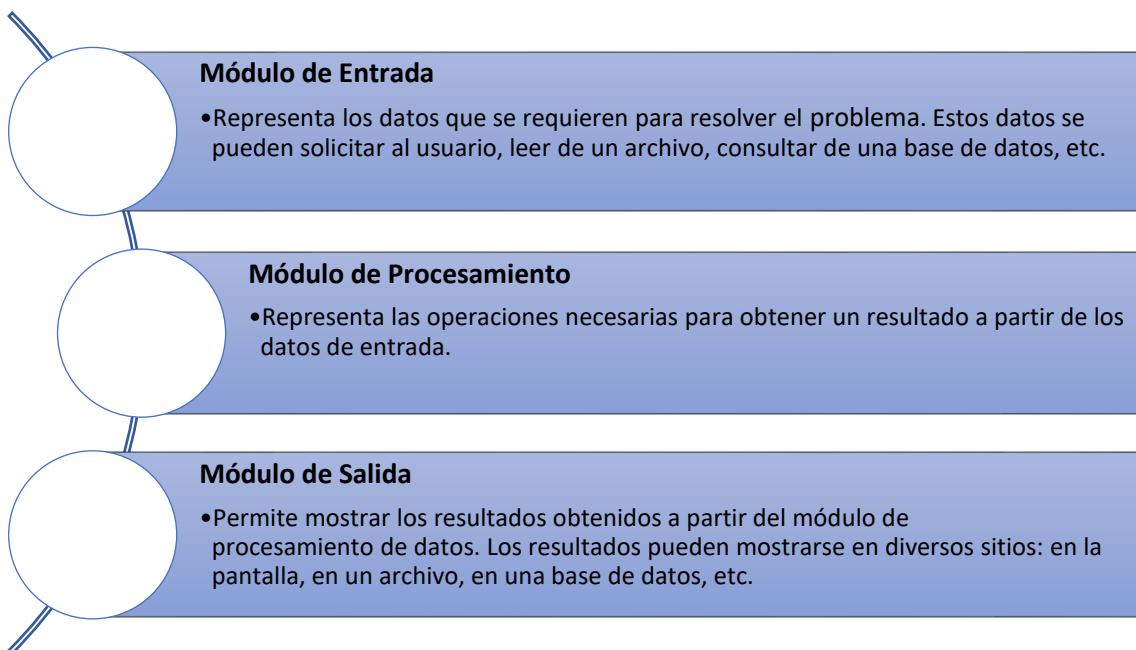


Figura 7. Módulos básicos del algoritmo

Variables

Un algoritmo requiere del uso de *variables* porque guardan el valor (numérico o no numérico) de los datos de entrada; también suelen ser utilizadas para almacenar datos generados en el proceso y datos de salida. Es a través del valor de dichas variables que el algoritmo puede fluir en la secuencia de pasos a seguir.

Ejemplos de algoritmos

Ejemplo 1

PROBLEMA: Determinar si un número dado es positivo o negativo.

RESTRICIONES: El número no puede ser cero.

DATOS DE ENTRADA: Número real.

DATOS DE SALIDA: La indicación de si el número es positivo o negativo

DOMINIO: Todos los números reales.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 46/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

SOLUCIÓN:

1. Solicitar un número real y almacenarlo en una variable
2. Si el número ingresado es cero, se regresa al punto 1.
3. Si el número ingresado es diferente de cero, se validan las siguientes condiciones:
 - 3.1 Si el número ingresado es mayor a 0 se puede afirmar que el número es positivo.
 - 3.2 Si el número ingresado es menor a 0 se puede afirmar que el número es negativo.

Prueba de escritorio

El diseño de la solución de un problema implica la creación del algoritmo y la validación de este. La validación se suele realizar mediante una *prueba de escritorio*.

Una prueba de escritorio es una matriz formada por los valores que van adquiriendo cada una de las variables del algoritmo en cada iteración. Una iteración es el número de veces que se ejecuta una parte del algoritmo y permite ver los valores que van adquiriendo las variables en cada repetición.

Para el ejemplo en cuestión, la prueba de escritorio quedaría de la siguiente manera (considerando a X como la variable que almacena el número solicitado):

Iteración	X	Salida
1	5	El número es positivo

Iteración	X	Salida
1	-29	El número es negativo

Iteración	X	Salida
1	0	-
2	0	-
3	0	-
4	100	El número es positivo

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	47/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Ejemplo 2

PROBLEMA: Obtener el mayor de dos números dados.

RESTRICIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: Dos números reales.

DATOS DE SALIDA: La escritura del número más grande.

DOMINIO: Todos los números reales.

SOLUCIÓN:

1. Solicitar un primer número real y almacenarlo en una variable.
2. Solicitar un segundo número real y almacenarlo en otra variable.
3. Si el segundo número real es igual al primer número real, se regresa al punto 2.
4. Si el segundo número real es diferente al primer número real, se validan las siguientes condiciones:
 - 4.1 Si se cumple con la condición de que el primer número es mayor al segundo número, entonces se puede afirmar que el primer número es el mayor de los números.
 - 4.2 Si se cumple con la condición de que el segundo número es mayor al primer número, entonces se puede afirmar que el segundo número es el mayor de los números.

Prueba de escritorio. La variable X almacena el primer número solicitado, y la variable Y almacena el segundo:

Iteración	X	Y	Salida
1	5	6	El segundo número es el mayor de los números

Iteración	X	Y	Salida
1	-99	-222.2	El primer número es el mayor de los números

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	48/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Iteración	X	Y	Salida
1	15	15	-
2	15	15	-
3	15	15	-
4	15	10	El primer número es el mayor de los números

Ejemplo 3

PROBLEMA: Obtener el factorial de un número dado.

RESTRICIONES: El número de entrada debe ser entero y no puede ser negativo.

Nota: El factorial de un número está dado por el producto de ese número por cada uno de los números anteriores hasta llegar a 1. El factorial de 0 ($0!$) es 1.

DATOS DE ENTRADA: Número entero.

DATOS DE SALIDA: El factorial del número.

DOMINIO: Todos los números naturales y el cero.

SOLUCIÓN:

1. Solicitar un número entero y almacenarlo en una variable.
2. Si el número entero es menor a cero regresar al punto 1.
3. Si el número entero es mayor o igual a cero se crea una variable entera *contador* que inicie en 2 y una variable entera *factorial* que inicie en 1.
4. Si la variable *contador* es menor o igual al número entero de entrada se realiza lo siguiente:
 - 4.1 Se multiplica el valor de la variable *contador* con el valor de la variable *factorial*. El resultado se almacena en la variable *factorial*.
 - 4.2 Se incrementa en uno el valor de la variable *contador*.
 - 4.3 Regresar al punto 4.
5. Si la variable *contador* no es menor o igual al número entero de entrada se muestra el resultado almacenado en la variable *factorial*.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	49/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Prueba de escritorio. (La variable X almacena el número entero del que se calculará el factorial)

Iteración	X	factorial	contador	Salida
1	0	1	2	El factorial es 1

Iteración	X	factorial	contador	Salida
1	-2	1	2	-
2	-67	1	2	-
3	5	1	2	-
4	5	2	3	-
5	5	6	4	-
6	5	24	5	-
7	5	120	6	El factorial es 120

Iteración	X	factorial	contador	Salida
1	7	1	2	-
2	7	2	3	-
3	7	6	4	-
4	7	24	5	-
5	7	120	6	-
6	7	720	7	-
7	7	5040	8	El factorial es 5040

Si bien se ha ejemplificado la construcción de algoritmos que resuelven problemas numéricos, la construcción de algoritmos no se limita a resolver sólo a este tipo de problemas. A continuación, se presentan ejercicios que ponen a prueba al ejecutor el buen seguimiento del algoritmo para obtener un resultado correcto.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	50/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Ejercicio 1

PROBLEMA: Seguir el algoritmo para obtener una figura

ENTRADA: Hoja tamaño carta en limpio, regla y lápiz.

SALIDA: Figura correcta.

Algoritmo

1. Dibuja una V invertida. Empieza desde el lado izquierdo, sube, y baja hacia el lado derecho, no levantes el lápiz.
2. Ahora dibuja una línea en ángulo ascendente hacia la izquierda. Debe cruzar la primera línea más o menos a 1/3 de la altura. Todavía no levantes el lápiz del papel.
3. Ahora, dibuja una línea horizontal hacia la derecha. Debe cruzar la V invertida más o menos a 2/3 de la altura total. Sigue sin levantar el lápiz.
4. Dibuja una línea en un ángulo descendente hasta el punto de inicio. Las líneas deben unirse.
5. Ahora ya puedes levantar el lápiz del papel. Has terminado la estrella de 5 puntas.

Ejercicio 2

PROBLEMA: Seguir el algoritmo para obtener una figura

ENTRADA: Hoja tamaño carta en limpio, regla y lápiz.

SALIDA: Figura correcta.

Algoritmo

1. Empieza dibujando un círculo con un compás. Coloca un lápiz en el compás. Coloca la punta del compás en el centro de una hoja de papel.
2. Ahora gira el compás, mientras mantienes la punta apoyada en el papel. El lápiz dibujará un círculo perfecto alrededor de la punta del compás.
3. Marca un punto en la parte superior del círculo con el lápiz. Ahora, coloca la punta del compás en la marca. No cambies el radio del compás con que hiciste el círculo.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	51/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada	

4. Gira el compás para hacer una marca en el propio círculo hacia la izquierda. Haz una marca también en el lado derecho.
5. Ahora, coloca la punta del compás en uno de los puntos. Recuerda no cambiar el radio del compás. Haz otra marca en el círculo.
6. Continúa moviendo la punta del compás a las otras marcas, y continúa hasta que tengas 6 marcas a la misma distancia unas de otras. Ahora, ya puedes dejar tu compás a un lado.
7. Usa una regla para crear un triángulo que empiece en la marca superior del círculo. Coloca el lápiz en la marca superior. Ahora dibuja una línea hasta la segunda marca por la izquierda. Dibuja otra línea, ahora hacia la derecha, saltándose la marca de la parte más baja. Complementa el triángulo con una línea hacia la marca superior. Así completarás el triángulo.
8. Crea un segundo triángulo empezando en la marca en la base del círculo. Coloca el lápiz en la marca inferior. Ahora conéctala con la segunda marca hacia la izquierda. Dibuja una línea recta hacia la derecha, saltándose el punto superior. Completa el segundo triángulo dibujando una línea hasta la marca en la parte inferior.
9. Borra el círculo. Has terminado de dibujar tu estrella de 6 puntos.

Referencias

- Raghu Singh (1995). International Standard ISO/IEC 12207 Software Life Cycle Processes. Agosto 23 de 1996, de ISO/IEC. Consulta: Junio de 2015. Disponible en: <http://www.abelia.com/docs/12207cpt.pdf>
- Carlos Guadalupe (2013). Aseguramiento de la calidad del software (SQA). [Figura 1]. Consulta: Junio de 2015. Disponible en: <https://www.mindmeister.com/es/273953719/aseguramiento-de-la-calidad-delsoftware-sqa>
- Andrea S. (2014). Ingeniería de Software. [Figura 2]. Consulta: Junio de 2015. Disponible en: <http://ing-software-verano2014.blogspot.mx>
- Michael Littman. (2012). Intro to Algorithms: Social Network Analysis. Consulta Junio de 2015, de Udacity. Disponible en: <https://www.udacity.com/course/viewer#/c-cs215/l-48747095/m-48691609>

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	52/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 04: Diagramas de flujo



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes M.C. Laura Sandoval Montaño	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	53/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 04: Diagramas de flujo

Objetivo:

El alumno elaborará diagramas de flujo que representen soluciones algorítmicas vistas como una serie de acciones que comprendan un proceso.

Actividades:

- Elaborar un diagrama de flujo que represente la solución algorítmica de un problema, en el cual requiera el uso de la estructura de control condicional.
- Elaborar la representación gráfica de la solución de un problema, a través de un diagrama de flujo, en el cual requiera el uso de la estructura de control iterativa.

Introducción

Un diagrama de flujo es la representación gráfica de un proceso, es decir, muestra gráficamente el flujo de acciones a seguir para cumplir con una tarea específica.

Dentro de las ciencias de la computación, un diagrama de flujo es la representación gráfica de un algoritmo. La correcta construcción de estos diagramas es fundamental para la etapa de codificación, ya que, a partir del diagrama de flujo es posible codificar un programa en algún lenguaje de programación.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 54/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Formas de los diagramas de flujo

Los diagramas de flujo poseen símbolos que permiten estructurar la solución de un problema de manera gráfica. A continuación, se muestran los elementos que conforman este lenguaje gráfico.

1. Todo diagrama de flujo debe tener un inicio y un fin (Figura 1).

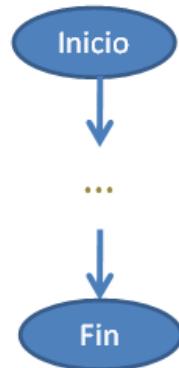


Figura 1. Representación gráfica de Inicio y Fin

2. Las líneas utilizadas para indicar la dirección del flujo del diagrama deben ser rectas, verticales u horizontales, exclusivamente (Figura 2).

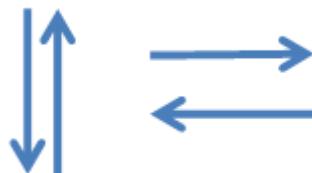


Figura 2. Representación gráfica de dirección de flujo

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 55/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

3. Todas las líneas utilizadas para indicar la dirección del flujo del diagrama deben estar conectadas a un símbolo (Figura 3).

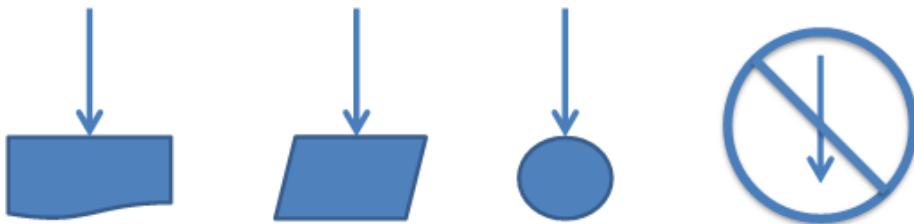


Figura 3. Conexión correcta de las líneas de dirección de flujo

4. El diagrama debe ser construido de arriba hacia abajo (top-down) y de izquierda a derecha (left to right).
5. La notación utilizada en el diagrama de flujo debe ser independiente del lenguaje de programación en el que se va a codificar la solución.
6. Se recomienda poner comentarios que expresen o ayuden a entender un bloque de símbolos.
7. Si la extensión de un diagrama de flujo ocupa más de una página, es necesario utilizar y numerar los símbolos adecuados.
8. A cada símbolo solo le puede llegar una línea de dirección de flujo (Figura 4).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 56/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

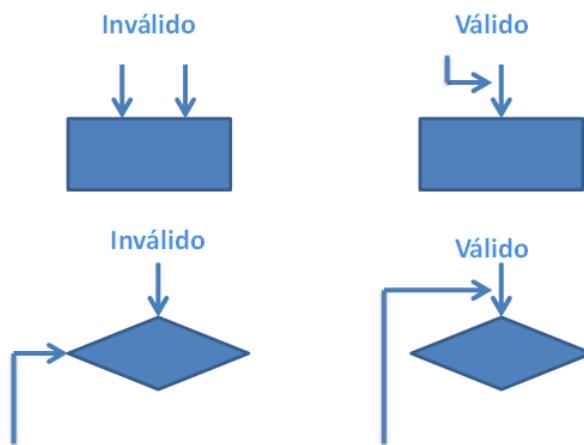


Figura 4. Uso de líneas de dirección

9. Notación de camello. Para nombrar variables y nombres de funciones se debe hacer uso de la notación de camello.

Los diagramas de flujo poseen símbolos que permiten estructurar la solución de un problema de manera gráfica. Por tanto, es fundamental conocer los elementos que conforman este lenguaje gráfico.



Represents el inicio o el fin del diagrama de flujo.

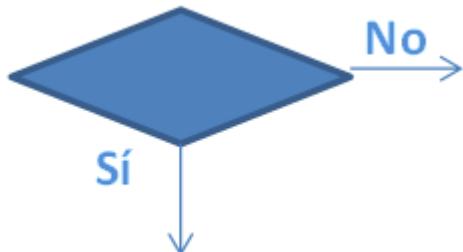
Datos de entrada. Expresa lectura de datos.



	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	57/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión 22 / enero / 2025	
Area/Departamento: Laboratorio de computación salas A y B			
La impresión de este documento es una copia no controlada			



Proceso. En su interior se expresan asignaciones u operaciones.



Decisión. Valida una condición y toma uno u otro camino.



Escritura. Escribe el o los resultado(s).



Dirección de flujo del diagrama.



Conexión dentro de la misma página.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión	MADO-17 05 58/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

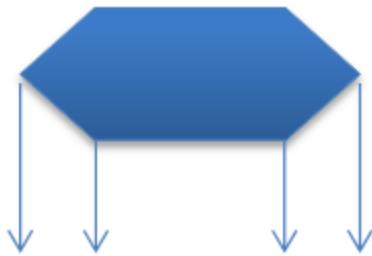
Conexión entre diferentes páginas.



Módulo de un problema. Llamada a otros módulos o funciones.



Decisión múltiple. Almacena un selector que determina la rama por la que sigue el flujo.



La Figura 5 muestra un ejemplo clásico de diagrama de flujo computacional. Se puede observar el uso de los diferentes símbolos para indicar las acciones y el flujo a seguir para la solución de problemas.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 59/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

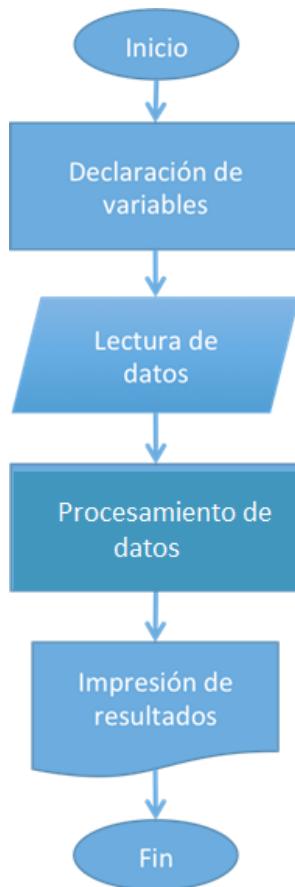


Figura 5. Ejemplo clásico de un diagrama de flujo computacional

Estructuras de control de flujo

Las estructuras de control de flujo permiten tanto la ejecución condicional como la repetición de un conjunto de instrucciones.

Existen 3 estructuras de control: secuencial, condicional y repetitivas o iterativas.

Estructura de control secuencial

Las estructuras de control secuenciales son las sentencias o declaraciones que se realizan una a continuación de otra en el orden en el que están escritas (Figura 6).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 60/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

```
x: REAL
x ← 5.8
x ← x*2
```

Figura 6. Ejemplos de estructuras de control secuencial.

Estructuras de control condicionales (o selectivas)

Las estructuras de control condicionales permiten evaluar una expresión lógica (condición que puede ser verdadera o falsa) y, dependiendo del resultado, se realiza uno u otro flujo de instrucciones. Estas estructuras son mutuamente excluyentes (o se realiza una acción o se realiza la otra).

La Figura 7 muestra la estructura de control condicional más simple. Se denomina estructura condicional SI (IF por su término en inglés).

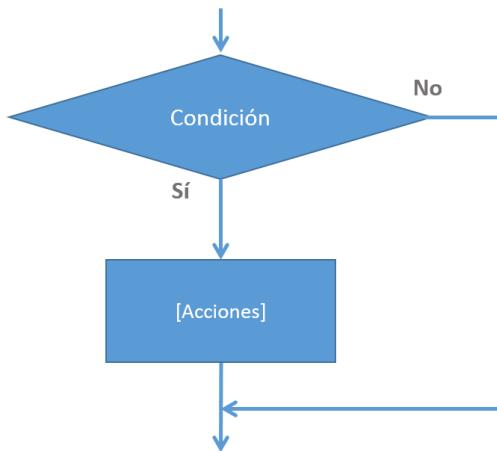


Figura 7. Estructura de control condicional simple.

Se evalúa la expresión lógica y si se cumple (si la condición es verdadera) se realizan las instrucciones del bloque [Acciones]. Si no se cumple la condición, se continúa con el flujo del diagrama descartando las [Acciones].

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 61/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Ejemplo

La Figura 8 muestra el diagrama de flujo que usa la estructura de control condicional simple donde, dados dos números enteros asignados a dos variables, revisa si el valor asignado a la variable *a* es el mayor.

