

| | | | |
|---|---|---------------------|---------|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| Versión: | | 01 | |
| Página | | 1/207 | |
| Sección ISO | | 8.3 | |
| Fecha de emisión | | 20 de enero de 2017 | |
| 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: |
|-----------------|---------------------------|-----------------------------|---------------------|
| Jorge A. Solano | Laura Sandoval Montaño | Alejandro Velázquez Mena | 20 de enero de 2017 |

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 2/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 |
|----|--|
| 1 | La computación como herramienta de trabajo del profesional de ingeniería |
| 2 | GNU/Linux |
| 3 | Solución de problemas y Algoritmos. |
| 4 | Diagramas de flujo |
| 5 | Pseudocódigo |
| 6 | Entorno de C (editores, compilación y ejecución) |
| 7 | Fundamentos de Lenguaje C |
| 8 | Estructuras de selección |
| 9 | Estructuras de repetición |
| 10 | Depuración de programas |
| 11 | Arreglos unidimensionales y multidimensionales |
| 12 | Funciones |
| 13 | Lectura y escritura de datos |

| | | | |
|---|---|--|---------------------|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 3/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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:

Ing. Jorge A. Solano Gálvez
 M.C. Edgar E. García Cano
 M.I. Tanya Itzel Arteaga Ricci
 Ing. Laura Sandoval Montaño
 Carlos Rodrigo Sanabria del Campo

Revisado por:

M.C. Martha Angélica Nakayama Cervantes

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 4/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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:

Descubrir y utilizar herramientas de software que se ofrecen en Internet que 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 y buscadores con funciones avanzadas.

Actividades:

- Crear un repositorio de almacenamiento en línea.
- Realizar búsquedas avanzadas de información especializada.

Introducción

El uso de un equipo de cómputo se vuelve fundamental para el desarrollo de muchas de las actividades y tareas cotidianas que se realizan día con día, no importando el giro al creando nuevas y versátiles soluciones que apoyen y beneficien directamente a la sociedad al realizar dichas actividades; es por ello, que comprender cómo funciona y cómo poder mejorar dicho funcionamiento se vuelve un tema importante durante la formación del profesionista en ingeniería.

Es por lo anterior, que en el desarrollo de proyectos se realizan varias actividades donde la computación es un elemento muy útil. De las actividades que se realizan en la elaboración de proyectos o trabajos podemos mencionar:

- Registro de planes, programas y cualquier documento con información del proyecto en su desarrollo y en producción.
- Almacenamiento de la información en repositorios que sean accesibles, seguros y que la disponibilidad de la información sea las 24 hrs de los 360 días del año.
- Búsqueda avanzada o especializada de información en Internet.

En la presente práctica se presentarán las herramientas de apoyo a la realización de dichas actividades.

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 5/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| 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 el cual 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

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

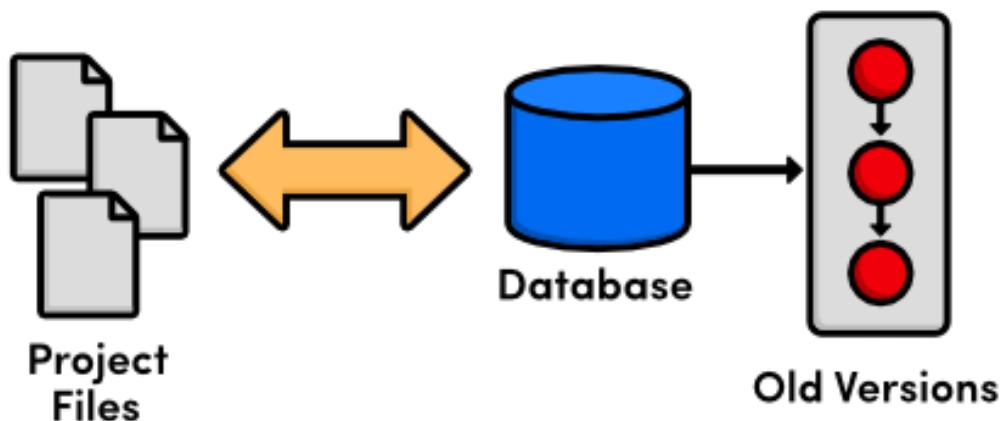


Figura 1: Control de Versiones Local

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 6/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| 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 con colaboradores, 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.

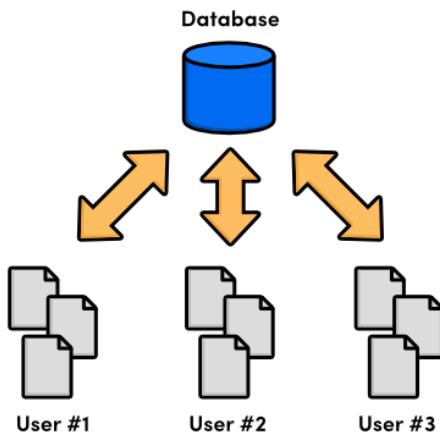


Figura 2: Control de Versiones Centralizado

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, además los usuarios pueden obtener los cambios en los archivos directamente del equipo de otros usuarios.

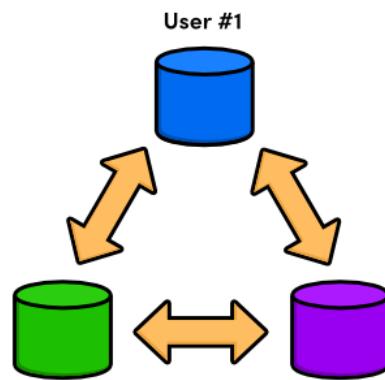


Figura 3: Control de Versiones Distribuido

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 7/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Git

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

Repositorio

Un 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 los archivos necesarios para llevar a cabo el control de versiones.

Repositorio Local

Un repositorio local, es aquel que se encuentra en nuestro propio equipo y solo el dueño del equipo tiene acceso a él.

Repositorio Remoto

Un 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 estas plataformas son: github.com, bitbucket.org o gitlab.com, todos ofreciendo diferentes características.

Github

Github es una plataforma de almacenamiento para control de versiones y colaboración. Esta plataforma nos 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 14 millones de usuarios haciéndola la plataforma más grande de almacenamiento de código en el mundo.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 8/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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.

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, SkyDrive, iCloud o Dropbox son algunos espacios de almacenamiento en la nube. Además, Google Drive (Google) y SkyDrive (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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 9/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |



<http://www.youtube.com/watch?v=wKJ9KzGQq0w>



<http://www.youtube.com/watch?v=hoTBiIpz8DI>

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 puedan 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.

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 10/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |



Google Forms

Google Drive cuenta con una aplicación para recolectar información usando formularios (Forms), una particularidad de la hoja de cálculo.

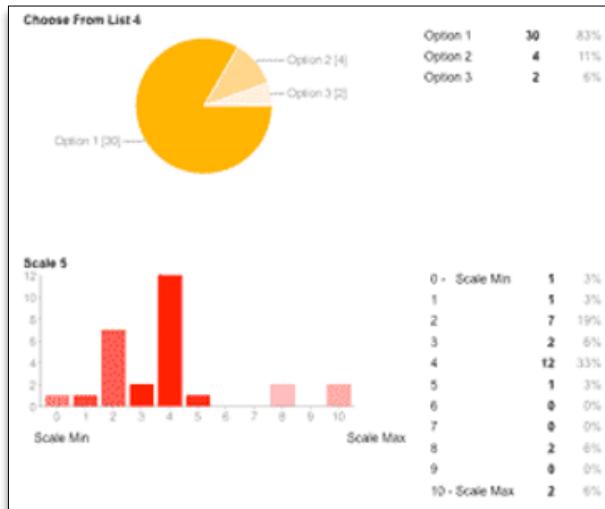
New forms features

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"/> |

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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 11/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |



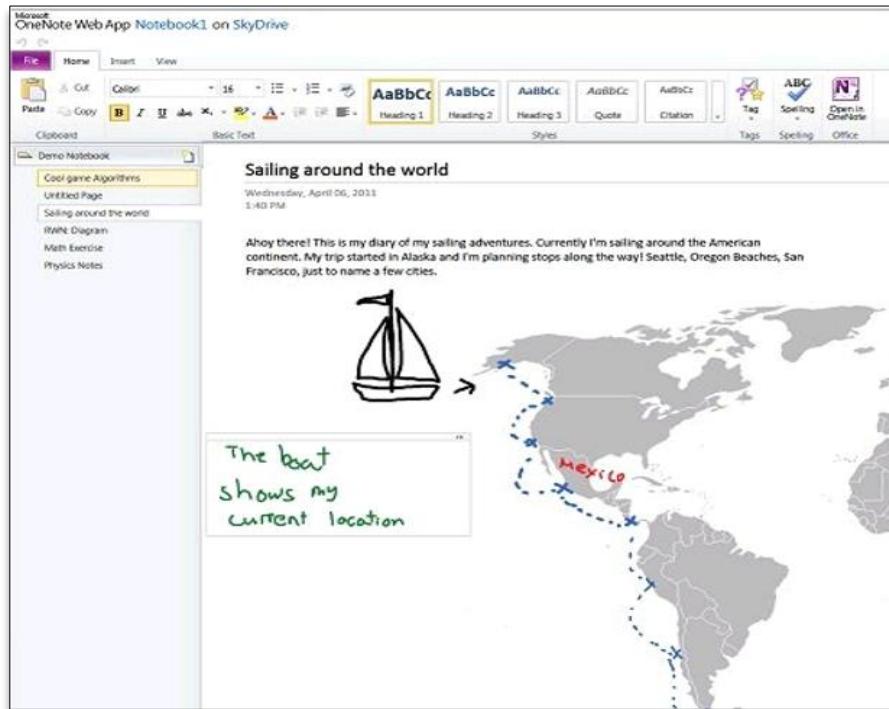
<http://www.youtube.com/watch?v=IzgaUOW6GIs>

OneNote

Por otro lado, a través de SkyDrive 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: 01 Página 12/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |



<http://www.youtube.com/watch?v=nxi9c6xBb0U>

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).

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 13/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |



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 link.

<https://www.dropbox.com/>

Buscadores de Internet

Los motores de búsqueda (también conocidos como buscadores) son aplicaciones informáticas que rastrean la red de redes (Internet) catalogando, clasificando y organizando información, para poder mostrarla en el navegador.

El rastreo de información se realiza a través de algoritmos propios de cada buscador, por ejemplo:

- Yahoo utiliza WebRank, a partir de una escala del 1 al 10, mide la popularidad de una página web.
- Live Search utiliza un algoritmo que analiza diversos factores, como son el contenido de una página, el número y calidad de los sitios web que han enlazado la página, así como las palabras clave contenidas en el sitio.

| | | | |
|---|---|--|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: Versión: Página: Sección ISO Fecha de emisión | MADO-17 01 14/207 8.3 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

- Google utilizar el llamado PageRank, que es un valor numérico que representa la popularidad que una página web tiene en Internet. PageRank es un concepto (marca registrada y patentada) de Google que introduce en su algoritmo de indexación.