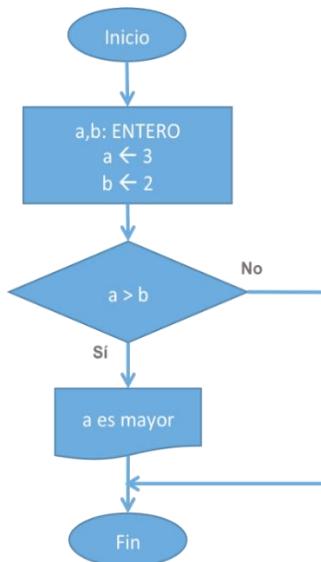


Figura 8. Diagrama de flujo que indica si el valor de la variable *a* es el mayor

Prueba de escritorio:

Instrucción	a	b	salida
a←3	3		
b←2		2	
a > b			
			a es mayor

Además de la estructura de control condicional simple SI, existe la estructura condicional completa nombrada SI-DE LO CONTRARIO (IF-ELSE, por sus términos en inglés). Véase la Figura 9,

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 62/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

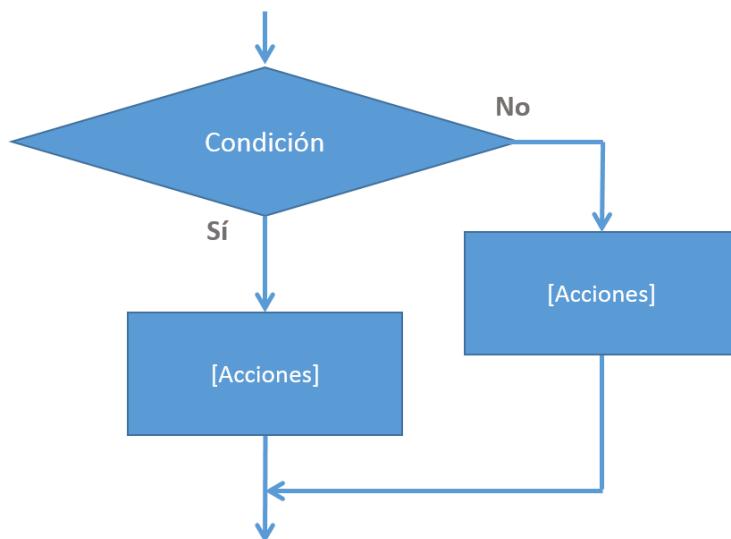


Figura 9. Estructura de control condicional SI-DE LO CONTRARIO

En esta estructura se evalúa la expresión lógica y si se cumple (si la condición es verdadera) se realizan las instrucciones del bloque Sí. Si no se cumple la condición se realizan las instrucciones del bloque No. Al final de la estructura condicional, el flujo continúa con las estructuras que le sigan.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página: 63/187 Sección ISO: 8.3 Fecha de emisión: 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Ejemplo

La Figura 10 muestra el diagrama de flujo que usa la estructura de control condicional SI-DE LO CONTRARIO donde, dados dos números enteros asignados a dos variables, muestra la variable que almacena el valor mayor.

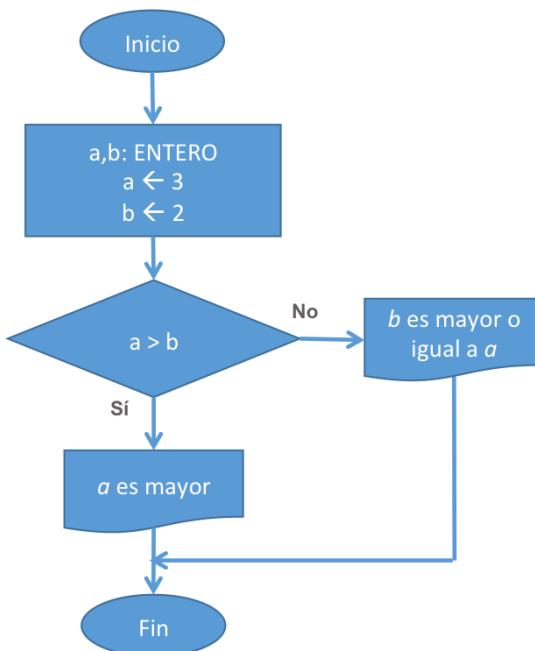


Figura 10. Diagrama de flujo que usa la estructura de control condicional SI-DE LO CONTRARIO

Prueba de escritorio:

Instrucción	a	b	salida
a←3	3		
b←2		2	
a > b			
			a es mayor

Otra estructura de control condicional es SELECCIONAR-CASO, la cual valida el valor de la variable que está en el hexágono y comprueba si es igual al valor que está definido en cada caso (ramas o líneas que emanen del hexágono). Si la variable no tiene el valor

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 64/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

de algún caso entonces fluye por la última rama etiquetada con *, la cual es la única a la que no se le especifica un valor (Figura 11).

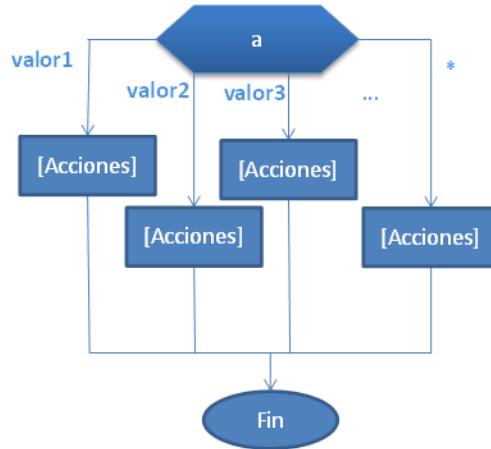


Figura 11. Estructura de control condicional SELECCIONAR-CASO

Ejemplo

La Figura 12 muestra el diagrama de flujo que usa la estructura de control condicional SELECCIONAR-CASO donde, de acuerdo con el valor de la variable a, escribe un mensaje.



Manual de prácticas del Laboratorio de Fundamentos de programación

Código:	MADO-17
Versión:	05
Página	65/187
Sección ISO	8.3
Fecha de emisión	22 / enero / 2025

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

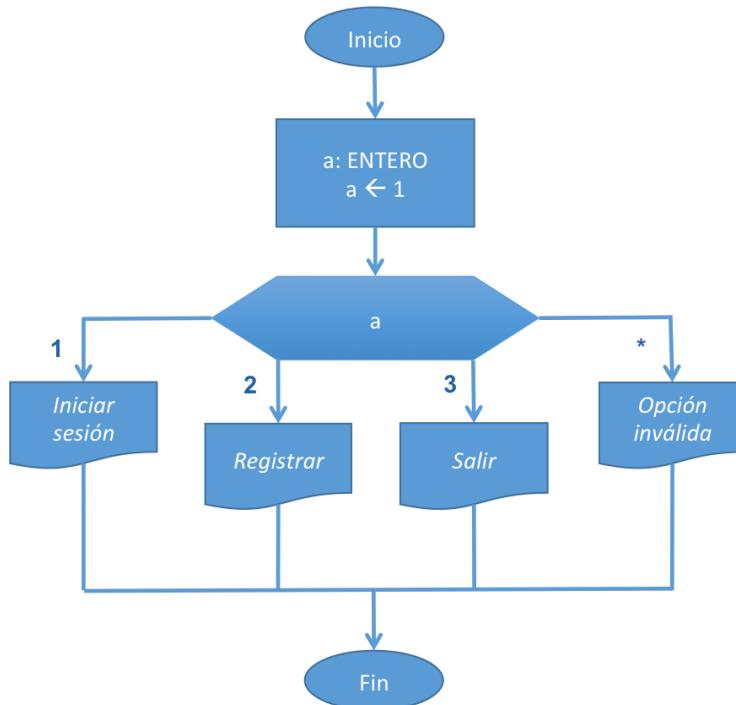


Figura 12. Diagrama de flujo que usa la estructura de control condicional SELECCIONAR-CASO

Prueba de escritorio:

Instrucción	a	salida
$a \leftarrow 1$	1	
		Iniciar sesión

Estructuras de control iterativas o repetitivas

Las estructuras de control de flujo **iterativas o repetitivas** (también llamadas cíclicas) permiten realizar una serie de instrucciones mientras se cumpla la expresión lógica. Existen dos tipos de expresiones cíclicas MIENTRAS y HACER- MIENTRAS.

La estructura MIENTRAS primero valida la condición y si ésta es verdadera (Sí) procede a realizar el bloque de instrucciones de la estructura [Acciones] y regresa a validar la condición, esto lo realiza mientras la condición sea verdadera (Sí); cuando la condición sea Falsa (No se cumpla) se rompe el ciclo y el flujo continúa con las estructuras que le sigan. Véase la Figura 13.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 66/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

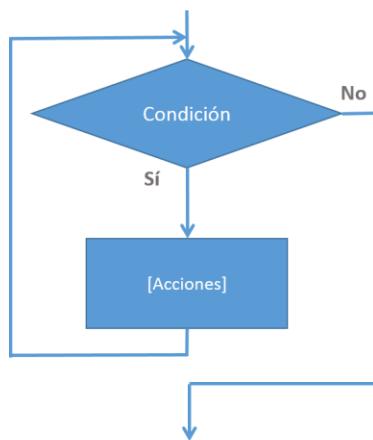


Figura 13. Estructura de control iterativa MIENTRAS

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 67/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Ejemplo

La Figura 14 muestra el diagrama de flujo que usa la estructura de control iterativa MIENTRAS donde se escribe el valor de la variable `valorInicial` mientras se cumpla que el valor de `valorInicial` sea menor al valor de `valorFinal`

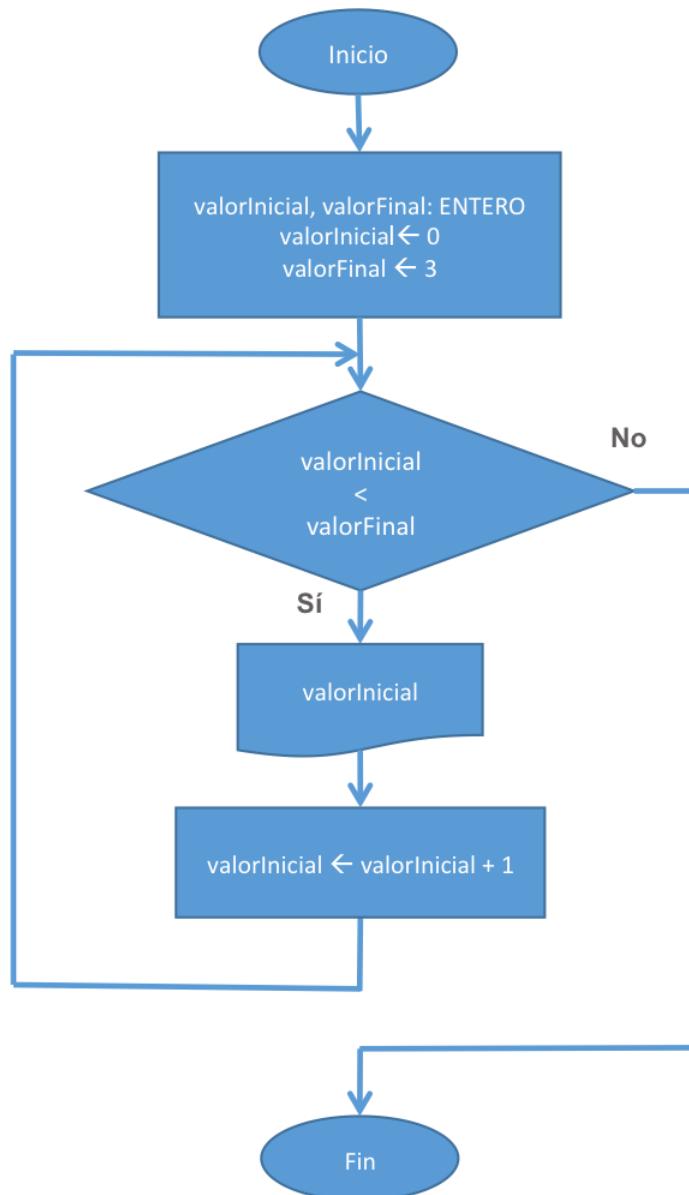


Figura 14. Diagrama de flujo que usa la estructura de control iterativa MIENTRAS

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	68/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Prueba de escritorio:

Instrucción	valorInicial	valorFinal	salida
valorInicial ← 0	0		
valorFinal ← 3		3	
valorInicial < valorFinal			
			0
valorInicial ← valorInicial +1	1		
valorInicial < valorFinal			
			1
valorInicial ← valorInicial +1	2		
valorInicial < valorFinal			
			2
valorInicial ← valorInicial +1	3		
valorInicial < valorFinal			

La otra estructura de control iterativa es HACER-MIENTRAS, la cual primero realiza las instrucciones descritas en la estructura [Acciones] y después valida la expresión lógica; si la expresión lógica es verdadera, realiza nuevamente las [Acciones] y esto se repite mientras la expresión lógica se cumpla. Cuando ésta sea falsa, se rompe el ciclo y el flujo continúa con las estructuras que le sigan. Véase la Figura 15.

Esta estructura asegura que, por lo menos, se realiza una vez el bloque de la estructura, ya que primero las realiza y después pregunta por la condición.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 69/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

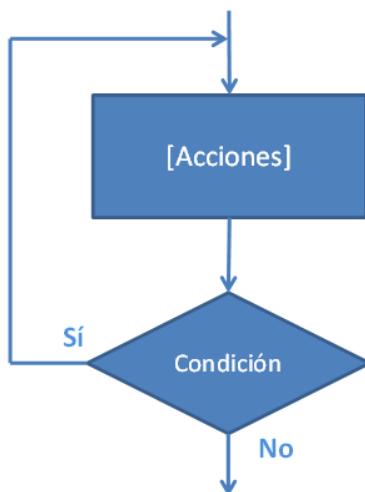


Figura 15. Estructura de control iterativa HACER-MIENTRAS

Ejemplo

La Figura 16 muestra el diagrama de flujo que usa la estructura de control iterativa HACER-MIENTRAS el cual escribe el valor de la variable `enteroValorInicial` y luego realiza el incremento de esta variable para posteriormente evaluar la condicional, esto se repite mientras la condición sea verdadera.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 70/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

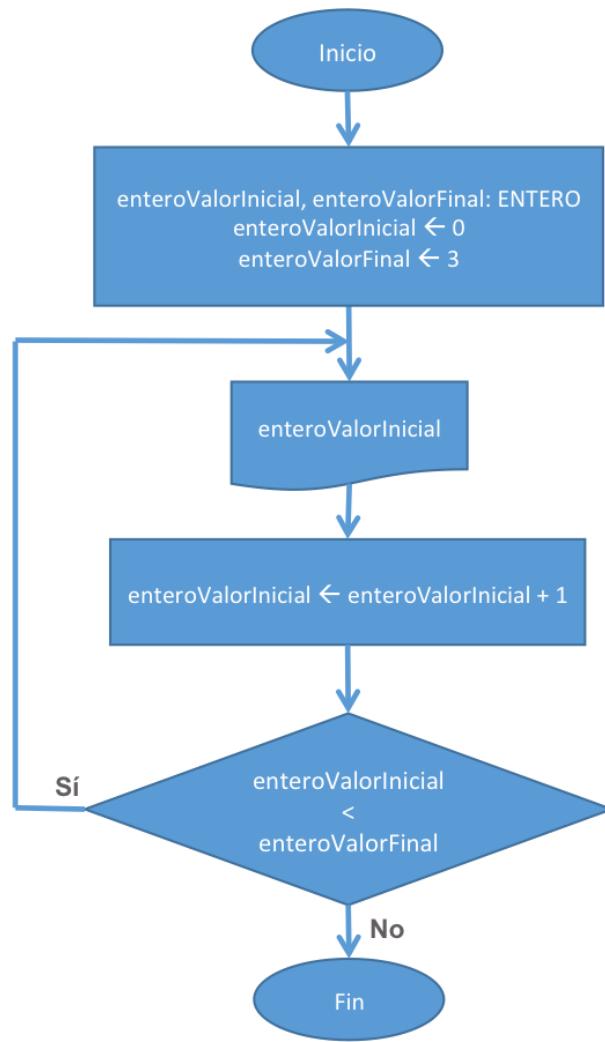


Figura 16. Diagrama de flujo que usa la estructura de control iterativa HACER-MIENTRAS

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	71/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Prueba de escritorio:

Instrucción	enteroValorInicial	enteroValorFinal	salida
enteroValorInicial ← 0	0		
enteroValorFinal ← 3		3	
			0
enteroValorInicial ← enteroValorInicial +1	1		
enteroValorInicial < enteroValorFinal			
			1
enteroValorInicial ← enteroValorInicial +1	2		
enteroValorInicial < enteroValorFinal			
			2
enteroValorInicial ← enteroValorInicial +1	3		
enteroValorInicial < enteroValorFinal			

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 72/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Bibliografía

- Metodología de la programación. Osvaldo Cairó, tercera edición, México D.F., Alfaomega 2005.



- Metodología de la programación a través de pseudocódigo. Miguel Ángel Rodríguez Almeida, primera edición, McGraw Hill



	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	73/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 05: Pseudocódigo



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes M.C. Laura Sandoval Montaño	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	74/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 05: Pseudocódigo

Objetivo:

El alumno elaborará pseudocódigos que representen soluciones algorítmicas empleando la sintaxis y semántica adecuadas.

Actividades:

- Elaborar un pseudocódigo que represente la solución algorítmica de un problema en el cual requiera el uso de la estructura de control de flujo condicional.
- A través de un pseudocódigo, representar la solución algorítmica de un problema en el cual requiera el uso de la estructura de control iterativa.

Introducción

Una vez que un problema dado ha sido analizado (se obtiene el conjunto de datos de entrada y el conjunto de datos de salida esperado) y se ha diseñado un algoritmo que lo resuelva de manera eficiente (procesamiento de datos), se debe proceder a la etapa de codificación del algoritmo.

Para que la solución de un problema (algoritmo) pueda ser codificada, se debe generar una representación de éste. Una representación algorítmica elemental es el pseudocódigo.

Un pseudocódigo es la representación escrita de un algoritmo, es decir, muestra en forma de texto los pasos a seguir para solucionar un problema. El pseudocódigo posee una sintaxis propia para poder realizar la representación del algoritmo (solución de un problema).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	75/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Sintaxis de pseudocódigo

El lenguaje pseudocódigo tiene diversas reglas semánticas y sintácticas. A continuación, se describen las más importantes:

1. Alcance del programa: Todo pseudocódigo está limitado por las etiquetas de INICIO y FIN. Dentro de estas etiquetas se deben escribir todas las instrucciones del algoritmo.
2. Palabras reservadas con mayúsculas: Todas las palabras propias del pseudocódigo deben de ser escritas en mayúsculas.
3. Sangría o tabulación: El pseudocódigo debe tener diversas alineaciones para que el código sea más fácil de entender y depurar.
4. Lectura / escritura: Para indicar lectura de datos se utiliza la etiqueta LEER. Para indicar escritura de datos se utiliza la etiqueta ESCRIBIR.

Ejemplo

```
ESCRIBIR "Ingresar la altura del polígono"
LEER altura
```

5. Declaración de variables: la declaración de variables la definen un identificador (nombre), seguido de dos puntos, seguido del tipo de dato, es decir:

```
<nombreVariable>:<tipoDeDato>
```

Los tipos de datos que se pueden utilizar son:

- ENTERO -> valor entero positivo y/o negativo
- REAL -> valor con punto flotante y signo
- BOOLEANO -> valor de dos estados: verdadero o falso
- CARACTER -> valor tipo carácter
- CADENA -> cadena de caracteres

Ejemplo

```
contador: ENTERO
producto: REAL
continuar: BOOLEANO
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	76/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Es posible declarar más de una variable de un mismo tipo de dato utilizando arreglos, indicando la cantidad de variables que se requieren, su sintaxis es la siguiente:

```
<nombreVariable>[cantidad]:<tipoDeDato>
```

Ejemplo

```
contador[5]: ENTERO      → 5 variables de tipo entero
division[3]: REAL        → 3 variables de tipo real
bandera[6]: BOOLEANO    → 6 variables de tipo booleano
```

Existe un tipo de dato compuesto, es decir, que puede contener uno o más tipos de datos simples diferentes. Este tipo de dato se conoce como registro o estructura y su sintaxis es la siguiente:

```
<nombreRegistro>:REG
  <nombreVariable_1>:<tipoDeDato>
  ...
  <nombreVariable_N>:<tipoDeDato>
FIN REG
```

Para crear una variable tipo registro se debe indicar el nombre del registro y el nombre de la variable. Para acceder a los datos del registro se hace uso del operador punto (.).

Ejemplo

```
domicilio:REG
  calle: CADENA
  número: ENTERO
  ciudad: CADENA
FIN REG
```

```
usuario:REG domicilio → variable llamada usuario de tipo registro
usuario.calle := "Av. Imán"
usuario.número := 3000
usuario.ciudad := "México"
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	77/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Es posible crear variables constantes con la palabra reservada CONST, la cual indica que un identificador no cambia su valor durante todo el pseudocódigo. Las constantes (por convención) se escriben con mayúsculas y se deben inicializar al momento de declararse.