Buscador de Internet Google

El buscador de Google (en inglés Google Search) es un motor de búsqueda en la web propiedad de Google Inc. Es el motor de búsqueda más utilizado en la Web. Fue desarrollado por Larry Page y Sergey Brin en 1997.



Características

1. Para encontrar todas las imágenes de natación o de futbol que no contengan la palabra tenis se utiliza la siguiente búsqueda:



Manual de prácticas del Laboratorio de Fundamentos de programación

| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 15/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Imagenes natacion **or** futbol **-tenis**

-, indica que la búsqueda no debe contener esa palabra.

or: indica que la búsqueda debe contener una palabra o la otra.

Nota: no es necesario agregar acentos en la búsqueda.

2. Para encontrar todos los datos pertenecientes sólo a la **jornada del futbol mexicano**:

"jornada del futbol mexicano"

Las comillas dobles ("<oración>") al inicio y al final de la búsqueda indican que sólo se deben buscar páginas que contengan exactamente dichas palabras. En este caso se agregó el conector *del* a la búsqueda para encontrar exactamente la frase.

3. Al momento de hacer búsquedas no es necesario incluir palabras como los artículos (el, la, los, las, un, etc.), pero en caso de ser necesario se puede hacer lo siguiente:

+la jornada

El símbolo de + sirve para que en la búsqueda se agregue la palabra y encuentre páginas que la incluyan.



Manual de prácticas del Laboratorio de Fundamentos de programación

| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 16/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Comandos

Si se quiere saber el significado de una palabra, simplemente hay que agregar **define:<palabra>**.

```
| define:computacion
```

site ayuda a buscar sólo en un sitio determinado.

```
| site:cnnmexico.com ~olimpiadas 2012..2013
```

~ indica que encuentre cosas relacionadas con una palabra.

.. sirve para buscar en un intervalo de números, en este caso de años.

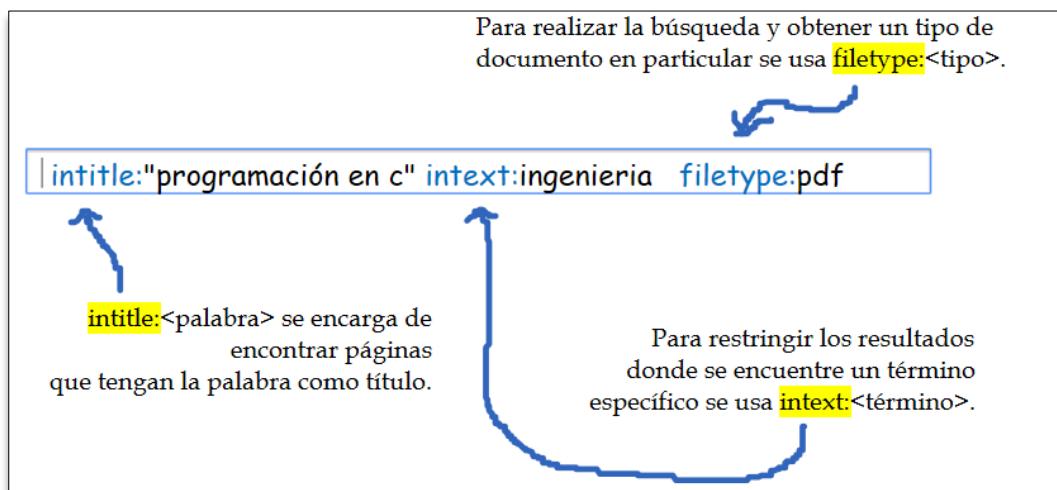
| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 17/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

Para realizar la búsqueda y obtener un tipo de documento en particular se usa **filetype:<tipo>**.

| intitle:"programación en c" intext:ingenieria filetype:pdf

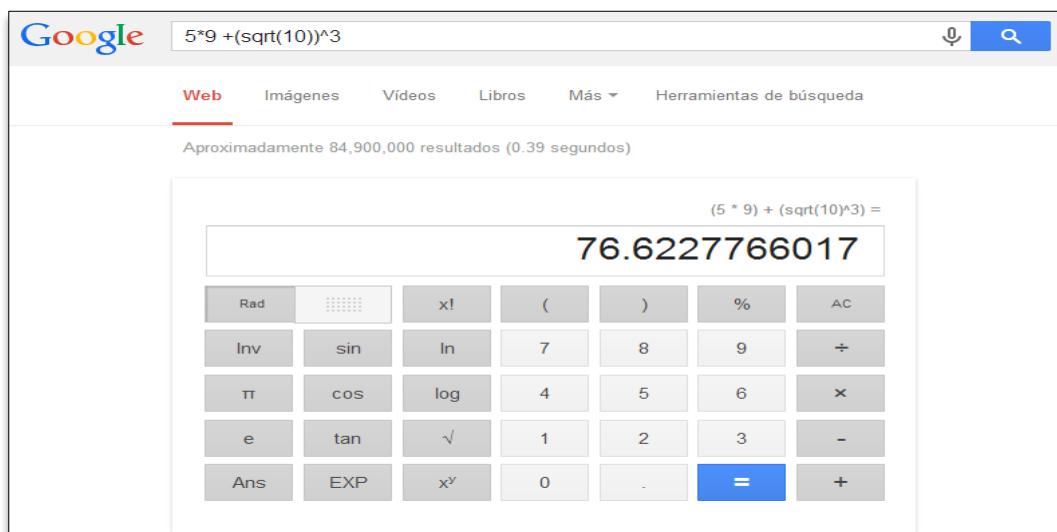
intitle:<palabra> se encarga de encontrar páginas que tengan la palabra como título.