Ejemplo

```
NUM_MAX := 1000: REAL, CONST
```

6. Operadores aritméticos: Se tiene la posibilidad de utilizar operadores aritméticos y lógicos:

Operadores aritméticos: suma (+), resta (-), multiplicación (*), división (/), división entera (div) **esto ya es en programación hay que omitirla**, módulo (mod), exponentiación (^), asignación (:=).

Operadores lógicos: igualdad (=), Y-lógica o AND (&), O-lógica u OR (|), negación o NOT (!), relaciones de orden (<, >, <=, >=) y diferente (<>).

La tabla de verdad de los operadores lógicos Y (AND), O (OR) y Negación (NOT) se describe en la Tabla 1.

Tabla 1: Tabla de verdad de operadores lógicos

A	B	A & B	A B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

NOTA: A y B son dos condiciones, el valor 0 indica falso y el valor 1 indica verdadero.

7. Notación de camello. Para nombrar variables y nombres de funciones se debe hacer uso de la notación de camello.

En la notación de camello (llamada así porque parecen las jorobas de un camello) los nombres de cada palabra empiezan con mayúscula y el resto se escribe con minúsculas. Existen dos tipos de notaciones de camello: lower camel case que en la

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 78/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

cual la primera letra de la variable inicia con minúscula y upper camel case en la cual todas las palabras inician con mayúscula. No se usan puntos ni guiones para separar las palabras (a excepción de las constantes que utilizan guiones bajos). Además, para saber el tipo de variable se recomienda utilizar un prefijo.

Ejemplo

```
realAreaDelTriangulo: REAL      → lower camel case
EnterRadioCirculo: REAL         → upper camel case
```

```
calcularArea()
obtenerPerimetro()
```

Estructuras de control de flujo

Las estructuras de control de flujo permiten la ejecución condicional y la repetición de un conjunto de instrucciones.

Existen 3 estructuras de control: secuencial, condicional y repetitivas o iterativas.

Estructura de control secuencial

Las estructuras de control secuenciales son las sentencias o declaraciones que se realizan una a continuación de otra en el orden en el que están escritas.

Ejemplo

```
INICIO
  x : REAL
  x := 5.8
  x := x * 2
FIN
```

Estructuras de control condicionales (o selectivas)

Las estructuras de control condicionales permiten evaluar una expresión lógica (condición que puede ser verdadera o falsa) y, dependiendo del resultado, se realiza uno u otro flujo de instrucciones. Estas estructuras son mutuamente excluyentes (o se realiza una acción o se realiza la otra)

La estructura de control de flujo más simple es la estructura condicional SI, su sintaxis es la siguiente:

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	79/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

SI condición ENTONCES

[Acciones]

FIN SI

Se evalúa la expresión lógica y si se cumple (si la condición es verdadera) se realizan las instrucciones del bloque [Acciones]. Si no se cumple la condición, se continúa con el flujo del pseudocódigo descartando las [Acciones].

Ejemplo

INICIO

a,b: ENTERO

a := 3

b := 2

SI a > b ENTONCES

ESCRIBIR "a es mayor"

FIN SI

FIN

Prueba de escritorio:

Instrucción	a	b	salida	
a := 3	3			
b := 2		2		
a > b				
			a es mayor	

Además de la estructura de control condicional simple SI, existe la estructura condicional completa nombrada SI-DE LO CONTRARIO.

SI expresión Lógica

ENTONCES

[Acciones ENTONCES]

DE LO CONTRARIO

[Acciones DE LO CONTRARIO]

FIN DEL SI

Se evalúa la *expresión lógica* y si se cumple (si es verdadera) se realizan las instrucciones del bloque SI [Acciones ENTONCES]. Si no se cumple la *expresión lógica* (si es falsa) se realizan las instrucciones del bloque DE LO CONTRARIO [Acciones DE LO

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 80/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

CONTRARIO]. Al final de la estructura condicional, el flujo continúa con las estructuras que le sigan.

Ejemplo

INICIO

```

a,b:ENTERO
a := 3
b := 5
SI a > b
    ENTONCES
        ESCRIBIR "a es mayor"
    DE LO CONTRARIO
        ESCRIBIR "b es mayor"
    FIN DEL SI
FIN

```

Prueba de escritorio:

Instrucción	a	b	salida
a := 3	3		
b := 2		5	
a > b			
			b es mayor

Otra estructura de control condicional es SELECCIONAR-CASO, la cual valida el valor de la variable que está entre paréntesis y comprueba si es igual al valor que está definido en cada caso. Si la variable no tiene el valor de ningún caso se va a la instrucción por defecto (DEFECTO).

```

SELECCIONAR (variable) EN
    CASO valor1 ->
        [Acciones]
    CASO valor2 ->
        [Acciones]
    CASO valor3 ->
        [Acciones]
    DEFECTO ->
        [Acciones]
FIN SELECCIONAR

```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	81/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Ejemplo

```

INICIO
  a :ENTERO
  a := 1
  SELECCIONAR (a) EN
    CASO 1 ->
      ESCRIBIR "Iniciar sesión."
    CASO 2 ->
      ESCRIBIR "Registrarse."
    CASO 3 ->
      ESCRIBIR "Salir."
    DEFECTO ->
      ESCRIBIR "Opción inválida."
  FIN SELECCIONAR
FIN

```

Prueba de escritorio:

Instrucción	a	salida
a := 1	1	
		Iniciar sesión

Estructuras de control iterativas o repetitivas

Las estructuras de control de flujo **iterativas o repetitivas** (también llamadas cíclicas) permiten realizar una serie de instrucciones mientras se cumpla la expresión lógica. Existen dos tipos de expresiones cíclicas MIENTRAS y HACER- MIENTRAS.

La estructura MIENTRAS (WHILE en inglés) primero valida la condición y si ésta es verdadera procede a realizar el bloque de instrucciones de la estructura [Acciones] y regresa a validar la condición, esto lo realiza mientras la condición sea verdadera; cuando la condición es Falsa (no se cumpla) se rompe el ciclo y continúa el flujo normal del pseudocódigo.

```

MIENTRAS condición ENTONCES
  [Acciones]
FIN MIENTRAS

```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	82/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

El final de la estructura lo determina la etiqueta FIN MIENTRAS.

Ejemplo

```

INICIO
    valorInicial,valorFinal:ENTERO
    valorInicial:=0
    valorFinal:=3
    MIENTRAS valorInicial < valorFinal
        ESCRIBIR valorInicial
        valorInicial := valorInicial + 1
    FIN MIENTRAS
FIN

```

Prueba de escritorio:

Instrucción	valorInicial	valorFinal	salida
valorInicial := 0	0		
valorFinal := 3		3	
valorInicial < valorFinal			
			0
valorInicial := valorInicial +1	1		
valorInicial < valorFinal			
			1
valorInicial := valorInicial +1	2		
valorInicial < valorFinal			
			2
valorInicial := valorInicial +1	3		
valorInicial < valorFinal			

La estructura HACER-MIENTRAS primero realiza las instrucciones descritas en la estructura y después valida la expresión lógica.

```

HACER
    [Acciones]
    MIENTRAS condición

```

Si la condición se cumple vuelve a realizar las instrucciones de la estructura, de lo contrario rompe el ciclo y sigue el flujo del pseudocódigo. Esta estructura asegura que,

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 83/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

por lo menos, se realiza una vez el bloque de la estructura, ya que primero realiza las instrucciones del bloque y después pregunta por la condición.

Ejemplo

```

INICIO
    valorInicial,valorFinal:ENTERO
    valorInicial:=0
    valorFinal:=3
    HACER
        ESCRIBIR valorInicial
        valorInicial := valorInicial + 1
    MIENTRAS valorInicial < valorFinal
FIN

```

Prueba de escritorio:

Instrucción	valorInicial	valorFinal	salida
valorInicial := 0	0		
valorFinal := 3		3	
			0
valorInicial := valorInicial +1	1		
valorInicial < valorFinal			
			1
valorInicial := valorInicial +1	2		
valorInicial < valorFinal			
			2
valorInicial := valorInicial +1	3		
valorInicial < valorFinal			

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 84/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Bibliografía

- Metodología de la programación. Osvaldo Cairó, tercera edición, México D.F., Alfaomega 2005.



- Metodología de la programación a través de pseudocódigo. Miguel Ángel Rodríguez Almeida, primera edición, McGraw Hill



	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	85/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 06: Entorno y fundamentos del lenguaje C



Elaborado por	Actualizado por:	Revisado por:
Ing. Jorge A. Solano Gálvez M.C. Laura Sandoval Montaño	Ing. Julio De León Razo M.T. Hugo Zúñiga Barragán	M.C. Cintia Quezada Reyes M.C. Laura Sandoval Montaño Ing. Jorge A. Solano Gálvez

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	86/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 06: Entorno y fundamentos del lenguaje C

Objetivo:

El alumnado elaborará programas en lenguaje C utilizando las instrucciones de control de tipo secuencia, para realizar la declaración de variables de diferentes tipos de datos, así como efectuar llamadas a funciones externas de entrada y salida para asignar y mostrar valores de variables y expresiones.

Actividades:

- Crear un archivo de texto (utilizando algún editor) y escribir un programa en lenguaje C que contenga variables de diferentes tipos, asignación de valores (por lectura desde la entrada estándar o asignación directa) y escritura del valor de las variables en la salida estándar
- Compilar un código fuente y ejecutarlo.
- Modificar y actualizar un programa usando un editor.
- Elaborar expresiones relacionales/lógicas en un programa en C y mostrar el resultado de su evaluación.

Introducción

Una vez que un problema dado ha sido analizado (se identifican los datos de entrada y la salida deseada), que se ha diseñado un algoritmo que lo resuelva de manera eficiente (procesamiento de datos), y que se ha representado el algoritmo de manera gráfica o escrita (diagrama de flujo o pseudocódigo) se puede proceder a la etapa de codificación.

La codificación se puede realizar en cualquier lenguaje de programación estructurada. En este curso se aprenderá el uso del lenguaje de programación C.

Dentro del ciclo de vida del software, la implementación de un algoritmo se encuentra en la etapa de *codificación* del problema. (Figura 1)

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 87/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

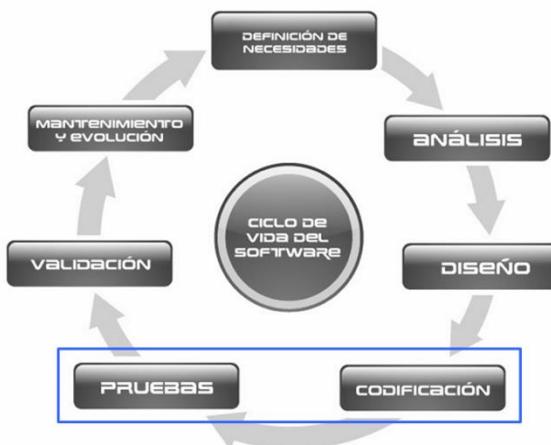


Figura 1: Ciclo de vida del software.

Entorno de C

Un lenguaje de programación permite expresar una serie de instrucciones que podrán ser realizadas por una computadora. Uno de los lenguajes de programación mayormente difundidos es el lenguaje C.

Una característica importante del lenguaje C es que es muy poderoso ya que combina las características de un lenguaje de alto nivel (facilidad de programación), con uno de bajo nivel (manejo más preciso de una máquina); por lo que se han creado variantes que permiten programar miles de dispositivos electrónicos en el mundo con sus respectivos compiladores.

Un programa en C se elabora describiendo cada una de las instrucciones de acuerdo con las reglas definidas en este lenguaje en un archivo de texto para después ser procesadas en un compilador. Un compilador es un programa que toma como entrada un archivo de texto y tiene como salida un programa ejecutable, éste tiene instrucciones que pueden ser procesadas por el hardware de la computadora en conjunto con el sistema operativo que corre sobre ella. Se tiene como ventaja que un programa escrito en lenguaje C, siguiendo siempre su estándar, puede ejecutarse en cualquier máquina siempre y cuando exista un compilador de C. A esto también se le conoce como multiplataforma.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	88/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Editores

Un programa en C debe ser escrito en un editor de texto para después generar un programa ejecutable en la computadora por medio de un compilador. Tanto el editor de texto como el compilador van de la mano con el sistema operativo y si posee o no interfaz gráfica, por lo que son factores que se deben tomar en cuenta a la hora de elegir el entorno para desarrollar programas en C.

Es importante señalar que no es lo mismo un editor de texto que un procesador de texto. El primero edita un texto plano que puede tener muchas utilidades como guardar una configuración, tener escrito un programa, etcétera, y será interpretado hasta que se haga una lectura de éste. Un procesador de texto permite dar formato al texto, a la hoja donde está escrito, incrustar imágenes, entre otros, su salida puede ser un archivo de texto plano que contiene etiquetas que señalan el formato que se le dio al texto o algo un poco más complejo. A continuación, se presentan algunos de los editores más comunes.

Editor Visual Interface de GNU/Linux (*vi*)

El editor *vi* (visual interface) es el editor más común en cualquier distribución de sistemas operativos con núcleo basado en UNIX. Está disponible en línea de comandos y si el sistema operativo tiene entorno gráfico se puede acceder a él desde la *terminal*. *vi* es un editor que puede resultar difícil de usar en un inicio. Aunque existen editores más intuitivos en su uso; en muchas ocasiones *vi* es el único disponible.

Para iniciar *vi*, debe teclearse desde la línea de comandos:

```
vi nombre_archivo[.ext]
```

Donde *nombre_archivo* es el nombre del archivo a editar o el nombre de un archivo nuevo que se creará con *vi*, y *[.ext]* se refiere a la *extensión* que indica que el texto es una programa escrito en algún lenguaje o es texto plano, por ejemplo. Es válido incluir la ruta donde se localiza o localizará el archivo. Existen más métodos de apertura para usuarios más avanzados.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 89/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Modo comando (Figura 2)

Es el modo por defecto de *vi* cuando se abre. Las teclas presionadas ejecutan diversas acciones predeterminadas y no se puede editar el texto libremente. Los comandos son sensativos a las mayúsculas y a las minúsculas. Algunos ejemplos son:

- ↑ o *k* mueve el cursor hacia arriba.
- ↓ o *j* mueve el cursor hacia abajo.
- ← o *h* mueve el cursor hacia la izquierda.
- → o *l* mueve el cursor hacia la derecha.
- 1G lleva el cursor al comienzo de la primera línea.
- G lleva el cursor al comienzo de la última línea.
- x borra el carácter marcado por el cursor.
- dd borra o corta la línea donde está el cursor.
- nnd donde *n* es la cantidad de líneas que se borrarán o cortarán después del cursor.
- D borra o corta desde la posición de cursor hasta el final de la línea.
- dw borra o corta desde la posición del cursor hasta el final de una palabra.
- yy copia la línea donde está el cursor.
- p pega un contenido copiado o borrado.
- u deshace el último cambio.

```
Ejemplo de editor de texto vi
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
I hola.txt [Modified] 1/1 100%
```

Figura 2: *vi* en modo comando.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	90/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Modo de última línea

Se puede acceder a él utilizando el modo comando pero los comandos no tendrán efecto hasta que se presiona la tecla Enter además de que se visualizará el comando en la última línea del editor. Es posible cancelar el comando con la tecla Esc. Los comandos de última línea se caracterizan porque inician con /, ? o :. Algunos ejemplos son:

- **/texto** donde la cadena **texto** será buscada hacia delante de donde se encuentra el cursor.
- **?texto** donde la cadena **texto** será buscada hacia atrás de donde se encuentra el cursor.
- **:q** para salir de *vi* sin haber editado el texto desde la última vez que se guardó.
- **:q!** para salir de *vi* sin guardar los cambios.
- **:w** para guardar los cambios sin salir de *vi*.
- **:w archivo** para realizar la orden “guardar como”, siendo **archivo** el nombre donde se guardará el documento.
- **:wq** guarda los cambios y sale de *vi*.

Modo Edición

Para poder editar un texto dentro del editor, solamente tendremos que oprimir la tecla **<i>** desde el modo comando y podremos insertar el texto en donde se encuentre el cursor y comienza a escribir justo en el carácter, con **<a>** podremos insertar el texto en donde se encuentre el cursor y comenzar a escribir justo después del carácter; si oprimimos la tecla **<l>** podremos insertar texto desde el inicio de la línea donde se encuentre el cursor y con **<A>** podremos insertar texto al final de la línea donde se encuentre el cursor.

GNU nano

Es un editor de texto disponible para sistemas operativos basados en UNIX en línea de comandos. Se puede acceder en un entorno gráfico desde la aplicación de terminal.

Este editor es mucho más intuitivo que *vi*, aunque menos potente. No es necesario saber cómo se utiliza ya que proporciona una interfaz que describe los comandos básicos. (Figura 3)

nano es un editor clon de otro editor llamado pico.

Para iniciar nano, debe teclearse desde la línea de comandos:

```
nano nombre_archivo[.ext]
```

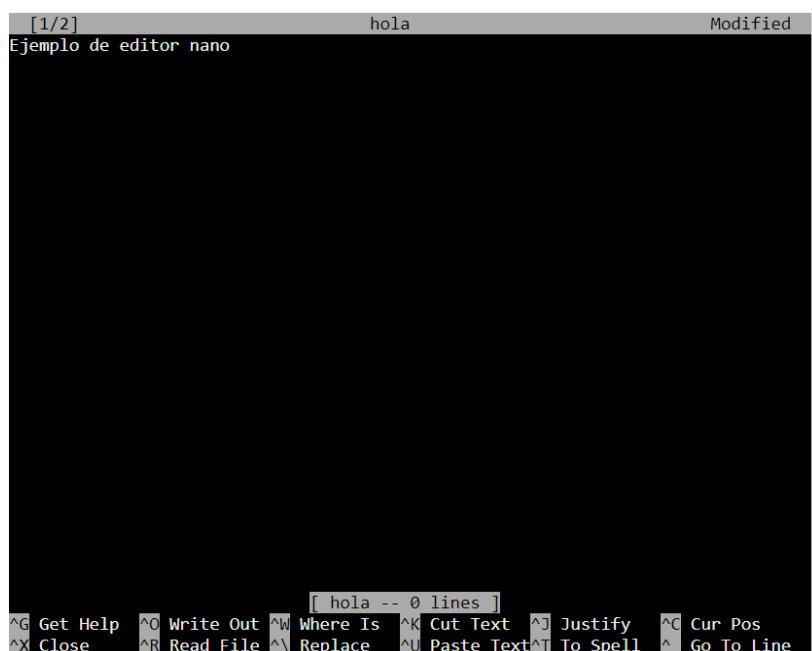
	Manual de prácticas del Laboratorio de Fundamentos de programación	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Código:</td><td>MADO-17</td></tr> <tr> <td>Versión:</td><td>05</td></tr> <tr> <td>Página</td><td>91/187</td></tr> <tr style="background-color: #cccccc;"> <td>Sección ISO</td><td>8.3</td></tr> <tr> <td>Fecha de emisión</td><td>22 / enero / 2025</td></tr> </table>	Código:	MADO-17	Versión:	05	Página	91/187	Sección ISO	8.3	Fecha de emisión	22 / enero / 2025
Código:	MADO-17											
Versión:	05											
Página	91/187											
Sección ISO	8.3											
Fecha de emisión	22 / enero / 2025											
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B										
La impresión de este documento es una copia no controlada												

Donde “*nombre_archivo*” es el nombre del archivo a editar o el nombre de un archivo nuevo.

Una vez en el editor, en la parte inferior se pueden observar los comandos básicos. Si se presiona la tecla **F1** es posible visualizar la ayuda con la lista de todos comandos que existen.

Los atajos de teclado pueden corresponder a:

- ^ que es la tecla **Ctrl**.
- M- que es la tecla **Esc** o bien **Alt**.



Ejemplo de editor nano

[1/2] hola Modified

[hola -- 0 lines]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
 ^X Close ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line

Figura 3: El editor de texto nano.

Otros

Existen otros editores actualmente como: Visual Studio Code, Atom, Sublime Text, Notepad++, entre otros.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	92/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Compiladores

Una vez codificado un programa en C en algún editor de texto, éste debe ser leído por un programa que produzca un archivo ejecutable. A este programa se le conoce como compilador. Para el caso del lenguaje C el compilador traduce el código fuente del programa a código ejecutable.

Un programa en C tampoco puede ser escrito de manera arbitraria debe respetar una serie de reglas para que el compilador pueda entenderlas y realizar su función. Un estándar muy común es **ANSI C** y existen diferentes extensiones como ISOC99 y GNU C que representan mejoras para el estándar original. Realizar un programa en dicho estándar garantiza que puede ejecutarse en cualquier máquina siempre y cuando exista un compilador hecho para ella. A veces, el programador no sigue un estándar o lo desconoce, usando características no estándar que, a la hora de usar el mismo programa para otra máquina no funciona, teniendo que realizar adaptaciones que se reflejan en costos. Por ejemplo, es muy común empezar a desarrollar programas en C en plataforma Windows en procesadores x86 usando características propias. Al trasladar, por alguna necesidad, dicho programa a plataforma GNU/Linux con procesador ARM, el programa no funcionará porque no se siguió el estándar que garantiza universalidad para el lenguaje C.

Es muy común cometer algún error al elaborar un programa en C como son faltas a la sintaxis que indica el estándar, usar elementos que no se habían declarado, utilizar funciones de una biblioteca sin haberla especificado, entre muchos otros que se irán conociendo conforme se vaya teniendo más experiencia en la programación. La mayoría de estos errores provocan que el compilador no pueda generar el programa ejecutable y muestre en la línea de comandos de qué error se trata y en qué línea pudo haberse producido.

Es importante señalar que un solo error puede desencadenar muchos otros y al corregirlo los demás dejarán de ser errores. También, en muchas ocasiones el error no se encuentra en la línea que el compilador señala sino en líneas anteriores, lo que señala es la línea en la que el compilador ya no encontró estructura el programa codificado ya que éste no tiene criterio propio sino se trata de un programa de computadora.

Cuando el compilador señala un error no cabe más que invocar algún editor de texto, revisar cuidadosamente el programa y corregir. Se debe verificar la coherencia total del programa para evitar tener que repetir este paso de manera continua.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 93/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

En ocasiones, el compilador genera advertencias (warnings) durante el proceso de compilación. Si solo se presentan advertencias, el compilador aún puede generar el archivo ejecutable, pero es posible que surjan problemas al intentar ejecutarlo. Por ello, es recomendable revisar detenidamente las advertencias emitidas por el compilador y realizar las correcciones necesarias.

A continuación, se describe el uso de un compilador de C comúnmente utilizado en plataformas basadas en el sistema operativo Unix, así como los Entornos de Desarrollo (IDE) más populares para la edición y compilación de programas en C.

gcc (GNU Compiler Collection)

Es un conjunto de compiladores de uso libre para sistemas operativos basados en UNIX. Entre sus compiladores existe el que sirve para programas escritos en C. Se encuentra por defecto en diversas distribuciones de GNU/Linux. El compilador trabaja en línea de comandos.

Existe también una versión modificada que puede ejecutar y crear programas para plataformas Windows en un paquete llamado MinGW (Minimalist GNU for Windows).

Al compilar un programa en C el compilador genera diversos archivos intermedios que corresponden a las distintas fases que realiza. Éstas no son de interés por el momento y son eliminadas una vez obtenido el archivo ejecutable. gcc tiene diferentes opciones de ejecución para usuarios más avanzados.

Suponiendo que se tiene un programa escrito en C y se le llamó *calculadora.c*, la manera de compilarlo es localizándose mediante la línea de comandos en la ruta donde el archivo se encuentra y ejecutando el comando:

`gcc calculadora.c`

Esto creará un archivo *a.out* (en Windows *a.exe*) que es el programa ejecutable resultado de la compilación.

Si se desea que la salida tenga un nombre en particular, debe definirse por medio del parámetro *-o* de *gcc*. Por ejemplo, para que se llame *calculadora.out* (en Windows *calculadora.exe*), se escribe en la línea de comandos:

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	94/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

`gcc calculadora.c -o calculadora.out`

A veces, para realizar un programa más complejo, se necesitan bibliotecas que se instalaron en el equipo previamente y se definió su uso en el programa escrito en C pero al momento de compilar es necesario indicar a GCC que se están usando bibliotecas que no se encuentran en su repertorio de bibliotecas estándar. Para ello es necesario utilizar el parámetro `-l` seguido inmediatamente por el nombre de la biblioteca, sin dejar espacio alguno:

`gcc calculadora.c -o calculadora -lnombre_biblioteca`

Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés)

Los IDE están diseñados para maximizar la productividad del programador proporcionando componentes muy unidos con interfaces de usuario similares. Los IDE presentan un único programa en el que se llevan a cabo las diversas etapas de desarrollo de software. Generalmente, este programa suele ofrecer muchas características para la creación, modificación, compilación, implementación y depuración de software. Esto contrasta con el desarrollo de software utilizando herramientas no relacionadas, como `vi`, GNU Compiler Collection (`gcc`) o `make`.

Dev-C++: Este emplea el compilador MinGW. Se trata de un software libre, sencillo, ligero y eficiente, para la plataforma Windows.

Code Blocks: Este es un software libre, multiplataforma. Code Blocks es una alternativa a Dev-C++ y desarrollada mediante el propio lenguaje C++. Sus capacidades son bastante buenas y es muy popular entre los nuevos programadores. Se puede encontrar separado del compilado o la versión “mingw” que incluye g++ (gcc para C++).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	95/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Xcode: Este es uno de los mejores IDE para programar en Mac con el compilador gcc e Interface Builder.

Ejecución

La ejecución es la etapa que sigue después de haber compilado el programa. Una vez compilado el programa, se puede distribuir para equipos que ejecuten el mismo sistema operativo y tengan la misma plataforma de hardware (tipo de procesador, set de instrucciones y arquitectura en general). Los pasos para realizar la ejecución dependen del sistema operativo y del entorno.

En Windows se puede ejecutar el programa haciendo doble clic sobre el programa ya compilado, pero se recomienda exhaustivamente que se haga desde *símbolo de sistema*.

Considerando que se tiene un programa compilado en un sistema base Unix cuyo nombre es *calculadora.out*, para ejecutar debe teclearse en línea de comandos:

```
./calculadora.out
```

Si el programa realizado necesita tener una entrada de información por medio de argumentos, éstos se colocan así:

```
./calculadora.out argumento1 argumento2
```

En un inicio, cualquier programa escrito en C sólo funcionará en modo línea de comandos, ya que para que tenga una interfaz gráfica se requiere de conocimientos avanzados sobre el lenguaje y del sistema operativo para el que se diseña el programa.

Es difícil realizar un programa con interfaz gráfica universal y que respete ANSI C, ya que el entorno depende del sistema operativo y las herramientas que provee.

Muchos errores no se reflejarán en el compilador porque el programa está correctamente escrito de acuerdo con lo que ANSI C señala, pero lo que se programó

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	96/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

puede ser erróneo y tener resultados distintos a los deseados. Por ello, en la fase de ejecución deben hacerse diversas pruebas para verificar que el programa hace lo que debería.

Aunque el programa aparentemente funciona, deben hacerse pruebas exhaustivas para verificar la integridad del programa. Por ejemplo, si el programa realiza una división, es necesario evaluar que el divisor no sea cero, o bien, que los números que se manejan no rebasen el valor máximo que el equipo soporta, entre muchos otros detalles que pueden presentarse.

Es muy común que se construya un programa conforme a lo que ANSI C determina y se pruebe en un equipo personal que, por lo general, es de altas prestaciones. A la hora de instalar ese programa en un equipo especializado que tiene prestaciones limitadas, el programa puede no funcionar ya que no se optimizó. Por ejemplo, se crea un programa que al ejecutarse ocupa 600 MB en memoria principal en un equipo con 4 GB y se ejecutará sin problemas. Sin embargo, si este programa tiene que ejecutarse en un equipo con 1GB de memoria principal su rendimiento será muy pobre debido a que tanto el sistema operativo como otros procesos están ocupando ya una parte de esta memoria.

Es muy importante tratar de que un programa sea íntegro y optimizado para garantizar su correcto funcionamiento y en ocasiones, el prestigio de su autor.

Lenguaje de programación C

C es un lenguaje de propósito general basado en el paradigma estructurado. El teorema del programa estructurado, demostrado por Böhm-Jacopini, dicta que todo programa puede desarrollarse utilizando únicamente 3 instrucciones de control: Secuencia, Selección e Iteración.

Un programa en C consiste en una o más funciones, de las cuales una de ellas debe llamarse *main()* y es la principal.

Al momento de ejecutar un programa objeto (código binario), se procesarán las instrucciones que estén definidas dentro de la función principal. Dicha función puede contener sentencias, estructuras de control y comentarios. Dentro de las sentencias se encuentran la declaración y/o asignación de variables, la realización de operaciones básicas, y las llamadas a funciones.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	97/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada	

Licencia GPL de GNU

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * Authors: Julio A. de León, Jorge A. Solano and Hugo Zuñiga
 */

```

Comentarios

Es una buena práctica en cualquier lenguaje de programación realizar comentarios para documentar el programa. En C existen dos tipos de comentarios: el comentario por línea y el comentario por bloque.

El comentario por línea inicia cuando se insertan los símbolos // y termina con el salto de línea (hasta donde termina el renglón).

El comentario por bloque inicia cuando se insertan los símbolos /* y termina cuando se encuentran los símbolos */. Cabe resaltar que el comentario por bloque puede abarcar varios renglones.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	98/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Código con comentarios (se emplean los símbolos para indicar comentarios en C)

El siguiente programa muestra las sintaxis de comentarios en C:

Programa1.c

```
#include <stdio.h>

int main() {

    // Comentario por línea

    /* Comentario por bloque

        que puede ocupar
        varios renglones */

    // Este código compila y ejecuta

    /* pero no muestra salida alguna
        debido a que un comentario
        ya sea por línea o por bloque */

    // no es tomado en cuenta al momento

    // de compilar el programa,
    /* sólo sirve como documentación en el */

    /*
        código fuente
    */

    return 0;
}
```

La primera línea del Programa1.c `#include <stdio.h>` es una directiva al preprocesador de C que agrega la biblioteca (que contiene funciones pre-definidas en el lenguaje) que requiere el programa para su ejecución. En este caso particular, la biblioteca `stdio.h`

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página: 99/187 Sección ISO: 8.3 Fecha de emisión: 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

(standard input(i) output(o)) provee funciones externas necesarias para lectura y escritura de datos que se verán más adelante. Además, revisaremos la ubicación de estas directivas dentro de la estructura de un programa en C.

El conjunto de caracteres en C

Del mismo modo que en nuestro lenguaje habitual utilizamos un conjunto de caracteres para construir instrucciones que tengan significado, los programas que se realicen en C se escriben utilizando un conjunto de caracteres formado por lo siguiente:

- Las 26 letras minúsculas del alfabeto inglés (a b c d e f g h i j k l m n o p q r s t u v w x y z).
- Las 26 letras mayúsculas (A B C D E F G H I J K L M N O P Q R S T U V W X Y Z).
- Los 10 dígitos (1 2 3 4 5 6 7 8 9 0).
- Los símbolos especiales (@ ^ { } [] () & \$ % # ~ ‘ “ / ? ; : _ . , = / * - +).
- El espacio en blanco o barra espaciadora.

Nota: Lenguaje C es sensible a minúsculas y mayúsculas.

Esquema de la Estructura General de un programa en C (Figura 4 y 5).

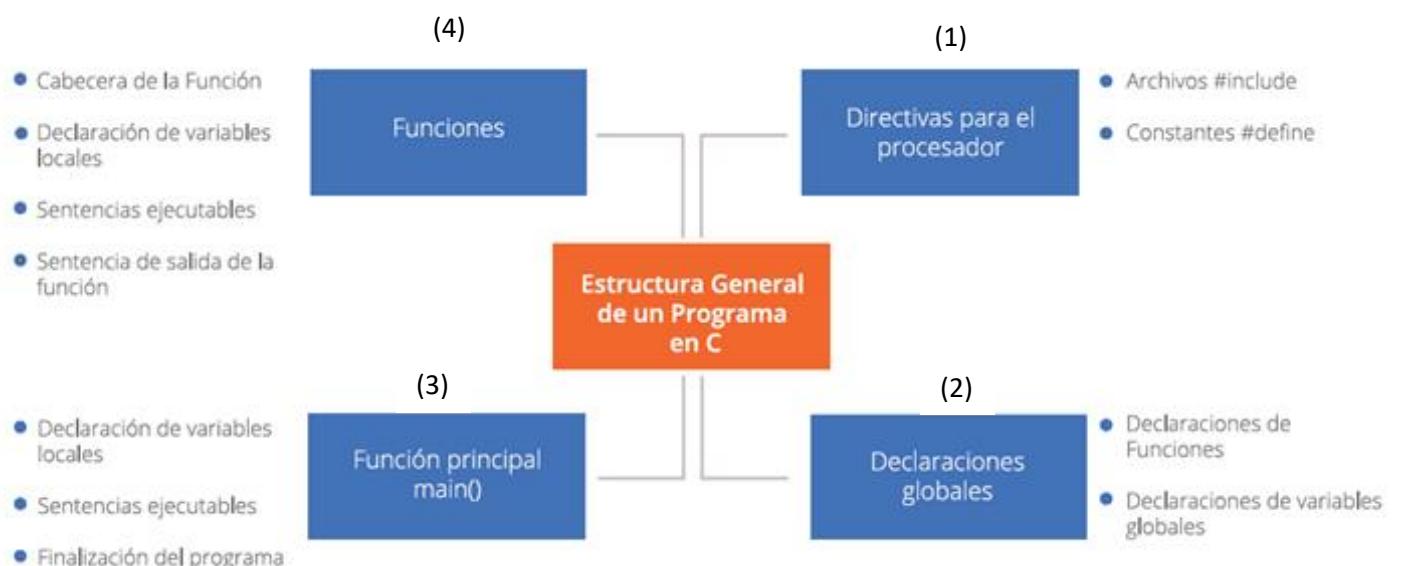


Figura 4: Estructura General de un programa en C

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	100/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

```
#include <stdio.h>           Directivas o bibliotecas
long cuadrado(long a);      Declaraciones globales
long v;

int main()
{
    long t;
    printf("\nPrograma de ejemplo para obtener el cuadrado de un número");
    printf("\nDa un número entero: ");
    scanf("%ld",&t);
    printf("\nEl cuadrado del número es: %ld\n",cuadrado(t));
    return 0;
}

long cuadrado(long a)
{
    return a*a;
}
```

Función principal main

Funciones secundarias o adicionales

Figura 5: Ejemplo de un programa con la estructura general de un programa en C

Palabras reservadas

Las palabras reservadas (Tabla 1) son palabras que tienen un significado predefinido estándar y sólo se pueden utilizar para su propósito ya establecido; no se pueden utilizar como identificadores definidos por el programador. En lenguaje C son las siguientes:

Tabla 1: Palabras reservadas

<i>auto</i>	<i>double</i>	<i>int</i>	<i>struct</i>
<i>break</i>	<i>else</i>	<i>long</i>	<i>switch</i>
<i>case</i>	<i>enum</i>	<i>register</i>	<i>typedef</i>
<i>char</i>	<i>extern</i>	<i>return</i>	<i>union</i>
<i>const</i>	<i>float</i>	<i>short</i>	<i>unsigned</i>
<i>continue</i>	<i>for</i>	<i>signed</i>	<i>void</i>
<i>default</i>	<i>goto</i>	<i>sizeof</i>	<i>volatile</i>
<i>do</i>	<i>if</i>	<i>static</i>	<i>while</i>

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	101/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Tipos de datos

El lenguaje C ofrece distintos tipos de datos (Tabla 2), cada uno de los cuales se puede encontrar representado de forma diferente en la memoria de la computadora.

Tabla 2: Tipos de datos.

Tipo	Descripción	Espacio en Memoria	Rango
<i>int</i>	Cantidad entera	2 bytes o una palabra* (varía de un compilador a otro)	-32767 a 32766
<i>char</i>	Carácter	1 byte	-128 a 127
<i>float</i>	Número en punto flotante (un número que incluye punto decimal y/o exponente)	1 palabra* (4 bytes)	-3.4E ³⁸ a 3.4 E ³⁸
<i>double</i>	Número en punto flotante de doble precisión (más cifras significativas y mayor valor posible del exponente)	2 palabras* (8 bytes)	-1.7E ³⁰⁸ a 1.7E ³⁰⁸

* *palabra*: conjunto de bits manejados por una arquitectura de procesador que se emplea para realizar referencias, operaciones y almacenamiento de información.

Existen algunos calificadores que se aplican a ciertos tipos de datos básicos. *short* (corto) y *long* (largo), califican a enteros, por ejemplo: *short int* y *long int*, en estos casos se puede omitir la palabra *int*. Los calificadores *signed* (con signo) y *unsigned* (sin signo) se usan para los tipos entero y carácter, por ejemplo: *unsigned char*, *unsigned long int*, *signed short*; si se omiten estos calificadores, por defecto se considera *signed*. Por último, el calificador *long* también se puede usar con *double*: *long double*; existen algunos compiladores de C que no reconocen el calificador *long* con el tipo *double*.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	102/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Los datos de tipo flotante o doble siempre tienen signo, por lo que no se debe usar el calificador de signo en estos tipos de datos básicos.

Código declaración de variables (se declaran variables y se asignan datos)

El siguiente programa muestra la manera en la que se declaran y asignan variables de diferentes tipos: numéricas (enteros y reales) y tipo carácter.

Programa2.c

```
#include <stdio.h>

int main() {
    short enteroNumero1 = 115;
    signed int enteroNumero2 = 55;
    unsigned long enteroNumero3 = 789;
    char caracterA = 65;
    char caracterB = 'B';

    float puntoFlotanteNumero1 = 89.8;
    unsigned double puntoFlotanteNumero2 = 238.2236;
    return 0;
}
```

No compila por la línea `unsigned double puntoFlotanteNumero2 = 238.2236;`, lo cual es correcto y es lo que se espera. La razón de que no compile es porque los valores reales siempre llevan signo y no es correcto usar algún calificador de signo, en este caso el *unsigned*.

Entrada y salida de datos

Se puede acceder a una función de entrada/salida de datos desde cualquier parte de un programa con simplemente escribir el nombre de la función, seguido de una lista de argumentos entre paréntesis. Los argumentos representan los datos que le son enviados a la función. Algunas funciones de entrada/salida no requieren argumentos, pero deben aparecer los paréntesis vacíos.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	103/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

La mayoría de las versiones de C incluyen una colección de archivos de cabecera que proporcionan la información necesaria para las distintas funciones de biblioteca. Cada archivo contiene generalmente la información necesaria para la utilización de un determinado grupo de funciones de biblioteca.

Estos archivos se incluyen en un programa mediante la instrucción `#include` al comienzo del programa. Como norma general, el archivo de cabecera requerido para la entrada/salida estándar se llama `stdio.h`.

La biblioteca `stdio.h` contiene diversas funciones, tanto para imprimir en la salida estándar (monitor) como para leer de la entrada estándar (teclado).

Se pueden escribir datos en el dispositivo de salida estándar, utilizando la función de biblioteca `printf`.

En términos generales, la función `printf` se escribe:

`printf(cadena de control, arg1, arg2, argn)`

En donde cadena de control hace referencia a una cadena de caracteres que contiene información sobre el formato de la salida y arg1, arg2, ... , argn son argumentos que representan los datos de salida.

Los argumentos pueden ser constantes, variables simples o nombres de arreglos o expresiones más complicadas.

La cadena de control contiene dos tipos de objetos: caracteres ordinarios que son copiados al flujo de salida y especificaciones de conversión; estos últimos causan la conversión y escritura de los argumentos que le siguen a la cadena de control, de acuerdo con el orden de aparición.

Cada especificación de conversión debe comenzar con el signo de porcentaje (%).

En su forma más sencilla, cada especificación de conversión estará formada por el signo de porcentaje, seguido de un carácter de conversión que indica el tipo de dato correspondiente.

Dentro de la cadena de control, las diferentes especificaciones de conversión pueden estar seguidas o pueden estar separadas por caracteres de espaciado (espacios en blanco, tabuladores o caracteres de nueva línea) (Tabla 3).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	104/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Tabla 3: Carácteres de conversión de los datos de salida de uso común [tabla], basada en Gottfried, 1997, *Programación en C* (p. 102).

Carácter de conversión	Significado
c	El dato es visualizado como un carácter
d	El dato es visualizado como un entero decimal con signo
e	El dato es visualizado como un valor en coma flotante con exponente
f	El dato es visualizado como un valor en coma flotante sin exponente
g	El dato es visualizado como un valor en coma flotante utilizando la conversión tipo e o tipo f según sea el caso. No se visualizan los ceros finales ni el punto decimal cuando no es necesario
i	El dato es visualizado como un entero con signo
o	El dato es visualizado como un entero octal, sin el cero inicial
s	El dato es visualizado como una cadena de caracteres
u	El dato es visualizado como un entero decimal sin signo
x	El dato es visualizado como un entero hexadecimal sin el prefijo 0x

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	105/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código que ejemplifica el uso de la función *printf*.

El siguiente programa imprime en pantalla diversos tipos de datos utilizando la función *printf*. Compila y ejecuta bien.

Programa3.c