Para restringir los resultados donde se encuentre un término específico se usa **intext:<término>**.



Calculadora

Google permite realizar diversas operaciones dentro de la barra de búsqueda simplemente agregando la ecuación en dicho campo.



Google

5*9 +(sqrt(10))^3

Web Imágenes Vídeos Libros Más Herramientas de búsqueda

Aproximadamente 84,900,000 resultados (0.39 segundos)

(5 * 9) + (sqrt(10)^3) =

76.6227766017

| | | | | | | |
|-----|-----|----------------|---|---|---|----|
| Rad | | x! | (|) | % | AC |
| Inv | sin | ln | 7 | 8 | 9 | ÷ |
| π | cos | log | 4 | 5 | 6 | × |
| e | tan | √ | 1 | 2 | 3 | - |
| Ans | EXP | x ^y | 0 | . | = | + |

| | | | | |
|---|---|--|---------------------|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 | |
| | | Versión: | 01 | |
| | | Página | 18/207 | |
| | | Sección ISO | 8.3 | |
| | | Fecha de emisión | 20 de enero de 2017 | |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | | |



Google search results for $\sin(1) + \cos(0)$. The search bar shows "sin(1) + cos(0)". Below the search bar, there are tabs for Web, Imágenes, Vídeos, Noticias, Más, and Herramientas de búsqueda. It says "Aproximadamente 37,300,000 resultados (0.33 segundos)". A suggestion says "Sugerencia: Buscar solo resultados en español". Below the search bar, a digital calculator interface shows the input $\sin(1 \text{ radian}) + \cos(0 \text{ radians}) =$ and the result **1.84147098481**. The calculator has a standard layout with buttons for Rad, Inv, π, e, Ans, sin, cos, tan, EXP, x!, ln, log, √, x^y, (,), ., %, ÷, ×, -, +, and =.

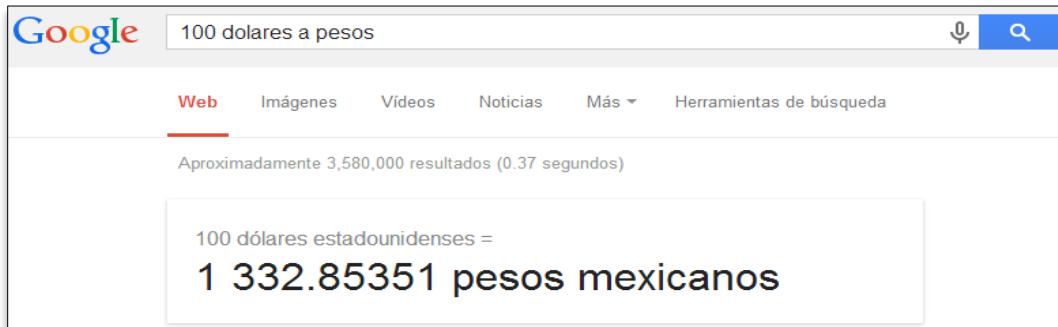
Convertidor de unidades

El buscador de Google también se puede utilizar para obtener la equivalencia entre dos sistemas de unidades.



Google search results for "90 grados centigrados a fahrenheit". The search bar shows "90 grados centigrados a fahrenheit". Below the search bar, there are tabs for Web, Imágenes, Vídeos, Noticias, Más, and Herramientas de búsqueda. It says "Aproximadamente 112,000 resultados (0.46 segundos)". The search results show the conversion: "90 grados centigrados = **194 grados Fahrenheit**".

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 19/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |



Google 100 dolares a pesos

Web Imágenes Vídeos Noticias Más Herramientas de búsqueda

Aproximadamente 3,580,000 resultados (0.37 segundos)

100 dólares estadounidenses =
1 332.85351 pesos mexicanos

Nota: el navegador interpreta la moneda nacional, si se requiere la conversión a otra moneda solo se especifica el tipo de peso (colombianos, argentinos, chilenos, etc.).

Graficas en 2D

Es posible graficar funciones, para ello simplemente se debe insertar ésta en la barra de búsqueda. También se puede asignar el intervalo de la función que se desea graficar.

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página: 20/207 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |



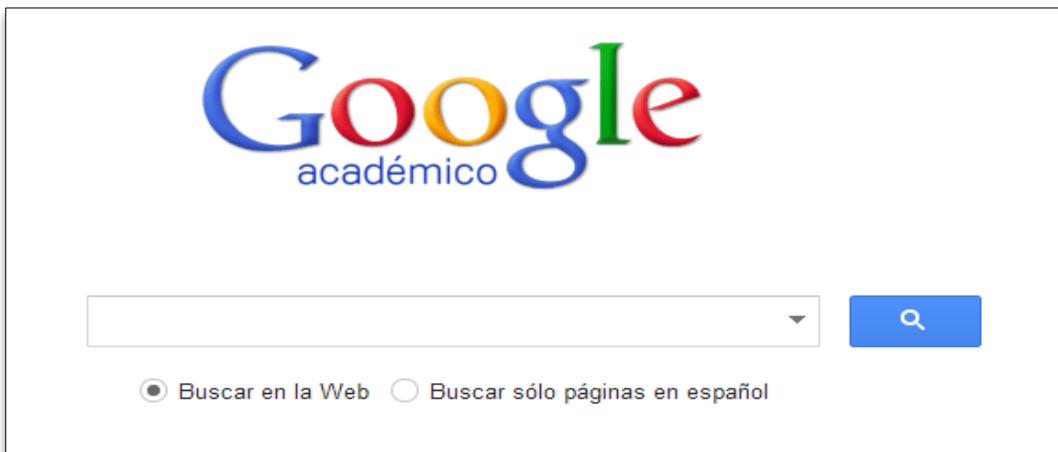
Google académico

Si se realiza la siguiente búsqueda define:"google scholar", se obtiene:

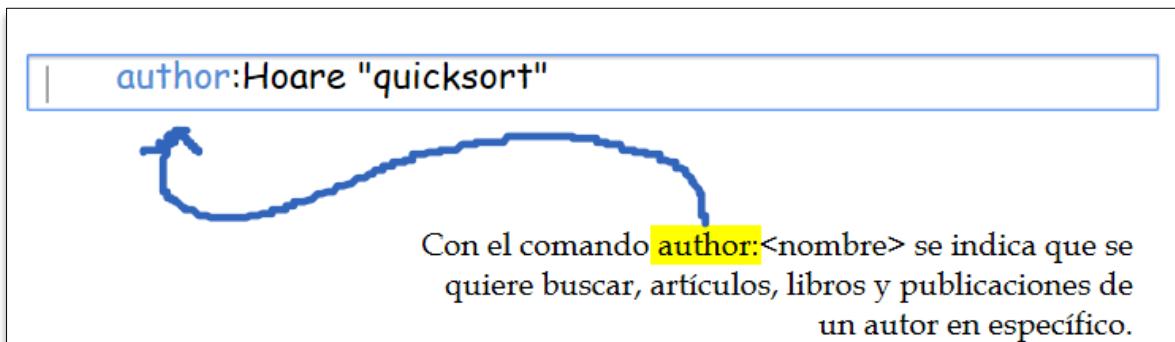
"Google Académico es un buscador de Google especializado en artículos de revistas científicas, enfocado en el mundo académico, y soportado por una base de datos disponible libremente en Internet que almacena un amplio conjunto de trabajos de investigación científica de distintas disciplinas y en distintos formatos de publicación."

<http://scholar.google.es/>

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 21/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |



La siguiente búsqueda encuentra referencias del algoritmo de ordenamiento Quicksort creado por Hoare:



Dentro de la página se pueden observar varias características de la búsqueda realizada:

| | | |
|---|---|--|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 22/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

Guardar artículos

Google Académico

Artículos Mi biblioteca Cualquier momento Desde 2014 Desde 2013 Desde 2010 Intervalo específico... Ordenar por relevancia Ordenar por fecha Buscar en la Web Buscar sólo páginas en español ✓ incluir patentes Rango de tiempo Crear alerta

author Hoare "quicksort"

Aproximadamente 15 resultados (0,04 s)

Sugerencia: Buscar solo resultados en **español**. Puedes especificar el idioma de búsqueda en [Configuración de Google Académico](#).

Quicksort
M.Foley, C.R. Hoare - The Computer Journal, 1971 - Br Computer Soc
Abstract: A description is given of a new method of sorting in a random-access store of a Computer. The method compares very favourably with other known methods in speed, in economy of storage, and in ease of programming. Certain refinements of the method, ... Citados por 985 Artículos relacionados Las 3 versiones Citar Guardar

Proof of a recursive program: Quicksort
M.Foley, C.R. Hoare - The Computer Journal, 1971 - Br Computer Soc
Abstract: This paper gives the proof of a useful and non-trivial program, Quicksort (Hoare, 1971). It is shown that the algorithm is correct and that it is optimal. It is also shown that it is a rigorous but informal proof of correctness. Formal methods are introduced. ... Citado por 60 Artículos relacionados Las 2 versiones Citar Guardar

Concurrent theory of parallel programming
C.R. Hoare - The origin of concurrent programming, 2002 - Springer
... consists of forty-nine books. It is fitting that the fifth book be a treasure-house of works by Hoare himself. Read it for a sense of history. Read it for the facts about Quicksort, proving programs correct, CSP, etc. Read it for its ... Citados por 420 Artículos relacionados Las 4 versiones Citar Guardar

Essays in computing science
Hoare, C.R. Jones - 1969 - dl.acm.org
... consisted of forty-nine books. It is fitting that the fifth book be a treasure-house of works by Hoare himself. Read it for a sense of history. Read it for the facts about Quicksort, proving programs correct, CSP, etc. Read it for its ... Citados por 116 Artículos relacionados Las 5 versiones Citar Guardar

Sitio en el que está publicado. [PDF] de oxfordjournals.org [PDF] de cmu.edu

Google imágenes

Permite realizar una búsqueda arrastrando una imagen almacenada en la computadora hacia el buscador de imágenes.

<http://www.google.com/imghp>

Buscar por imágenes

Arrastra la imagen aquí

Mover

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 23/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Google Koala.jpg x mi pc

Web Imágenes Videos Más Herramientas de búsqueda

Cerca de 1,970,000 resultados (0.50 segundos)

 Tamaño de la imagen:
1024 × 768

Buscar esta imagen en otros tamaños:
[Todos los tamaños](#) - [Grande](#)

Resultados de [mi pc](#)

[Mi PC Comunicaciones, Computadoras, Laptops, Impresoras y ...](#)
www.mipc.com.mx/
En MiPC.com.mx contamos con los precios más bajos en Computadoras, Laptops y Accesorios para venta en línea en México. Contamos con 15 sucursales en ...

[Imágenes similares](#) - Notificar imágenes













| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 24/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

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.

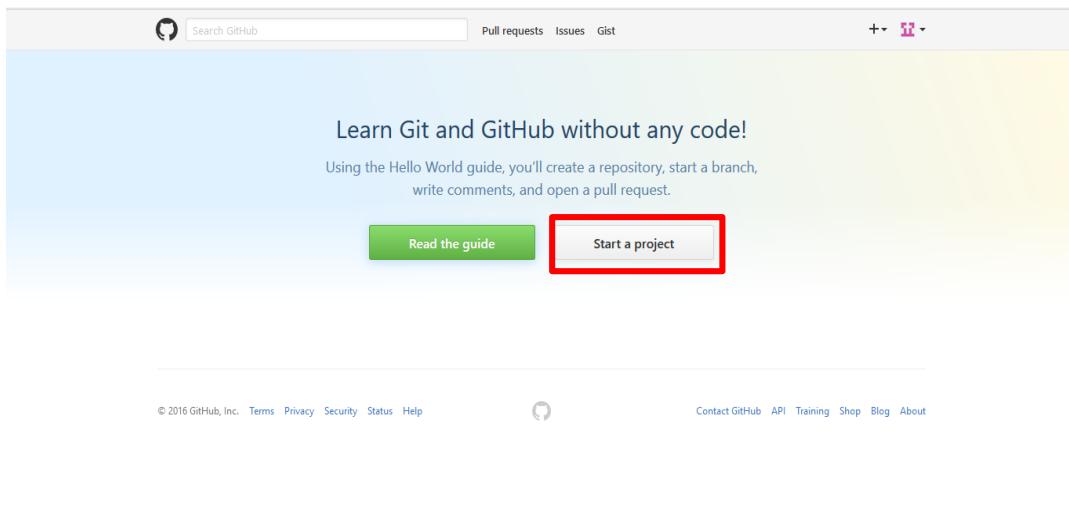


Escribimos un usuario propio, un correo, una contraseña y damos click en “Create an account”, elegimos el plan gratuito y damos en continuar. Damos click en “skip this step”, esperamos el correo de verificación, y verificamos nuestra cuenta.

Creando nuestro primer repositorio

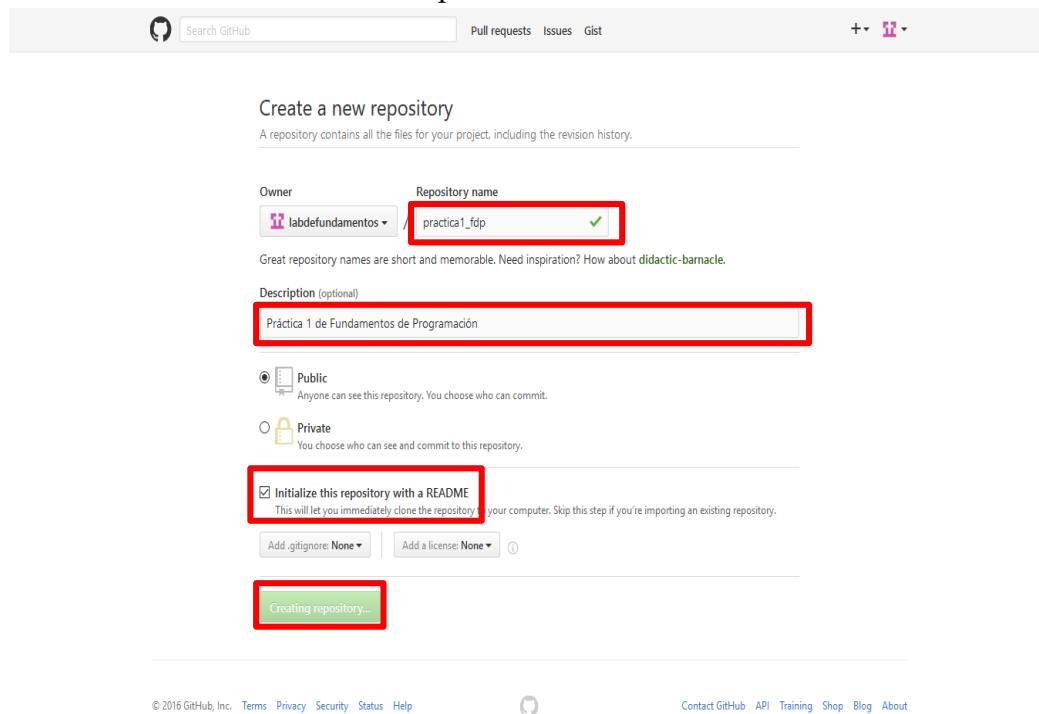
Damos click en el botón de “Start a Project”

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 25/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| 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 the GitHub homepage. At the top, there is a search bar, a pull requests button, an issues button, and a gist button. Below the search bar, the text "Learn Git and GitHub without any code!" is displayed, followed by a brief description: "Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request." There are two main buttons: "Read the guide" (green) and "Start a project" (white with blue text). The "Start a project" button is highlighted with a red box.