```
#include <stdio.h>

int main() {
    //Declaración de variables
    int entero;
    float flotante;
    double doble;
    char caracter;
    //Asignación de variables
    entero = 14;
    flotante = 3.5f;
    doble = 6.8e10;
    caracter ='A';

    //Funciones de salida de datos en pantalla
    printf("La variable entera tiene valor: %i \n", entero);
    printf("La variable flotante tiene valor: %f \n", flotante);
    printf("La variable doble tiene valor: %f \n", doble);
    printf("La variable caracter tiene valor: %c \n", caracter);

    printf("Entero como octal: %o \n Como Hexadecimal %X \n", entero, entero);
    printf("Flotante con precisión: %5.2f \n", flotante);
    printf("Doble con precisión: %5.2f \n", doble);
    printf("Carácter como entero: %d \n", caracter);

    return 0;
}
```

Podemos observar que en los programas Programa2.c y Programa3.c hacen uso del operador = para asignarle valores a las variables. Este operador pertenece a un grupo de operadores denominados *operadores de asignación* que tiene el lenguaje C. En su momento, tu profesor enseñará cuáles son y su uso en una clase teórica.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	106/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Para imprimir con formato también se utilizan algunas secuencias de caracteres de escape. C maneja los siguientes (Tabla 4):

Tabla 4: Secuencias de caracteres

Especificador	Descripción
\a	Carácter de alarma
\b	Retroceso
\f	Avance de hoja
\n	Salto de línea
\r	Regreso de carro
\t	Tabulador horizontal
\v	Tabulador vertical
\0	Carácter nulo

Se pueden leer datos desde el dispositivo de entrada estándar, utilizando la función de biblioteca *scanf*.

La función *scanf* se puede utilizar para introducir cualquier combinación de valores numéricos, caracteres individuales y cadenas de caracteres. Es análoga a la función *printf*, con la diferencia de que su propósito es introducir datos en lugar de visualizarlos.

En términos generales, la función se escribe:

scanf(cadena de control, arg1, arg2, ... , argn)

Donde cadena de control hace referencia a una cadena de caracteres que contiene cierta información sobre el formato de los datos y arg1, arg2,..., argn son argumentos que representan los datos (Tabla 5).

Tabla 5: Caracteres de conversión de los datos de entrada de uso común [tabla], basada en Gottfried, 1997, *Programación en C* (p. 92).

Carácter de conversión	Significado
c	El dato es un carácter
d	El dato es un entero decimal
e	El dato es un valor en coma flotante
f	El dato es un valor en coma flotante
g	El dato es un valor en coma flotante

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	107/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

<i>h</i>	El dato es un entero corto
<i>i</i>	El dato es un entero decimal, octal o hexadecimal
<i>o</i>	El dato es un entero octal
<i>s</i>	El dato es una cadena de caracteres seguida de un carácter de espaciado (se añade automáticamente el carácter nulo \0 al final)
<i>u</i>	El dato es un entero sin signo
<i>x</i>	El dato es un entero hexadecimal

Los argumentos pueden ser variables o arreglos y sus tipos deben coincidir con los indicados por caracteres de conversión correspondientes en la cadena de control. Cada nombre de variable debe ser precedido por un *ampersand (&)*; El uso del carácter especial & es para referirse a la dirección de memoria.

Código que ejemplifica el uso de la función scanf

El siguiente programa muestra el almacenamiento de datos en variables. Compila y ejecuta bien.

Programa4.c

```
#include <stdio.h>

int main()
{
    int entero;
    float flotante;

    printf("Ingresa el valor entero: ");
    scanf("%i", &entero);
    printf("El valor ingresado es: %d\n", entero);

    printf("Ingresa el valor float: ");
    scanf("%f", &flotante);
    printf("El valor ingresado es: %f\n", flotante);

    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	108/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código que utiliza las funciones *printf* y *scanf*.

El siguiente código muestra cómo almacenar e imprimir variables. Compila y ejecuta bien.

Programa5.c

```
#include <stdio.h>

int main()
{
    int enteroNumero;
    char caracterA = 65; // Convierte el entero a carácter ASCII.
    double puntoFlotanteNumero;

    // Asignar valor de teclado a una variable.
    printf("Escriba un valor entero: ");
    scanf("%i", &enteroNumero);
    printf("Escriba un valor real: ");
    scanf("%lf", &puntoFlotanteNumero);

    // Imprimir valores con formato.
    printf("\nImprimiendo las variables \a\n");
    printf("\t Valor de enteroNumero = %i \a\n", enteroNumero);
    printf("\t Valor de caracterA = %c \a\n", caracterA);
    printf("\t Valor de puntoFlotanteNumero = %lf \a\n",
           puntoFlotanteNumero);

    printf("\t Valor de enteroNumero en base 16 = %x \a\n", enteroNumero);
    printf("\t Valor de caracterA en código hexadecimal = %x\n", caracterA);
    printf("\t Valor de puntoFlotanteNumero\n");
    printf("en notación científica = %e\n", puntoFlotanteNumero);

    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	109/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Operadores aritméticos

Los operadores aritméticos que maneja el lenguaje C se describen en Tabla 6:

Tabla 6: Operadores aritméticos

Operador	Operación	Uso	Resultado
+	Suma	125.78 + 62.5	188.28
-	Resta	65.3 - 32.33	32.97
*	Multiplicación	8.27 * 7	57.75
/	División	15 / 4	3.75
%	Módulo	4 % 2	0

Expresiones lógicas

Las expresiones lógicas están constituidas por números, caracteres, constantes o variables que están relacionados entre sí por operadores relationales y lógicos. Una expresión lógica puede tomar únicamente los valores verdadero o falso.

Los operadores de relación permiten *comparar* elementos numéricos, alfanuméricos, constantes o variables (Tabla 7).

Tabla 7: Operadores relationales

Operador	Operación	Uso	Resultado
==	Igual que Equivalente a	'h' == 'H'	Falso
!=	Diferente a	'a' != 'b'	Verdadero
<	Menor que	7 < 15	Verdadero
>	Mayor que	11 > 22	Falso
<=	Menor o igual	15 <= 22	Verdadero
>=	Mayor o igual	20 >= 35	Falso

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	110/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

NOTA: Es muy importante distinguir la diferencia entre el uso del operador relacional == y el operador de asignación =. El primero hace una comparación por lo que se puede leer como *igual que* o *equivalente a*, mientras que el operador = realiza una asignación.

Los operadores lógicos permiten formular condiciones complejas a partir de condiciones simples (Tabla 8).

Tabla 8: Operadores lógicos

Operador	Operación	Uso
!	No	! p
&&	Y	a > 0 && a < 11
	O	opc == 1 salir != 0

El uso de las expresiones lógicas se realizará en las siguientes prácticas.

Otros operadores (avanzado).

Los operadores a nivel de bits que maneja el lenguaje C se describen en la Tabla 9:

Tabla 9: Operadores a nivel de bits (avanzado)

Operador	Operación	Uso	Resultado
>>	Corrimiento a la derecha	8 >> 2	2
<<	Corrimiento a la izquierda	8 << 1	16
&	Operador AND	5 & 4	4
	Operador OR	3 2	3
~	Complemento ar-1	~2	1

Para el buen entendimiento del funcionamiento de estos operadores, se necesita del conocimiento de la representación de números en base dos y su manejo, y que no es objetivo de esta práctica.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	111/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código que ejemplifica el uso de operadores a nivel de bits.

El siguiente programa muestra cómo manipular números a nivel de bits:

- Corrimiento de bits a la izquierda y a la derecha
- Operador AND a nivel de bits
- Operador OR a nivel de bits

Compila y ejecuta bien.

Programa6.c

```
#include <stdio.h>
int main()
{
    short ocho, cinco, cuatro, tres, dos, uno;

    // 8 en binario: 0000 0000 0000 1000
    ocho = 8;
    // 5 en binario: 0000 0000 0000 0101
    cinco = 5;
    // 4 en binario: 0000 0000 0000 0100
    cuatro = 4;
    // 3 en binario: 0000 0000 0000 0011
    tres = 3;
    // 2 en binario: 0000 0000 0000 0010
    dos = 2;
    // 1 en binario: 0000 0000 0000 0001
    uno = 1;
    printf("Operadores aritméticos\n");
    printf("5 modulo 2 = %d\n",cinco%dos);
    printf("Operadores lógicos\n");
    printf("8 >> 2 = %d\n",ocho>>dos);
    printf("8 << 1 = %d\n",ocho<<1);
    printf("5 & 4 = %d\n",cinco&cuatro);
    printf("3 | 2 = %d\n",tres|dos);

    printf("\n");
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	112/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Lenguaje C posee operadores para realizar incrementos y decrementos de un número.

El operador **++** agrega una unidad (1) a su operando. Es posible manejar pre-incrementos (**++n**) o pos-incrementos (**n++**).

El operador **--** resta una unidad (1) a su operando. Se pueden manejar pre-decrementos (**--n**) o pos-decrementos (**n--**).

Bibliografía

- El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	113/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 07: Estructuras de selección



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	114/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 07: Estructuras de selección

Objetivo:

El alumno elaborará programas en lenguaje C que incluyan las estructuras de selección *if, if-else, switch* y ternaria (o condicional) para la resolución de problemas básicos.

Actividades:

- Elaborar expresiones lógicas/condicionales utilizadas en las estructuras de selección y realizar su evaluación.
- Elaborar un programa en lenguaje C para cada estructura de selección.

Introducción

Las estructuras de control de flujo en un lenguaje especifican el orden en que se realiza el procesamiento de datos.

Las estructuras de selección (o condicionales) permiten realizar una u otra acción con base en una expresión lógica. Las acciones posibles a realizar son mutuamente excluyentes, es decir, solo se puede ejecutar una a la vez dentro de toda la estructura.

Lenguaje C posee 3 estructuras de selección: la estructura *if-else*, la estructura *switch* y la estructura condicional o ternaria.

Licencia GPL de GNU

El software presente en esta práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 115/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```

*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*
* Author: Jorge A. Solano
*
*/

```

Estructura de control selectiva *if*

La estructura de control de flujo más simple es la estructura condicional *if*, su sintaxis es la siguiente:

```

if (expresión_lógica)

{
    // bloque de código a ejecutar
}

```

En esta estructura se evalúa la expresión lógica y, si se cumple (si la condición es verdadera), se ejecutan las instrucciones del bloque que se encuentra entre las llaves de la estructura. Si no se cumple la condición, se continúa con el flujo normal del programa.

NOTA 1: Si el bloque de código a ejecutar consta de una sola línea de código no es necesario el uso de las llaves.

NOTA 2: Como ya se explicó en la práctica anterior, la expresión lógica evaluada regresará como resultado un número entero. Dentro de las estructuras de control, el 0 indica que la expresión lógica es falsa y cualquier número diferente de 0 indica que la expresión lógica es verdadera.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	116/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Códigos (estructura de control selectiva if)

Este programa valida si el valor de la variable *a* es mayor al valor de la variable *b*.

Programa1.c

```
#include<stdio.h>
int main ()
{
    int a, b;
    a = 3;
    b = 2;
    if (a > b)
    {
        printf("\ta (%d) es mayor a b (%d).\n",a,b);
    }
    printf("\tEl programa sigue su flujo.\n");
    return 0;
}
```

Este programa comprueba que las condiciones son numéricas 0 → falso, ≠ 0 → verdadero.

Programa2.c

```
#include<stdio.h>
int main()
{
    if (0)
    {
        printf("Esta instrucción nunca se ejecuta\n");
        printf("porque la condición siempre es falsa (0).\n");
    }
    if (-38) // El bloque de código de esta estructura if
              // solo consta de una línea porque los comentarios
              // no son tomados en cuenta por el compilador.
              // La condición siempre es verdadera (diferente de 0)
        printf("Esta instrucción siempre se ejecuta.\n");
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 117/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Estructura de control selectiva *if-else*

La sintaxis de la estructura de control de flujo *if-else* es la siguiente:

```
if (expresión_lógica)
{
    // bloque de código a ejecutar
    // si la condición es verdadera
}
else
{
    // bloque de código a ejecutar
    // si la condición es falsa
}
```

Esta estructura evalúa la expresión lógica y si la condición es verdadera se ejecutan las instrucciones del bloque que se encuentra entre las primeras llaves, si la condición es falsa se ejecuta el bloque de código que está entre las llaves después de la palabra reservada *else*. Al final de que se ejecute uno u otro código, se continúa con el flujo normal del programa.

Es posible *anidar* varias estructuras *if-else*, es decir, dentro de una estructura *if-else* tener una o varias estructuras *if-else*.

Código (estructura de control selectiva *if-else*)

Este programa permite validar si un número es par o impar. El número se lee desde la entrada estándar (el teclado).

Programa3.c

```
#include <stdio.h>
int main()
{
    int num;
    printf("Ingrese un número:\n");
    scanf("%d",&num);
    if ( num%2 == 0 )
        printf("El número %d es par.\n",num);
    else
        printf("El número %d es impar.\n",num);
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	118/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Código (estructura de control selectiva *if-else* anidada)

Este programa ordena en forma descendente tres valores enteros dados. Los valores se leen desde la entrada estándar (el teclado).

Programa4.c

```
#include <stdio.h>
int main()
{
    int uno, dos, tres;
    printf ("Ingrese 3 números separados por espacios:\n");
    scanf ("%d %d %d", &uno, &dos, &tres);
    if (uno > dos)
    {
        if (dos > tres)
        {
            printf("%d es mayor a %d que es mayor a %d\n", uno, dos, tres);
        }
        else
        {
            if (uno > tres)
            {
                printf("%d es mayor a %d que es mayor a %d\n", uno, tres, dos);
            }
            else
            {
                printf("%d es mayor a %d que es mayor a %d\n", tres, uno, dos);
            }
        }
    }
    else
    {
        if (dos > tres)
        {
            if (tres > uno)
            {
                printf("%d es mayor a %d que es mayor a %d\n", dos, tres, uno);
            }
            else
            {
                printf("%d es mayor a %d que es mayor a %d\n", dos, uno, tres);
            }
        }
        else
        {
            printf("%d es mayor a %d que es mayor a %d\n", tres, dos, uno);
        }
    }

    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	119/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Estructura de control selectiva switch-case

La sintaxis de la estructura *switch-case* es la siguiente:

```
switch (opcion_a_evaluar)
{
    case valor1:
        /* Código a ejecutar*/
    break;
    case valor2:
        /* Código a ejecutar*/
    break;
    case valorN:
        /* Código a ejecutar*/
    break;
    default:
        /* Código a ejecutar*/
}
```

La estructura *switch-case* evalúa la variable que se encuentra entre paréntesis después de la palabra reservada *switch* y la compara con los valores constantes que posee cada caso (*case*). Los tipos de datos que puede evaluar esta estructura son enteros, caracteres y enumeraciones (se verá más adelante). Al final de cada caso se ejecuta la instrucción *break*, ya que si se omite esta palabra reservada se ejecutaría el siguiente caso, es decir, se utiliza para indicar que el bloque de código a ejecutar ya terminó y poder así salir de la estructura.

Si la opción a evaluar no coincide dentro de algún caso, entonces se ejecuta el bloque por defecto (*default*). El bloque por defecto normalmente se escribe al final de la estructura, pero se puede escribir en cualquier otra parte. Si se escribe en alguna otra parte el bloque debe terminar con la palabra reservada *break*.

Código (estructura de control selectiva switch-case)

Este programa permite elegir una opción del menú a partir del carácter ingresado. La opción se lee desde la entrada estándar (el teclado).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	120/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada	

Programa5.c

```
#include <stdio.h>
int main()
{
    char op = '\0';
    printf("\tMenú\n\n");
    printf("Elegir la opción deseada\n");
    printf("a) Ingresar\n");
    printf("b) Registrarse\n");
    printf("c) Salir\n");
    scanf("%c",&op);
    switch(op)
    {
        default:
            printf("Opción no válida.\n");
            break;
        case 'a':
            printf("Se seleccionó 'Ingresar'.\n");
            break;
        case 'b':
            printf("Se seleccionó 'Registrarse'.\n");
            break;
        case 'c':
            printf("Se seleccionó 'Salir'.\n");
            break;
    }
    return 0;
}
```

Código (estructura de control selectiva *switch-case*)

Este programa permite elegir una opción del menú a partir del entero.

Programa6.c

```
#include <stdio.h>
int main()
{
    int op;
    printf("\tMenú\n\n");
    printf("Elegir la opción deseada\n");
    printf("1) Ingresar\n");
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 121/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

```

printf("2) Registrarse\n");
printf("3) Salir\n");
scanf("%d",&op);
switch(op)
{
    case 1:
        printf("Se seleccionó 'Ingresar'\n");
        break;
    case 2:
        printf("Se seleccionó 'Registrarse'\n");
        break;
    case 3:
        printf("Se seleccionó 'Salir'\n");
        break;
    default:
        printf("Opción no válida\n");
}
return 0;
}

```

Enumeración

Existe otro tipo de dato constante conocido como enumeración. Una variable enumerador se puede crear de la siguiente manera:

```
enum identificador {VALOR1, VALOR2, ... , VALORN};
```

Para crear una enumeración se utiliza la palabra reservada *enum*, seguida de un identificador (nombre) y, entre llaves se ingresan los nombres de los valores que puede tomar dicha enumeración, separando los valores por coma. Los valores son elementos enteros y constantes (por lo tanto se escriben con mayúsculas).

Ejemplo

```
enum boolean {FALSE, TRUE};
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	122/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

La enumeración se llama '*boolean*' y contiene dos elementos, el primero (FALSE) posee el valor 0 y el siguiente (TRUE) posee el valor 1. Si hubiese más elementos en la enumeración, la numeración correría de manera ascendente.

Es posible cambiar el valor de un elemento, para ello solo se le asigna el valor deseado:

```
enum diasSemana {LUNES, MARTES, MIERCOLES=5, JUEVES, VIERNES};
```

Códigos (variables tipo enumeración)

Este programa crea diversas variables tipo enum (enumerador) y permite visualizar la manera en la que se maneja el tipo de dato.

Programa7.c

```
#include <stdio.h>
int main()
{
    // declaración de la enumeración
    enum boolean {NO, YES};

    // declaración de una variable tipo enumeración
    enum boolean valorBooleano;
    valorBooleano = YES;

    // Se comprueba que el valor de una enumeración es entero
    printf("%d\n", valorBooleano);

    // Se comprueba que el valor de una enumeración se puede reasignar
    enum diasSemana {LUNES, MARTES, MIERCOLES=5, JUEVES, VIERNES};
    printf("\n%d", LUNES);
    printf("\n%i", MARTES);
    printf("\n%d", MIERCOLES);
    printf("\n%i", JUEVES);
    printf("\n%d\n", VIERNES);

    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	123/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada	

Este programa permite elegir una opción del menú a partir del entero ingresado. La opción se lee desde la entrada estándar (el teclado).

Programa8.c

```
#include <stdio.h>
int main()
{
    // Los valores de una enumeración son enteros y constantes
    enum diasSemana {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO};
    int op;
    printf("Ingrese el día de la semana.\n");
    printf("1) Lunes\n");
    printf("2) Martes\n");
    printf("3) Miércoles\n");
    printf("4) Jueves\n");
    printf("5) Viernes\n");
    printf("6) Sábado\n");
    printf("7) Domingo\n");
    scanf("%d", &op);
    switch(op-1)
    {
        case LUNES:
        case MARTES:
            printf("Inicio de semana.\n");
            break;
        case MIERCOLES:
            printf("Mitad de semana.\n");
            break;
        case JUEVES:
            printf("¡Casi inicia el fin de semana!\n");
            break;
        case VIERNES:
        case SABADO:
            printf("¡Fin de semana!\n");
            break;
        case DOMINGO:
            printf("Día de descanso.\n");
            break;
        // No se necesita default
    }
    return 0x0;           // Valor entero en hexadecimal
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	124/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Estructura de control selectiva condicional

La estructura condicional (también llamado operador ternario) permite realizar una comparación rápida. Su sintaxis es la siguiente:

Condición ? SiSeCumple : SiNoSeCumple

Consta de tres partes, una condición y dos acciones a seguir con base en la expresión condicional. Si la condición se cumple (es verdadera) se ejecuta la instrucción que se encuentra después del símbolo '?'; si la condición no se cumple (es falsa) se ejecuta la instrucción que se encuentra después del símbolo ':'.