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”

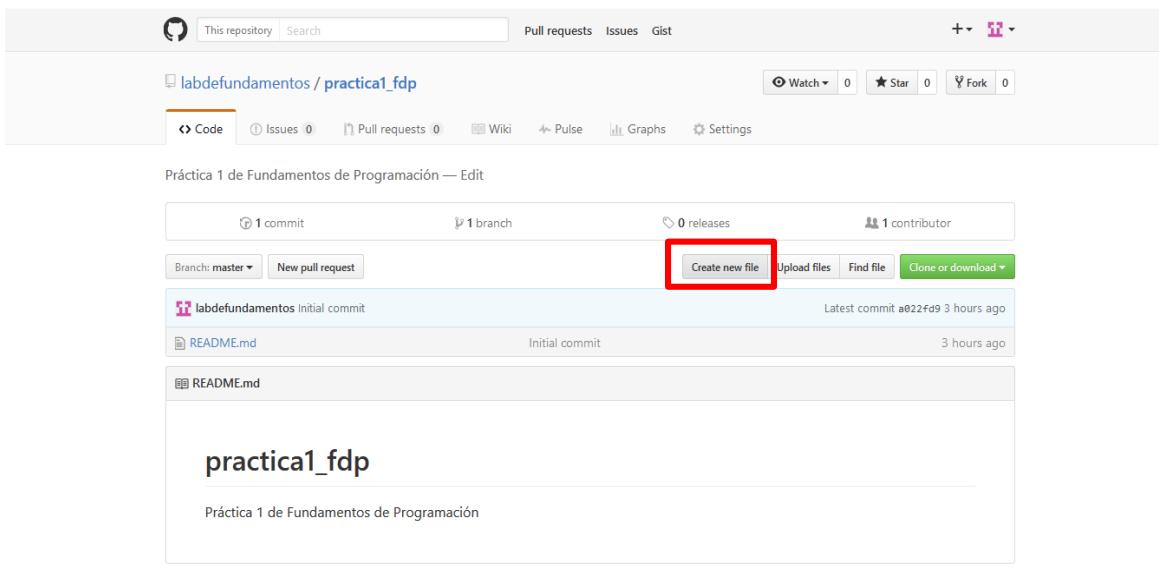


The screenshot shows the "Create a new repository" page on GitHub. The "Owner" dropdown is set to "labfundamentos". The "Repository name" field contains "practica1_fdp" and has a green checkmark icon. The "Description (optional)" field contains "Práctica 1 de Fundamentos de Programación" and is highlighted with a red box. Below these, there are two radio buttons: "Public" (selected) and "Private". Under "Public", it says "Anyone can see this repository. You choose who can commit." Under "Private", it says "You choose who can see and commit to this repository." A checkbox labeled "Initialize this repository with a README" is checked and highlighted with a red box. Below the checkbox, a note says "This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository." At the bottom, there are buttons for "Add .gitignore: None" and "Add a license: None". The "Creating repository..." button at the very bottom is also highlighted with a red box.

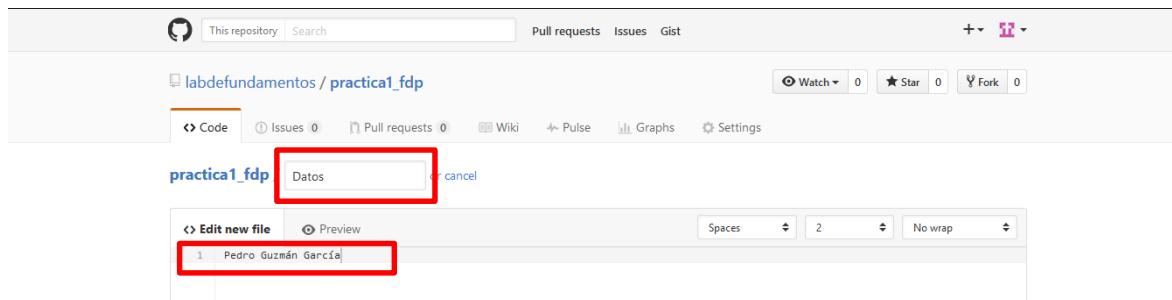
| | | |
|---|--|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 26/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

Creación de archivos en nuestro repositorio

Damos click en el botón de “Create new file”

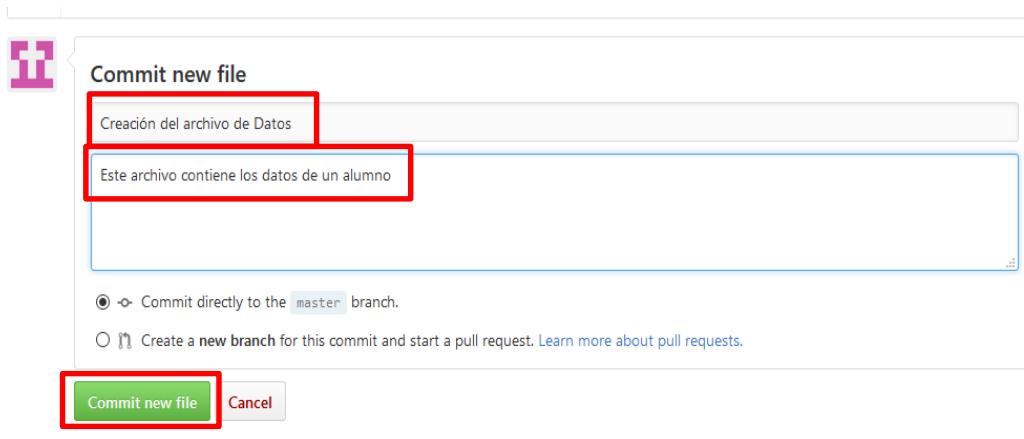


Crearemos un archivo llamado Datos, y en la primera línea agregaremos nuestro nombre.



En la sección de Commit new file, haremos una explicación del archivo creado, posteriormente damos click al botón de Commit new file.

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 27/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |



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.