Código (Estructura de control selectiva condicional o ternaria).

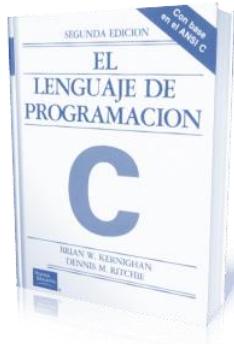
Este programa permite calcular el error matemático a partir de dos valores (a y b) ingresados desde la entrada estándar (el teclado), a partir de la fórmula: $E = |a - b|$. Donde a es el valor real y b es el valor aproximado o viceversa.

Programa9.c

```
#include <stdio.h>
int main()
{
    double a, b, res;
    printf("Calcular el error matemático E = |a - b|\n\n");
    printf("Ingrese el valor de a:\n");
    scanf("%lf",&a);
    printf("Ingrese el valor de b:\n");
    scanf("%lf",&b);
    res = a < b ? b-a : a-b;
    printf("El error matemático de\n");
    printf("|\nlf - %lf | es %lf\n", a, b, res);
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	125/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	126/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	
		22 / enero / 2025	
			Área/Departamento:
			Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 08: Estructuras de repetición



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	127/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía de práctica de estudio 08: Estructuras de repetición

Objetivo:

El alumno elaborará programas en C para la resolución de problemas básicos que incluyan las estructuras de repetición.

Actividades:

- Elaborar un programa que utilice la estructura *while* en la solución de un problema
- Elaborar un programa que requiera el uso de la estructura *do-while* para resolver un problema. Hacer la comparación con el programa anterior para distinguir las diferencias de operación entre *while* y *do-while*.
- Resolver un problema dado por el profesor que utilice la estructura *for* en lugar de la estructura *while*.

Introducción

Las estructuras de repetición son las llamadas también estructuras cíclicas, iterativas o de bucles. Permiten ejecutar un conjunto de instrucciones de manera repetida (o cíclica) mientras que la expresión lógica a evaluar se cumpla (sea verdadera).

En lenguaje C existen tres estructuras de repetición: *while*, *do-while* y *for*. Las estructuras *while* y *do-while* son estructuras repetitivas de propósito general.

Licencia GPL de GNU

El software presente en esta práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	128/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
Área/Departamento: Laboratorio de computación salas A y B			
La impresión de este documento es una copia no controlada			

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * Author: Jorge A. Solano
 *
 */

```

Estructura de control repetitiva **while**

La estructura repetitiva (o iterativa) *while* primero valida la expresión lógica y si ésta se cumple (es verdadera) procede a ejecutar el bloque de instrucciones de la estructura, el cual está delimitado por las llaves {} y regresa a validar la condición nuevamente, esto lo realiza mientras la condición sea verdadera. Cuando la condición no se cumple se continúa el flujo normal del programa sin ejecutar el bloque de la estructura, es decir, el bloque se puede ejecutar de cero a un determinado número de veces. Su sintaxis es la siguiente:

```
while (expresión_lógica) {
    // Bloque de código a repetir
    // mientras que la expresión
    // lógica sea verdadera.
}
```

Si el bloque de código a repetir consta de una sola sentencia, entonces se pueden omitir las llaves.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	129/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Códigos (estructura de repetición while)

Este programa genera la tabla de multiplicar de un número dado. El número se lee desde la entrada estándar (teclado).

Programa1.c

```
#include <stdio.h>

int main()
{
    int num, cont = 0;
    printf("\a----- Tabla de multiplicar -----\n");
    printf("Ingrese un número: \n");
    scanf("%d", &num);
    printf("La tabla de multiplicar del %d es:\n", num);
    while (++cont <= 10)
        printf("%d x %d = %d\n", num, cont, num*cont);
    return 0;
}
```

Este programa genera un ciclo infinito.

Programa2.c

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

/* Al igual que en la estructura if-else, 0 -> falso y diferente de 0 -> verdadero.
   El siguiente es un ciclo infinito porque la condición siempre es verdadera.
   Así mismo, debido a que el ciclo consta de una sola línea, las llaves {} son
   opcionales.*/

    while (100)
    {
        printf("Ciclo infinito.\nPara terminar el ciclo presione ctrl + c.\n");
    }
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	130/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Estructura de control repetitiva *do-while*

do-while es una estructura cíclica que ejecuta el bloque de código que se encuentra dentro de las llaves y después valida la condición, es decir, el bloque de código se ejecuta de una a un determinado número de veces. Su sintaxis es la siguiente:

```
do {
    /*
    Bloque de código que se ejecuta
    por lo menos una vez y se repite
    mientras la expresión lógica sea
    verdadera.
    */
} while (expresión_lógica);
```

Si el bloque de código a repetir consta de una sola sentencia, entonces se pueden omitir las llaves. Esta estructura de control siempre termina con el signo de puntuación punto y coma (;).

Código (estructura de repetición *do-while*)

Este programa obtiene el promedio de calificaciones ingresadas por el usuario. Las calificaciones se leen desde la entrada estándar (teclado). La inserción de calificaciones termina cuando el usuario presiona una tecla diferente de 'S' o 's'.

Programa3.c

```
#include <stdio.h>
int main ()
{
    char op = 'n';
    double sum = 0, calif = 0;
    int veces = 0;
    do
    {
        printf("\tSuma de calificaciones\n");
        printf("Ingrese la calificación:\n");
        scanf("%lf", &calif);
        veces++;
        sum = sum + calif;
        printf("¿Desea sumar otra? S/N\n");
        setbuf(stdin, NULL);           // limpia el buffer del teclado
        scanf("%c", &op);
        getchar();
    }
    while (op == 'S' || op == 's');
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 131/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

```

printf("El promedio de las calificaciones ingresadas es: %lf\n", sum/veces);
return 0;
}

```

Código (estructura de repetición *do-while*)

Este programa genera una calculadora básica

Programa4.c

```

#include <stdio.h>
int main ()
{
    int op, uno, dos;
    do
    {
        printf(" --- Calculadora ---\n");
        printf("\n¿Qué desea hacer\n");
        printf("1) Sumar\n");
        printf("2) Restar\n");
        printf("3) Multiplicar\n");
        printf("4) Dividir\n");
        printf("5) Salir\n");
        scanf("%d",&op);
        switch(op)
        {
            case 1:
                printf("\tSumar\n");
                printf("Introduzca los números a sumar separados por comas\n");
                scanf("%d, %d",&uno, &dos);
                printf("%d + %d = %d\n", uno, dos, (uno + dos));
                break;
            case 2:
                printf("\tRestar\n");
                printf("Introduzca los números a restar separados por comas\n");
                scanf("%d, %d",&uno, &dos);
                printf("%d - %d = %d\n", uno, dos, (uno - dos));
                break;
            case 3:
                printf("\tMultiplicar\n");
                printf("Introduzca los números a multiplicar separados por comas\n");
                scanf("%d, %d",&uno, &dos);
                printf("%d * %d = %d\n", uno, dos, (uno * dos));
                break;
            case 4:
                printf("\tDividir\n");
                printf("Introduzca los números a dividir separados por comas\n");

```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 132/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

```

        scanf("%d, %d", &uno, &dos);
        printf("%d / %d = %.2lf\n", uno, dos, ((double)uno / dos));
        break;
    case 5:
        printf("\tSalir\n");
        break;
    default:
        printf("\tOpción inválida.\n");
    }
}
while (op != 5);
}

```

Estructura de control de repetición **for**

El lenguaje C posee la estructura de repetición *for* la cual permite realizar repeticiones cuando generalmente el control de la repetición está definido sobre una variable contador. La sintaxis que generalmente se usa es la siguiente:

```

for (inicialización ; expresión_lógica ; operaciones por iteración) {
    /*
        Bloque de código
        a ejecutar
    */
}

```

La estructura *for* ejecuta 3 acciones básicas, dos antes y una después de ejecutar el bloque de código. La primera acción es la inicialización, en la cual se pueden definir variables e inicializar sus valores; esta acción solo se ejecuta una vez cuando se ingresa al ciclo y es opcional. La segunda acción consta de una expresión lógica, la cual se evalúa y, si ésta es verdadera, ejecuta el bloque de código, si no se cumple se continúa la ejecución del programa; esta acción es opcional. La tercera acción consta de un conjunto de operaciones que se realizan cada vez que termina de ejecutarse el bloque de código y antes de volver a validar la expresión lógica; esta acción también es opcional.

Código (estructura de repetición *for*).

Este programa genera un arreglo unidimensional de 5 elementos y se accede a cada elemento del arreglo a través de un ciclo *for*.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	133/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Programa5.c

```
#include <stdio.h>
int main ()
{
    int enteroNumAlumnos = 5;
    float realCalif = 0.0, realPromedio = 0.0;
    printf("\tPromedio de calificaciones\n");
    for (int indice = 0 ; indice < enteroNumAlumnos ; indice++)
    {
        printf("\nIngrese la calificación del alumn %d\n", indice+1);
        scanf("%f",&realCalif);
        realPromedio += realCalif;
    }

    printf("\nEl promedio de las calificaciones ingresadas es: %f\n",
    realPromedio/enteroNumAlumnos);
    return 0;
}
```

Define

Las líneas de código que empiezan con # son directivas del preprocesador, el cual se encarga de realizar modificaciones en el texto del código fuente, como reemplazar un símbolo definido con #define por un parámetro o texto, o incluir un archivo en otro archivo con #include.

define permite definir constantes o literales; se les nombra también como constantes simbólicas. Su sintaxis es la siguiente:

```
#define <nombre> <valor>
```

Al definir la constante simbólica con #define, se emplea un nombre y un valor. Cada vez que aparezca el nombre en el programa se cambiará por el valor definido. El valor puede ser numérico o puede ser texto.

Código (*define*)

Este programa define un valor por defecto para el tamaño del arreglo de tal manera que si el tamaño de éste cambia, solo se debe modificar el valor de la constante MAX.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 134/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

Programa6.c

```
#include <stdio.h>
#define MAX 5
int main ()
{
    int arreglo[MAX], cont;
    for (cont=0; cont<MAX; cont++)
    {
        printf("Ingrese el valor %d del arreglo: ", cont+1);
        scanf("%i", &arreglo[cont]);
    }

    printf("El valor ingresado para cada elemento del arreglo es:\n[" );
    for (cont=0; cont<MAX; cont++)
    {
        printf("%d\t", arreglo[cont]);
    }
    printf("]\n");
    return 0;
}
```

Cuando se compila el programa, se reemplazan la palabra MAX por el valor definido para la misma. Esto permite que, si el tamaño del arreglo cambia, solo se tiene que modificar el valor definido para MAX y en automático todos los arreglos y el recorrido de los mismos adquieran el nuevo valor (mientras se use MAX para definir el o los arreglos y para realizar los recorridos).

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	135/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 09: Arreglos unidimensionales



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes Ing. María Guadalupe Morales Nava	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	136/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 09: Arreglos unidimensionales

Objetivo:

El alumno utilizará arreglos de una dimensión en la elaboración de programas que resuelvan problemas que requieran agrupar datos del mismo tipo, alineados en un vector o lista.

Actividades:

- Elaborar programas en lenguaje C que empleen arreglos de una dimensión.
- Manipular este tipo de arreglos a través de índices.

Introducción

Un arreglo es un conjunto de datos contiguos del mismo tipo con un tamaño fijo definido al momento de crearse.

A cada elemento (dato) del arreglo se le asocia una posición particular, el cual se requiere indicar para acceder a un elemento en específico. Esto se logra a través del uso de índices.

Los arreglos pueden ser unidimensionales o multidimensionales. La dimensión del arreglo va de acuerdo con el número de índices que se requiere emplear para acceder a un elemento del arreglo. Así, si se requiere ubicar a un elemento en un arreglo de una dimensión (unidimensional), se requiere de un índice, para un arreglo de dos dimensiones se requieren dos índices y así sucesivamente. Los arreglos se utilizan para hacer más eficiente el código de un programa, así como manipular datos del mismo tipo con un significado común.

En esta práctica nos enfocaremos a trabajar con los arreglos unidimensionales.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	137/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Licencia GPL de GNU

El software presente en esta práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or
 * modify
 * it under the terms of the GNU General Public License as published
 * by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see
<http://www.gnu.org/licenses/>.
*
* Authors: María Guadalupe Morales, Cintia Quezada and Jorge A.
Solano
*
*/
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	138/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Arreglos unidimensionales

Un arreglo unidimensional de n elementos se almacena en la memoria de la siguiente manera (Figura 1):

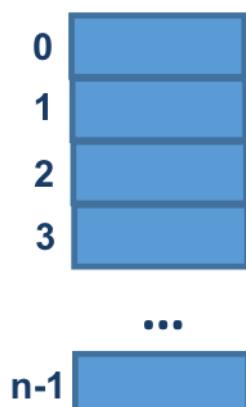


Figura 1. Almacenamiento en memoria de un arreglo unidimensional

La primera localidad del arreglo corresponde al índice 0 y la última corresponde al índice $n-1$, donde n es el tamaño del arreglo.

La sintaxis para definir un arreglo unidimensional en lenguaje C es la siguiente:

```
tipoDeDato nombre[tamaño]
```

Donde *nombre* se refiere al identificador del arreglo, *tamaño* es un número entero y define el número máximo de elementos que puede contener el arreglo. El *tipoDeDato* es el tipo de dato de los elementos del arreglo, el cual puede ser tipo entero, real, carácter o estructura.

NOTA: El tipo de dato *estructura* no se abordará en esta práctica.

Código (arreglo unidimensional empleando la estructura while)

A continuación, se muestra el código de un programa que genera un arreglo unidimensional de 5 elementos y que para poder acceder, recorrer y mostrar cada elemento del arreglo se usa la variable *indice* desde 0 hasta 4 haciendo uso de un ciclo *while*.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 139/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Programa1a.c

```
#include <stdio.h>
int main ()
{
    int lista[5] = {10, 8, 5, 8, 7}; // Se declara e inicializa el arreglo unidimensional
    int indice = 0;
    printf("\tLista\n");
    while (indice < 5) // Acceso a cada elemento del arreglo unidimensional usando while
    {
        printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
        indice += 1;          // Sentencia análoga a indice = indice + 1;
    }

    printf("\n");
    return 0;
}
```

Código (arreglo unidimensional empleando la estructura do-while)

En el siguiente código se genera un arreglo unidimensional de 5 elementos y que para poder acceder, recorrer y mostrar cada elemento del arreglo se usa la variable *indice* desde 0 hasta 4 haciendo uso de un ciclo *do-while*.

Programa1b.c

```
#include <stdio.h>
int main ()
{
    int lista[5] = {10, 8, 5, 8, 7}; // Se declara e inicializa el arreglo unidimensional
    int indice = 0;
    printf("\tLista\n");
    do // Acceso a cada elemento del arreglo unidimensional usando do-while
    {
        printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
        indice += 1;          // Sentencia análoga a indice = indice + 1;
    }
    while (indice < 5 );
    printf("\n");
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	140/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código (arreglo unidimensional empleando la estructura for)

A continuación, se muestra el código que genera un arreglo unidimensional de 5 elementos y que para poder acceder, recorrer y mostrar cada elemento del arreglo se usa la variable *indice* desde 0 hasta 4 haciendo uso de un ciclo *for*.

Programa1c.c

```
#include <stdio.h>
int main ()
{
    int lista[5] = {10, 8, 5, 8, 7}; // Se declara e inicializa el arreglo unidimensional
    int indice=0;
    printf("\tLista\n");
    // Acceso a cada elemento del arreglo unidimensional usando for
    for (indice = 0 ; indice < 5 ; indice++)
    {
        printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
    }

    printf("\n");
    return 0;
}
```

Código (arreglo unidimensional empleando la estructura for)

A continuación, se muestra un programa que genera un arreglo unidimensional de máximo 10 elementos. Para poder leer y almacenar datos en cada elemento y posteriormente mostrar el contenido de estos elementos se hace uso de un ciclo *for* respectivamente.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión	MADO-17 05 141/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Programa2.c

```
#include <stdio.h>
int main ()
{
    int lista[10]; // Se declara el arreglo unidimensional
    int indice=0;
    int numeroElementos=0;
    printf("\nDa un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo\n");
    scanf("%d",&numeroElementos);
    if((numeroElementos>=1) && (numeroElementos<=10))
    {
        // Se almacena un número en cada elemento del arreglo unidimensional usando for
        for (indice = 0 ; indice <= numeroElementos-1 ; indice++)
        {
            printf("\nDar un número entero para el elemento %d del arreglo ", indice );
            scanf("%d",&lista[indice]);
        }
        printf("\nLos valores dados son: \n");
        // Se muestra el número almacenado en cada elemento del arreglo unidimensional usando for
        for (indice = 0 ; indice <= numeroElementos-1 ; indice++)
        {
            printf("%d ", lista[indice] );
        }
    }
    else printf("el valor dado no es válido");
    printf("\n");
    return 0;
}
```

El código anterior también puede realizarse utilizando los ciclos while y do-while; se invita al lector a que pruebe la implementación de ambos casos.

Apuntadores

Un apuntador es una variable que contiene la dirección de una variable, es decir, hace referencia a la localidad de memoria de otra variable. Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a un dato.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 142/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

La sintaxis para declarar un apuntador y para asignarle la dirección de memoria de otra variable es, respectivamente:

```
TipoDeDato *apuntador, variable;
apuntador = &variable;
```

La declaración de una variable apuntador inicia con el carácter *. Cuando a una variable le antecede un ampersand (&), lo que se hace es referirse a la dirección de memoria donde se ubica el valor de dicha variable (es lo que pasa cuando se lee un dato con scanf).

Los apuntadores solo pueden apuntar a direcciones de memoria del mismo tipo de dato con el que fueron declarados; para referirse al contenido de dicha dirección, a la variable apuntador se le antepone * (Figura 2).

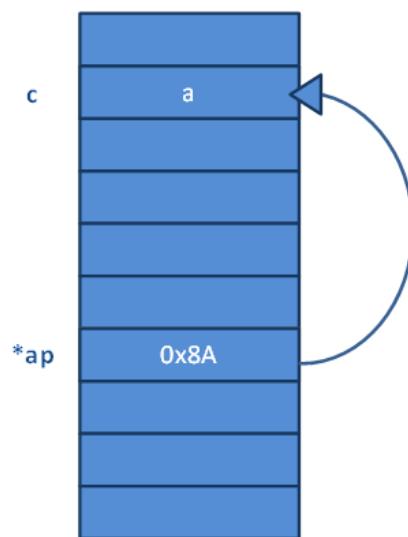


Figura 2. Un apuntador almacena la dirección de memoria de la variable a la que apunta

Código (apuntadores)

En el siguiente código se aprecia la declaración de un apuntador de tipo carácter y se muestra en pantalla, haciendo uso de apuntadores, el contenido de la variable c, así como el código ASCII del carácter 'a' que tiene almacenado dicha variable.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 143/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Programa3.c

```
#include <stdio.h>
int main ()
{
    char *ap, c = 'a'; // Se declara el apuntador ap de tipo alfanumérico
    ap = &c; //Se le asigna al apuntador la dirección de memoria de la variable c
    printf("Carácter: %c\n",*ap); /* Se imprime el contenido de la variable a la
                                    que apunta el apuntador ap */
    printf("Código ASCII: %d\n",*ap); /*Se imprime el código ASCII del carácter
                                    'a' */
    printf("Dirección de memoria: %d\n",ap);/*Se imprime la dirección de
                                    memoria que almacena el apuntador*/
    return 0;
}
```

Código (apuntadores)

Enseguida se observa el código de un programa que permite acceder a las localidades de memoria de distintas variables a través de un apuntador.

Programa4.c

```
#include<stdio.h>
int main ()
{
    int a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0};
    int *apEnt;
    apEnt = &a;
    printf("a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}\n");
    printf("apEnt = &a\n");
    /*A la variable b se le asigna el contenido de la variable a la que apunta
    apEnt*/
    b = *apEnt;
    printf("b = *apEnt \t-> b = %i\n", b);
    /*A la variable b se le asigna el contenido de la variable a la que apunta
    apEnt y se le suma uno*/
    b = *apEnt +1;
    printf("b = *apEnt + 1 \t-> b = %i\n", b);
    //La variable a la que apunta apEnt se le asigna el valor cero
    *apEnt = 0;
    printf("*apEnt = 0 \t-> a = %i\n", a);
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 144/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```

/*A apEnt se le asigna la dirección de memoria que tiene el elemento 0 del
arreglo c*/

apEnt = &c[0];
printf("apEnt = &c[0] \t-> apEnt = %i\n", *apEnt);
return 0;
}

```

Apuntadores y su relación con los arreglos.

Cabe mencionar que el nombre de un arreglo es un apuntador fijo al primero de sus elementos; por lo que las siguientes instrucciones, para el código de arriba, son equivalentes:

```

apEnt = &c[0];
apEnt = c;

```

Código (apuntadores)

El programa que se observa a continuación trabaja con aritmética de apuntadores para acceder a todos los valores que se encuentran almacenados en cada uno de los elementos de un arreglo.

Programa5.c

```

#include <stdio.h>
int main ()
{
    int arr[] = {5, 4, 3, 2, 1};
    int *apArr;//Se declara el apuntador apArr
    int x;
    apArr = arr;
    printf("int arr[] = {5, 4, 3, 2, 1};\n");
    printf("apArr = &arr[0]\n");
    x = *apArr; /*A la variable x se le asigna el contenido del arreglo arr en su
elemento 0*/
    printf("x = *apArr \t -> x = %d\n", x);
    x = *(apArr+1); /*A la variable x se le asigna el contenido del arreglo arr
en su elemento 1*/
}

```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	145/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

printf("x = *(apArr+1) \t -> x = %d\n", x);
x = *(apArr+2); /*A la variable x se le asigna el contenido del arreglo arr
en su elemento 2*/
printf("x = *(apArr+2) \t -> x = %d\n", x);
x = *(apArr+3); /*A la variable x se le asigna el contenido del arreglo arr
en su elemento 3*/
printf("x = *(apArr+3) \t -> x = %d\n", x);
x = *(apArr+4); /*A la variable x se le asigna el contenido del arreglo arr
en su elemento 4*/
printf("x = *(apArr+2) \t -> x = %d\n", x);
return 0;
}

```

Código (apuntadores en ciclo for)

El siguiente programa genera un arreglo unidimensional de 5 elementos y accede a cada uno de los elementos del arreglo haciendo uso de un apuntador, para ello se utiliza un ciclo *for*.

Programa6a.c

```

#include <stdio.h>
int main ()
{
    int lista[5] = {10, 8, 5, 8, 7};
    int *ap = lista; //Se declara el apuntador ap
    int indice;
    printf("\tLista\n");
    //Se accede a cada elemento del arreglo haciendo uso del ciclo for
    for (indice = 0 ; indice < 5 ; indice++)
    {
        printf("\nCalificación del alumno %d es %d", indice+1, (ap+indice));
    }
    printf("\n");
    return 0;
}

```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	146/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código (apuntadores en ciclo while)

A continuación, se muestra el código de un programa que genera un arreglo unidimensional de 5 elementos y accede a cada elemento del arreglo a través de un apuntador utilizando un ciclo *while*.

Programa6b.c

```
#include <stdio.h>
int main ()
{
    int lista[5] = {10, 8, 5, 8, 7};
    int *ap = lista; //Se declara el apuntador ap
    int indice = 0;

    printf("\tLista\n");
    //Se accede a cada elemento del arreglo haciendo uso del ciclo while
    while (indice < 5)
    {
        printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
        indice++;
    }
    printf("\n");
    return 0;
}
```

Código (apuntadores en ciclo do-while)

El programa que a continuación se observa genera un arreglo unidimensional de 5 elementos y accede a cada elemento del arreglo a través de un apuntador utilizando un ciclo *do while*.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	147/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Programa6c.c

```
#include <stdio.h>
int main ()
{
    int lista[5] = {10, 8, 5, 8, 7};
    int *ap = lista; //Se declara el apuntador ap
    int indice = 0;
    printf("\tLista\n");
    //Se accede a cada elemento del arreglo haciendo uso del ciclo do-while
    do
    {
        printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
        indice++;
    }
    while (indice < 5);
    printf("\n");
    return 0;
}
```

Código (apuntadores en cadenas)

El siguiente programa muestra el manejo básico de cadenas en lenguaje C.

Programa7.c

```
#include <stdio.h>
int main()
{
    char palabra[20];
    int i=0;
    printf("Ingrese una palabra: ");
    scanf("%s", palabra); /* Se omite & porque el propio nombre del arreglo de tipo cadena apunta, es decir, es equivalente a la dirección de comienzo del propio arreglo*/
    printf("La palabra ingresada es: %s\n", palabra);
    for (i = 0 ; i < 20 ; i++)
```

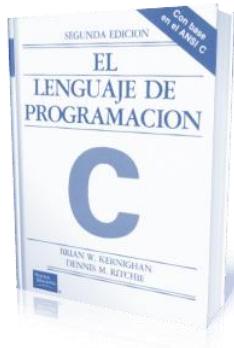
	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 148/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```

{
    printf("%c\n", palabra[i]);
}
return 0;
}

```

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	149/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 10: Arreglos multidimensionales



Elaborado por:	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes Ing. María Guadalupe Morales Nava	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	150/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 10: Arreglos multidimensionales

Objetivo:

El alumno utilizará arreglos de dos dimensiones en la elaboración de programas que resuelvan problemas que requieran agrupar datos del mismo tipo, en estructuras que utilicen dos índices.

Actividades:

- Resolver problemas que requieran el uso de un arreglo de dos dimensiones, a través programas en lenguaje C.
- Manipular este tipo de arreglos a través de índices.

Introducción

Un arreglo de dos dimensiones es un conjunto de datos contiguos del mismo tipo con un tamaño fijo definido al momento de crearse. Para acceder a un elemento en este tipo de arreglos se requiere el uso de dos índices.

Los arreglos se utilizan para hacer más eficiente el código de un programa, ya que la manipulación de datos del mismo tipo que son agrupados en un arreglo por tener un significado común, se realiza de una forma más clara y eficaz.

Licencia GPL de GNU

El software presente en esta práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or
 * modify
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 151/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

* it under the terms of the GNU General Public License as published
by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see
<http://www.gnu.org/licenses/>.
*
* Author: María Guadalupe Morales, Cintia Quezada and Jorge A. Solano
*
*/

```

Arreglos multidimensionales

Lenguaje C permite crear arreglos de varias dimensiones con la siguiente sintaxis:

tipoDato nombre[tamaño][tamaño]...[tamaño];

Donde *nombre* se refiere al identificador del arreglo, *tamaño* es un número entero y define el número máximo de elementos que puede contener el arreglo por dimensión (el número de dimensiones está determinado por el número de corchetes). El *tipoDeDatos* es el tipo de dato de los elementos del arreglo, el cual puede ser tipo entero, real, carácter o estructura.

De manera práctica se puede considerar que la primera dimensión corresponde a los renglones, la segunda a las columnas, la tercera al plano, y así sucesivamente. Sin embargo, en la memoria cada elemento del arreglo se guarda de forma contigua, por lo tanto, se puede recorrer un arreglo multidimensional con apuntadores.

Esta práctica estará enfocada exclusivamente al manejo de arreglos de dos dimensiones (bidimensionales).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	152/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Códigos (arreglos multidimensionales usando for)

A continuación, se observa un programa que genera un arreglo de dos dimensiones (arreglo multidimensional) y accede a cada uno de sus elementos por la posición que indica el renglón y la columna a través de dos ciclos `for`, uno anidado dentro de otro.

Programa1a.c

```
#include<stdio.h>
int main()
{
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, j;
    printf("Imprimir Matriz\n");
    for (i=0 ; i<3 ; i++) //Representa al renglón del arreglo
    {
        for (j=0 ; j<3 ; j++)//Representa a la columna del arreglo
        {
            printf("%d, ",matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

El siguiente programa genera un arreglo de dos dimensiones (arreglo multidimensional) y accede a sus elementos por la posición que indica el renglón y la columna a través de dos ciclos `for`, uno anidado dentro de otro, el contenido de cada elemento de este arreglo es la suma de sus índices.

Programa2a.c

```
#include<stdio.h>

int main()
{
    int i,j,a[5][5];
    for (i=0 ; i<5 ; i++)//Representa al renglón del arreglo
    {
        for (j=0 ; j<5 ; j++)//Representa a la columna del arreglo
        {
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 153/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

```

        a[i][j]=i+j;
        printf("\t%d, ",a[i][j]);
    }
    printf("\n");
}

return 0;
}

```

Códigos (arreglos multidimensionales usando while)

El código que se observa a continuación genera un arreglo de dos dimensiones (arreglo multidimensional) y accede a sus elementos por la posición que indica el renglón y la columna a través de dos ciclos *while*, uno anidado dentro de otro.

Programa1b.c

```

#include<stdio.h>
int main()
{
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, j;
    printf("Imprimir Matriz\n");
    i=0;
    while(i<3) //Representa al renglón del arreglo
    {
        j=0;
        while (j<3)//Representa a la columna del arreglo
        {
            printf("%d, ",matriz[i][j]);
            j++;
        }
        printf("\n");
        i++;
    }
    return 0;
}

```

Enseguida se muestra el código de un programa que permite generar un arreglo de dos dimensiones (arreglo multidimensional) y accede a sus elementos por la posición que indica el renglón y la columna a través de dos ciclos *while*, uno anidado dentro de otro, el contenido de cada elemento de este arreglo es la suma de sus índices.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 154/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Programa2b.c

```
#include<stdio.h>
int main()
{
    int i,j,a[5][5];
    i=0;
    while (i<5) //Representa al renglón del arreglo
    {
        j=0;
        while (j<5) //Representa a la columna del arreglo
        {
            a[i][j]=i+j;
            printf("\t%d, ",a[i][j]);
            j++;
        }
        printf("\n");
        i++;
    }
    return 0;
}
```

Códigos (arreglos multidimensionales usando do-while)

A continuación, se presenta un código que permite la generación de un arreglo de dos dimensiones (arreglo multidimensional) y se puede acceder a cada uno de sus elementos por la posición que indica el renglón y la columna a través de dos ciclos *do-while*, uno anidado dentro de otro.

Programa1c.c

```
#include<stdio.h>
int main()
{
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, j;
    printf("Imprimir Matriz\n");
    i=0;

    do //Representa al renglón del arreglo
    {
        j=0;
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	155/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

```

do //Representa a la columna del arreglo
{
    printf("%d, ", matriz[i][j]);
    j++;
}
while (j<3);
printf("\n");
i++;
}
while(i<3);
return 0;
}

```

Como puede observarse en el programa que se lee a continuación, se genera un arreglo de dos dimensiones (arreglo multidimensional) y se accede a todos sus elementos por la posición que indica el renglón y la columna a través de dos ciclos *do-while*, uno anidado dentro de otro, el contenido de cada uno de los elementos de este arreglo es la suma de sus índices.

Programa2c.c

```

#include<stdio.h>
int main()
{
    int i,j,a[5][5];
    i=0;
    do //Representa al renglón del arreglo
    {
        j=0;
        do //Representa a la columna del arreglo
        {
            a[i][j]=i+j;
            printf("\t%d, ",a[i][j]);
            j++;
        }
        while (j<5);
        printf("\n");
        i++;
    }
    while (i<5);
    return 0;
}

```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 156/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

A continuación, se muestra un programa que genera un arreglo multidimensional de máximo 10 renglones y 10 columnas, para poder almacenar datos en cada elemento y posteriormente mostrar el contenido de esos elementos se hace uso de ciclos *for* anidados respectivamente.

Programa3.c

```
#include <stdio.h>
int main ()
{
    int lista[10][10]; // Se declara el arreglo multidimensional
    int i,j;
    int renglon,columna;
    printf("\nDa el número de renglones y columnas separados con coma, ");
    printf("cuyo valor no sea mayor a 10 ");
    scanf("%d,%d",&renglon,&columna);
    if(((renglon>=1) && (renglon<=10))&&((columna>=1) && (columna<=10)))
    {
        // Acceso a cada elemento del arreglo multidimensional usando for
        for (i= 0 ; i <= renglon-1 ; i++)
        {
            for(j= 0 ; j <= columna-1 ; j++)
            {
                printf("\nNúmero para el elemento [%d][%d] del arreglo ", i,j );
                scanf("%d",&lista[i][j]);
            }
        }
        printf("\nLos valores dados son: \n");
        // Acceso a cada elemento del arreglo multidimensional usando for
        for (i= 0 ; i <= renglon-1 ; i++)
        {
            for(j= 0 ; j <= columna-1 ; j++)
            {
                printf("%d ", lista[i][j]);
            }
        }
        printf("\n");
    }
    else printf("Los valores dados no es válido");
    printf("\n");
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 157/187 8.3 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

El código anterior también puede realizarse utilizando los ciclos *while* y *do-while*, se invita al lector a que pruebe la implementación de ambos casos.

Apuntadores y su relación con arreglos de dos dimensiones

Recordemos que un apuntador es una variable que contiene la dirección de una variable, es decir, hace referencia a la localidad de memoria de otra variable. Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a un dato.

La sintaxis para declarar un apuntador y para asignarle la dirección de memoria de otra variable es, respectivamente:

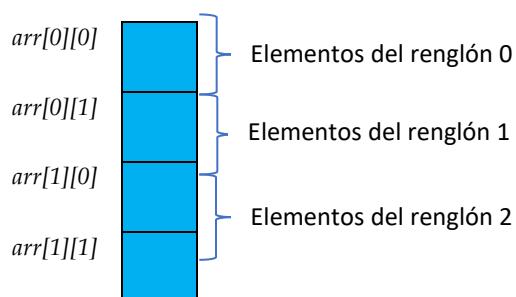
```
TipoDeDato *apuntador, variable;
apuntador = &variable;
```

La declaración de una variable apuntador inicia con el carácter *. Cuando a una variable le antecede un ampersand (&), lo que se hace es referirse a la dirección de memoria donde se ubica el valor de dicha variable (es lo que pasa cuando se lee un dato con scanf).

Los apuntadores solo pueden apuntar a direcciones de memoria del mismo tipo de dato con el que fueron declarados; para acceder al contenido de dicha dirección, a la variable apuntador se le antepone *.

Como se mencionó al inicio de esta práctica, en la memoria cada elemento del arreglo se guarda de forma contigua, por lo tanto, se puede recorrer un arreglo multidimensional con apuntadores.

Por ejemplo, los elementos de un arreglo de 3 renglones y 2 columnas de nombre *arr* se almacenan en memoria como lo muestra la Figura 1.



	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 158/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

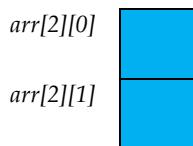


Figura 1. Almacenamiento en memoria de un arreglo de 3X2

Como podemos observar en la Figura 1, los elementos de un arreglo bidimensional se almacenan por renglones; esto se debe considerar al momento de acceder a los elementos del arreglo usando apuntadores.

Códigos (arreglos multidimensionales con apuntadores)

El programa siguiente genera un arreglo de dos dimensiones (arreglo multidimensional) y accede a sus elementos a través de un apuntador utilizando un ciclo for.

Programa4a.c

```
#include<stdio.h>
int main()
{
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, cont=0, *ap;
    ap = *matriz; //Esta sentencia es análoga a: ap = &matriz[0][0];
    printf("Imprimir Matriz\n");
    for (i=0 ; i<9 ; i++)
    {
        if (cont == 3) //Se imprimió un renglón y se hace un salto de línea
        {
            printf("\n");
            cont = 0; //Inicia conteo de elementos del siguiente renglón
        }
        printf("%d\t",*(ap+i));//Se imprime el siguiente elemento de la matriz
        cont++;
    }
    printf("\n");
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	159/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

El código del siguiente programa genera un arreglo de dos dimensiones (arreglo multidimensional) y accede a sus elementos a través de un apuntador utilizando un ciclo *while*.

Programa4b.c

```
#include<stdio.h>
int main()
{
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, cont=0, *ap;
    ap = *matriz;//Esta sentencia es análoga a: ap = &matriz[0][0];
    printf("Imprimir Matriz\n");
    i=0;
    while (i<9)
    {
        if (cont == 3) //Se imprimió un renglón y se hace un salto de línea
        {
            printf("\n");
            cont = 0; //Inicia conteo de elementos del siguiente renglón
        }
        printf("%d\t",*(ap+i));//Se imprime el siguiente elemento de la matriz

        cont++;
        i++;
    }
    printf("\n");
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 160/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

La generación de un arreglo de dos dimensiones (arreglo multidimensional) y el acceso a sus elementos a través de un apuntador utilizando un ciclo *do-while* se puede observar en el programa siguiente:

Programa4c.c

```
#include<stdio.h>
int main()
{
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, cont=0, *ap;
    ap = *matriz;//Esta sentencia es análoga a: ap = &matriz[0][0];
    printf("Imprimir Matriz\n");
    i=0;
    do
    {
        if (cont == 3) //Se imprimió un renglón y se hace un salto de línea
        {
            printf("\n");
            cont = 0; //Inicia conteo de elementos del siguiente renglón
        }
        printf("%d\t",*(ap+i)); //Se imprime el siguiente elemento de la matriz
        cont++;
        i++;
    }
    while (i<9);
    printf("\n");
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	161/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	162/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 11: Funciones



Elaborado por	Actualizado por:	Revisado por:
M.C. Cintia Quezada Reyes Ing. Maricela Castañeda Perdomo.	M.I. Manuel Castañeda Castañeda.	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	163/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada	

Guía práctica de estudio 11: Funciones

Objetivo:

El alumnado elaborará programas en C donde la solución del problema se divide en funciones. Distinguirá lo que es el prototipo o firma de una función y la implementación de ella, asimismo manipulará parámetros tanto en la función principal como en otras.

Actividades:

- Implementar en un programa en C la solución de un problema dividido en funciones.
- Elaborar un programa en C que maneje argumentos en la función principal.
- En un programa en C, manejar variables y funciones estáticas.

Introducción

Como ya se mencionó, un programa en lenguaje C consiste en una o más funciones. C permite tener dentro de un archivo fuente varias funciones, esto con el fin de dividir las tareas y que sea más fácil la depuración, la mejora y el entendimiento del código.

En lenguaje C la función principal se llama *main*. Cuando se ordena la ejecución del programa, el sistema inicia la ejecución de las instrucciones que se encuentran dentro de la función *main*, y ésta puede llamar a ejecutar otras funciones, que a su vez éstas pueden llamar a ejecutar a otras funciones, y así sucesivamente.

Licencia GPL de GNU

El software presente en esta práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión	MADO-17 05 164/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

```

* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*
* Author: Jorge A. Solano
*
*/

```

Funciones

La sintaxis básica para definir una función es la siguiente:

```

tipoValorRetorno nombre (parámetros)
{
    // bloque de código de la función
}

```

El nombre de la función se refiere al identificador con el cual se ejecutará la función; se debe seguir la notación de camello.

Una función puede recibir parámetros, los cuales son datos de entrada con los que trabajará la función; dichos parámetros se deben definir dentro de los paréntesis de la función, separados por comas e indicando su tipo de dato, de la siguiente forma:

(*tipoDato nom1, tipoDato nom2, tipoDato nom3...*)

El tipo de dato puede ser cualquiera de los vistos hasta el momento (entero, real, carácter o arreglo) y el nombre debe seguir la notación de camello. Los parámetros de una función son opcionales.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	165/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

El tipo del valor de retorno de una función indica el tipo de dato que va a regresar la función al terminar el bloque de código de ésta a través de la sentencia *return*. El valor de retorno puede ser cualquiera de los tipos de datos vistos hasta el momento (entero, real, carácter o arreglo), aunque también se puede poner como tipo de la función la palabra reservada *void* (genérico o vacío) en cuyo caso se puede omitir la sentencia *return*.

Las distribuciones de los compiladores de C que siguen los estándares internacionales del lenguaje revisan que las funciones estén definidas o declaradas antes de ser invocadas. Por lo que una buena práctica es declarar todas las funciones al inicio del programa. Una declaración, prototipo o firma de una función tiene la siguiente sintaxis:

tipoValorRetorno nombre (*parámetros*);

La firma de una función está compuesta por tres elementos: el tipo del valor de retorno de la función, el nombre de la función y los parámetros que recibe la función; finaliza con punto y coma (;). Los nombres de los parámetros no necesariamente deben ser iguales a los que se encuentran en la definición de la función. Las funciones definidas en el programa no necesariamente deberán ser declaradas; esto dependerá de su ubicación en el código.

Código (funciones)

El siguiente programa contiene dos funciones: la función *main* y la función *imprimir*. La función *main* manda llamar a la función *imprimir*. La función *imprimir* recibe como parámetro un arreglo de caracteres y lo recorre de fin a inicio imprimiendo cada carácter del arreglo.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	166/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Programa1.c

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

// Prototipo o firma de las funciones del programa
void imprimir(char[]);

// Definición o implementación de la función main
int main(){
    char nombre[] = "Facultad de Ingeniería";
    imprimir(nombre);
}

// Implementación de las funciones del programa
void imprimir(char s[]){
    int tam;
    for ( tam=strlen(s)-1 ; tam>=0 ; tam-- )
        printf("%c", s[tam]);
    printf("\n");
}
```

NOTA: `strlen` es una función que recibe como parámetro un arreglo de caracteres y regresa como valor de retorno un entero que indica la longitud de la cadena. La función se encuentra dentro de la biblioteca `string.h`, por eso se incluye ésta al principio del programa.

Ámbito o alcance de las variables

Las variables declaradas dentro de un programa tienen un tiempo de vida que depende de la posición donde se declaren. En C existen dos tipos de variables con base en el lugar donde se declaren: variables locales y variables globales.

Como ya se vio, un programa en C puede contener varias funciones. Las variables que se declaren dentro de cada función se conocen como variables locales (a cada función). Estas variables existen al momento de que la función es llamada y desaparecen cuando la función llega a su fin.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	167/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código (variables locales)

El siguiente programa muestra la declaración y uso de variables locales en la función de *suma*.

[Programa2.c](#)

```
#include <stdio.h>

void sumar(); // prototipo de la función
int main()
{
    sumar(); // llamado de la función suma
}
void sumar() // función suma
{
    int z, x=5, y=10; //variables locales
    z=x+y;
    printf("%i",z);
}
```

Las variables que se declaran fuera de cualquier función se llaman variables globales. Las variables globales existen durante la ejecución de todo el programa y pueden ser utilizadas por cualquier función.

Código (variables globales)

El siguiente programa muestra la declaración de la variable global *resultado*, la cual es utilizada en ambas funciones. Las variables globales en C se inician, por defecto, con el valor de 0.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 168/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Programa3.c

```
#include <stdio.h>

int resultado; //variable global

void multiplicar(); // prototipo de la función

int main()
{
    multiplicar(); //llamado de la función multiplicar
    printf("%i",resultado);
    return 0;
}
void multiplicar() //función multiplicar
{
    resultado = 5 * 4;
}
```

Código (Ámbito de las variables)

Este programa contiene dos funciones: la función *main* y la función *incremento*. La función *main* manda llamar a la función *incremento* dentro de un ciclo *for*. La función *incremento* aumenta el valor de la variable *enteraGlobal* cada vez que es invocada.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17	
		Versión:	05	
		Página	169/187	
		Sección ISO	8.3	
		Fecha de emisión	22 / enero / 2025	
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada				

Programa4.c

```
#include<stdio.h>

void incremento();

/* La variable enteraGlobal es vista por todas
   las funciones (main e incremento) */
int enteraGlobal;

int main()
{
    // La variable cont es local a la función main
    int cont;
    enteraGlobal = 0; // La función main accede a la variable global
    for (cont=0 ; cont<5 ; cont++)
    {
        incremento();
    }

    return 0;
}

void incremento()
{
    // La variable enteraLocal es local a la función incremento
    int enteraLocal = 5;
    enteraGlobal += 2;
    printf("global(%i) + local(%i) = %d\n",enteraGlobal, enteraLocal, enteraGlobal+enteraLocal);
}
```

Argumentos para la función *main*

Como se mencionó, la firma o prototipo de una función está compuesta por tres elementos: el tipo del valor de retorno de la función, el nombre de la función y los parámetros de la función.

Entonces, la función *main* también puede tener parámetros. Si se declaran parámetros en la función *main* deben ser dos: el primero se declara de tipo entero y el segundo se declara como un arreglo de cadenas; un ejemplo de la cabecera de la función *main* con parámetros es:

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	170/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

```
int main (int argc, char ** argv)
```

La longitud del arreglo se guarda en el primer parámetro (argument counter) y el arreglo de cadenas se guarda en el segundo parámetro (argument vector). Para enviar argumentos a la función *main*, el programa se debe ejecutar de la siguiente manera, en la línea de comandos:

- En plataforma Linux/Unix
./nombrePrograma arg1 arg2 arg3 ...
- En plataforma Windows
nombrePrograma.exe arg1 arg2 arg3 ...

Esto es, el nombre del programa seguido de los argumentos de entrada.

De tal manera que los parámetros recibirán los siguientes valores, suponiendo los nombres de la cabecera de ejemplo mostrada anteriormente:

argc: número de cadenas contabilizadas en la línea de comandos, incluyendo el nombre del programa.

argv: arreglo con las cadenas de la línea de comandos:

 argv[0] es el nombre del programa.

 argv[1] es la cadena de arg1

 argv[2] es la cadena de arg2

y así sucesivamente.

Código (argumentos función main)

Este programa muestra los argumentos enviados al ejecutarlo.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 171/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Programa5.c

```
#include <stdio.h>
#include <string.h>
int main(int argc, char** argv)
{
    if (argc == 1)
    {
        printf("El programa no contiene argumentos.\n");
        return 0;
    }

    printf("Los elementos del arreglo argv son:\n");
    for (int cont = 0 ; cont < argc ; cont++)
    {
        printf("argv[%d] = %s\n", cont, argv[cont]);
    }

    return 0;
}
```

Estático

El Lenguaje C permite definir elementos estáticos. La sintaxis para declarar elementos estáticos es la siguiente:

```
static tipoDato nombre;
static tipoValorRetorno nombre(parámetros);
```

Es decir, tanto a la declaración de una variable como a la firma de una función solo se le agrega la palabra reservada *static* al inicio de estas.

El atributo *static* en una variable hace que ésta permanezca en memoria desde su creación y durante toda la ejecución del programa, lo que quiere decir que su valor se mantendrá hasta que el programa llegue a su fin.

El atributo *static* en una función hace que esa función sea accesible solo dentro del mismo archivo, lo que impide que fuera de la unidad de compilación se pueda acceder a la función.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	172/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código (variable estática)

Este programa contiene dos funciones: la función *main* y la función *llamarFuncion*. La función *main* manda llamar a la función *llamarFuncion* dentro de un ciclo *for*. La función *llamarFuncion* crea una variable estática e imprime su valor.

Programa6.c

```
#include <stdio.h>
void llamarFuncion();

int main()
{
    for (int j=0 ; j < 5 ; j++)
    {
        llamarFuncion();
    }
}

void llamarFuncion()
{
    /* Solo la primera vez que se llame a esta función se creará y se le asignará
       el valor de 0 a la variable estática numVeces */
    static int numVeces = 0;
    printf("Esta función se ha llamado %d veces.\n",++numVeces);
}
```

La variable *numVeces* tiene un **alcance local** dentro de la función *llamarFuncion()*, lo que significa que solo puede ser utilizada dentro de esa función. Sin embargo, al ser declarada como **static**, su valor no se destruye cuando la función termina.

Una variable **static** tiene una **duración de vida global**, lo que significa que permanece en memoria durante toda la ejecución del programa. Aunque *numVeces* solo es accesible desde *llamarFuncion()*, su valor se conserva entre diferentes llamadas a la función.

La variable *numVeces* se inicializa a **0** solo **la primera vez** que se ejecuta la función *llamarFuncion()*. En las siguientes llamadas, la línea *static int numVeces = 0;* no se ejecuta nuevamente porque la variable ya existe en memoria.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	173/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Código (función estática). Este ejemplo consta de dos archivos: **funcEstatica.c** y **calculadora.c**.

El programa funcEstatica.c contiene las funciones de una calculadora básica: suma, resta, producto y cociente.

funcEstatica.c

```
//##### funcEstatica.c #####
#include <stdio.h>

int suma(int,int);
static int resta(int,int);

int producto(int,int);
static int cociente (int,int);

int suma (int a, int b)
{
    return a + b;
}

static int resta (int a, int b)
{
    return a - b;
}

int producto (int a, int b)
{
    return (int)(a*b);
}

static int cociente (int a, int b)
{
    return (int)(a/b);
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-17 05 174/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

El programa calculadora.c contiene el método principal, el cual invoca a las funciones del archivo funcEstatica.c.

calculadora.c

```
//##### calculadora.c #####
#include <stdio.h>

int suma(int,int);
//static int resta(int,int);
int producto(int,int);
//static int cociente (int,int);

int main()
{
    printf("5 + 7 = %i\n",suma(5,7));
    //printf("9 - 77 = %d\n",resta(9,77));
    printf("6 * 8 = %i\n",producto(6,8));
    //printf("7 / 2 = %d\n",cociente(7,2));
}
```

Cuando se compilan los dos archivos al mismo tiempo con la línea de comandos (gcc funcEstatica.c calculadora.c -o exe), las funciones suma y producto son accesibles desde el archivo calculadora y, por tanto, se genera el código ejecutable. Si se quitan los comentarios y se intenta compilar los archivos se enviará un error, debido a que las funciones son estáticas y no pueden ser accedidas fuera del archivo funcEstaticas.c.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	175/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	176/187
		Sección ISO	8.3
Facultad de Ingeniería		Fecha de emisión	22 / enero / 2025
			Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 12: Lectura y escritura de datos



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	177/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 12: Lectura y escritura de datos

Objetivo:

El alumnado elaborará programas en lenguaje C que requieran el uso de archivos de texto plano en la resolución de problemas, entendiendo a los archivos como un elemento de almacenamiento secundario.

Actividades:

- A través de programas en C, emplear las funciones para crear, leer, escribir y sobrescribir archivos de texto plano.
- Manipular archivos empleando los diferentes tipos de acceso a ellos.

Introducción

Un archivo es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros que son del mismo tipo, pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

El lenguaje C permite manejar la entrada y la salida de datos desde o hacia un archivo, respectivamente, a través del uso de la biblioteca de funciones de la cabecera `stdio.h`.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	178/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Licencia GPL de GNU

El software presente en esta práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * Author: Jorge A. Solano
 *
 */
```

Apuntador a archivo

Un apuntador a un archivo es un hilo común que unifica el sistema de Entrada/Salida (E/S) con un *buffer* donde se transportan los datos.

Un apuntador a archivo señala a la información que contiene y define ciertas características sobre él, incluyendo el nombre, el estado y la posición actual del archivo.

Los apuntadores a un archivo se manejan en lenguaje C como variables apuntador de tipo FILE que se define en la cabecera *stdio.h*. La sintaxis para obtener una variable apuntador de archivo es la siguiente:

FILE *F;

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	179/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Abrir archivo

La función *fopen()* abre una secuencia para que pueda ser utilizada y la asocia a un archivo. Su estructura es la siguiente:

```
*FILE fopen(char *nombre_archivo, char *modo);
```

Donde *nombre_archivo* es un puntero a una cadena de caracteres que representan un nombre válido del archivo y puede incluir una especificación del directorio. La cadena a la que apunta *modo* determina cómo se abre el archivo.

Existen diferentes modos de apertura de archivos, los cuales se mencionan a continuación, además de que se pueden utilizar más de uno solo:

- r: Abre un archivo de texto para lectura.
- w: Crea un archivo de texto para escritura.
- a: Abre un archivo de texto para añadir.
- r+: Abre un archivo de texto para lectura / escritura.
- w+: Crea un archivo de texto para lectura / escritura.
- a+: Añade o crea un archivo de texto para lectura / escritura.
- rb: Abre un archivo en modo lectura y binario.
- wb: Crea un archivo en modo escritura y binario.

Cerrar archivo

La función *fclose()* cierra una secuencia que fue abierta mediante una llamada a *fopen()*. Escribe la información que se encuentre en el *buffer* al disco y realiza un cierre formal del archivo a nivel del sistema operativo.

Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa. La firma de esta función es:

```
int fclose (FILE *apArch);
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 180/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Donde *apArch* es el apuntador al archivo devuelto por la llamada a *fopen()*. Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.

Código (abrir cerrar archivo)

Este programa permite abrir un archivo en modo de lectura, de ser posible ya que el archivo debe existir o debe tener permiso de lectura.

Programa1.c

```
#include<stdio.h>

int main()
{
    FILE *archivo;
    archivo = fopen("archivo.txt", "r");

    if (archivo != NULL)
    {
        printf("El archivo se abrió correctamente.\n");
        int res = fclose(archivo);
        printf("fclose = %d\n", res);
    }
    else
    {
        printf("Error al abrir el archivo.\n");
        printf("El archivo no existe o no se tienen permisos de lectura.\n");
    }

    return 0;
}
```

Funciones fgets y fputs

Las funciones *fgets()* y *fputs()* pueden leer y escribir, respectivamente, cadenas sobre los archivos. Las firmas de estas funciones son, respectivamente:

```
char *fgets(char *buffer, int tamaño, FILE *apArch);
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 181/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

```
char *fputs(char *buffer, FILE *apArch);
```

La función *fputs()* permite escribir una cadena en un archivo específico. La función *fgets()* permite leer una cadena desde el archivo especificado. Esta función lee un renglón a la vez.

Código (*fgets*)

Este programa permite leer el contenido de un archivo, de ser posible ya que debe existir, a través de la función *fgets()*.

Programa2.c

```
#include<stdio.h>

int main()
{
    FILE *archivo;
    char caracteres[50];
    archivo = fopen("gets.txt", "r");

    if (archivo != NULL)
    {
        printf("El archivo se abrió correctamente.");
        printf("\nContenido del archivo:\n");
        while (feof(archivo) == 0)
        {
            fgets (caracteres, 50, archivo);
            printf("%s", caracteres);
        }
        fclose(archivo);
    }

    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	182/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Código (fputs)

Este programa permite escribir una cadena dentro de un archivo, de ser posible ya que debe existir, a través de la función *fputs*.

Programa3.c

```
#include<stdio.h>

int main()
{
    FILE *archivo;
    char escribir[]="Escribir cadena en archivo mediante fputs.\n\tFacultad de
Ingeniería.\n";
    archivo = fopen("puts.txt", "r+");

    if (archivo != NULL)
    {
        printf("El archivo se abrió correctamente.\n");
        fputs (escribir, archivo);
        fclose(archivo);
    }
    else
    {
        printf("Error al abrir el archivo.\n");
        printf("El archivo no existe o no se tienen permisos de lectura.\n");
    }

    return 0;
}
```

Funciones fscanf y fprintf

Las funciones *fprintf()* y *fscanf()* se comportan similarmente a *printf()* (imprimir) y *scanf()* (leer), con la diferencia de que operan sobre un archivo. Sus estructuras son:

```
int fprintf(FILE *apArch, char *formato, ...);
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 183/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```
int fscanf(FILE *apArch, char *formato, ...);
```

Donde *apArch* es un apuntador al archivo devuelto por una llamada a la función *fopen()*, es decir, *fprintf()* y *fscanf()* dirigen sus operaciones de E/S al archivo al que apunta *apArch*. *formato* es una cadena que puede incluir texto o especificadores de impresión de variables. En los puntos suspensivos se agregan las variables (si es que existen) cuyos valores se quieren escribir en el archivo.

Código (fscanf)

Este programa permite leer el contenido de un archivo, de ser posible ya que debe existir, a través de la función *fscanf*.

Programa4.c

```
#include<stdio.h>

int main()
{
    FILE *archivo;
    char caracteres[50];
    archivo = fopen("fscanf.txt", "r");
    if (archivo != NULL)
    {
        while (feof(archivo)==0)
        {
            fscanf(archivo, "%s", caracteres);
            printf("%s\n", caracteres);
        }
        fclose(archivo);
    }
    else
    {
        printf("El archivo no existe.\n");
    }
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-17 05 184/187 8.3 22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

Código (fprintf)

Este programa permite escribir dentro de un archivo, de ser posible, a través de la función *fprintf*.

Programa5.c

```
#include<stdio.h>

int main()
{
    FILE *archivo;
    char escribir[] = "Escribir cadena en archivo mediante fprintf. \nFacultad de
Ingeniería.\n";
    archivo = fopen("fprintf.txt", "r+");
    if (archivo != NULL)
    {
        fprintf(archivo, escribir);
        fprintf(archivo, "%s", "UNAM\n");
        fclose(archivo);
    }
    else
    {
        printf("El archivo no existe o no se tiene permisos de lectura/escritura. \n");
    }
    return 0;
}
```

Funciones fread y fwrite

fread y *fwrite* son funciones que permiten trabajar con elementos de longitud conocida. *fread* permite leer uno o varios elementos de la misma longitud a partir de una dirección de memoria determinada (apuntador).

El valor de retorno es el número de elementos (bytes) leídos. Su sintaxis es la siguiente:

```
int fread(void *ap, size_t tam, size_t nelem, FILE *archivo)
```

fwrite permite escribir hacia un archivo uno o varios elementos de la misma longitud almacenados a partir de una dirección de memoria determinada.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	185/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

El valor de retorno es el número de elementos escritos. Su sintaxis es la siguiente:

```
int fwrite(void *ap, size_t tam, size_t nelem, FILE *archivo)
```

Código (fread)

Este programa muestra el contenido de un archivo de texto. El nombre del archivo se recibe como argumento de la función principal.

Programa6.c

```
#include <stdio.h>

int main(int argc, char **argv)
{
    FILE *ap;
    unsigned char buffer[2048]; // Buffer de 2 Kbytes
    int bytesLeidos;

    // Si no se ejecuta el programa correctamente
    if(argc < 2)
    {
        printf("Ejecutar el programa de la siguiente manera:
                \n\tnombre_\tprograma nombre_archivo\n");
        return 1;
    }

    // Se abre el archivo de entrada en modo lectura y binario
    ap = fopen(argv[1], "rb");

    if(!ap)
    {
        printf("El archivo %s no existe o no se puede abrir", argv[1]);
        return 1;
    }

    while(bytesLeidos = fread(buffer, 1, 2048, ap))
    {
        printf("%s", buffer);
    }
    fclose(ap);
    return 0;
}
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	05
		Página	186/187
		Sección ISO	8.3
		Fecha de emisión	22 / enero / 2025
Facultad de Ingeniería		Área/Departamento:	Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada	

Código (fwrite)

Este programa realiza una copia exacta de dos archivos. Los nombres de los archivos (origen y destino) se reciben como argumentos de la función principal.

Programa7.c

```
#include <stdio.h>

int main(int argc, char **argv)
{
    FILE *archEntrada, *archivoSalida;
    unsigned char buffer[2048]; // Buffer de 2 Kbytes
    int bytesLeidos;

    // Si no se ejecuta el programa correctamente
    if(argc < 3)
    {
        printf("Ejectuar el programa de la siguiente manera:\n");
        printf("\tnombre_programa \tarchivo_origen \tarchivo_destino\n");
        return 1;
    }

    // Se abre el archivo de entrada en modo de lectura y binario
    archEntrada = fopen(argv[1], "rb");

    if(!archEntrada)
    {
        printf("El archivo %s no existe o no se puede abrir", argv[1]);
        return 1;
    }

    // Se crea o sobreescribe el archivo de salida en modo binario
    archivoSalida = fopen(argv[2], "wb");
    if(!archivoSalida) {
        printf("El archivo %s no puede ser creado", argv[2]);
        return 1;
    }

    // Copia archivos
    while (bytesLeidos = fread(buffer, 1, 2048, archEntrada))
```

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código: MADO-17 Versión: 05 Página 187/187 Sección ISO 8.3 Fecha de emisión 22 / enero / 2025
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```

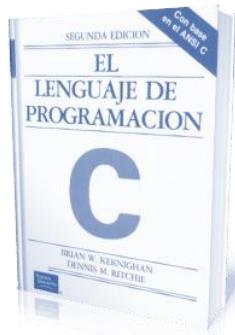
fwrite(buffer, 1, bytesLeidos, archivoSalida);

// Cerrar archivos
fclose(archEntrada);
fclose(archivoSalida);

return 0;
}

```

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Edu