This repository

Pull requests Issues Gist

labdefundamentos / **practica1_fdp**

Code Issues 0 Pull requests 0 Wiki Pulse Graphs Settings

Watch 0 Star 0 Fork 0

Práctica 1 de Fundamentos de Programación — Edit

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request

labdefundamentos committed on 8 minutes ago

Datos Creadión del archivo de Datos 8 minutes ago

README.md Initial commit 4 hours ago

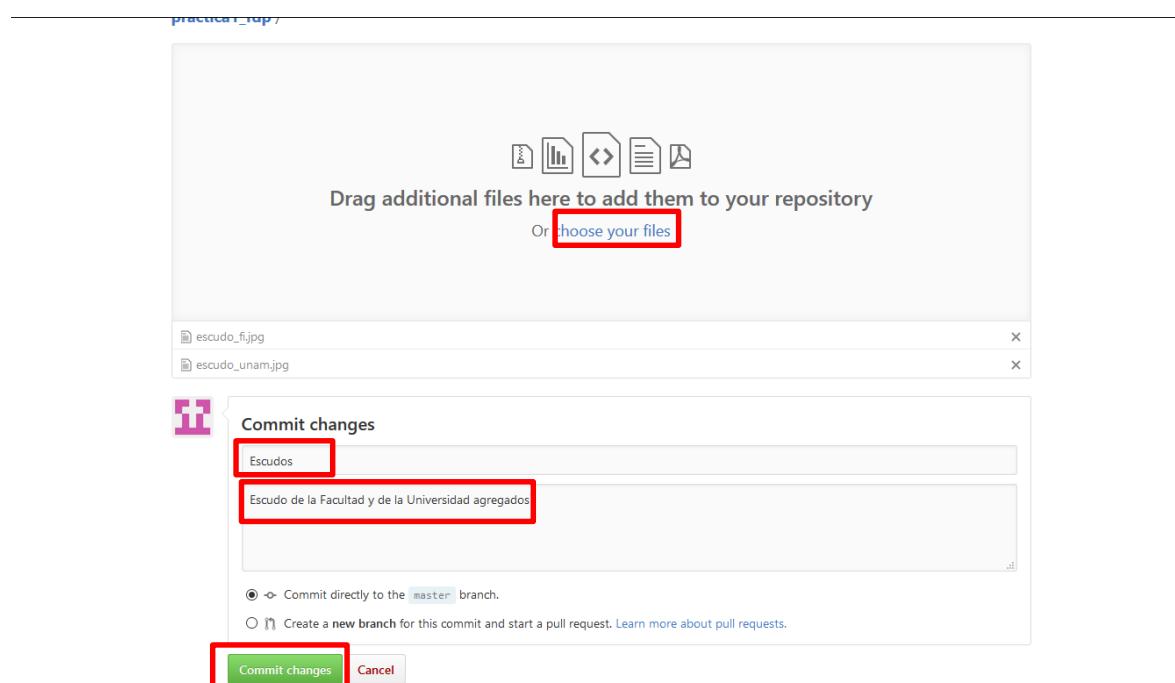
practica1_fdp

Práctica 1 de Fundamentos de Programación

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 28/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

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.



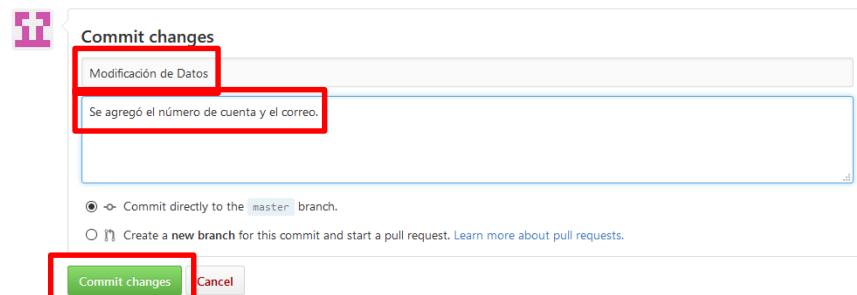
Como se observa, un commit puede ser de uno o más archivos.

Modificando un archivo

Damos click en el archivo “Datos” y posteriormente hacemos click en el botón 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.

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 29/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |



© 2016 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

Revisando la historia de nuestro repositorio

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

 4 commits

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 qué se hicieron en el commit. Git guarda cada estado de nuestros archivos, de esta manera siempre podemos acceder a versiones específicas.

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!

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 30/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Actividad Final

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:

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

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 31/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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.google.com.mx/>
4. <http://scholar.google.es/>
5. <http://www.google.com/imghp>
6. <http://www.youtube.com/watch?v=wKJ9KzGQq0w>
7. <http://www.youtube.com/watch?v=wKJ9KzGQq0w>
8. <http://www.youtube.com/watch?v=nxi9c6xBb0U>
9. <https://www.dropbox.com/>
10. http://bc.unam.mx/cultural/inicio/vis_virt/main.html
11. <http://www.inah.gob.mx/index.php/catalogo-paseos-virtuales>
12. <https://www.google.com/maps/views/home>
13. <https://maps.google.com/>
14. <http://translate.google.com/>
15. <http://www.google.com/earth/>
16. <http://news.google.com/>
17. <https://adwords.google.com/>
18. <http://books.google.com/>
19. <https://groups.google.com/>

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 32/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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



Elaborado por:

Ing. Jorge A. Solano Gálvez
M.C. Edgar E. García Cano

Actualizado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 33/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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:

Conocer la importancia del sistema operativo de una computadora, así como sus funciones. Explorar un sistema operativo GNU/Linux con el fin de conocer y utilizar los comandos básicos en GNU/Linux.

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 entre otros. 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: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 34/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 del mismo.

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: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 35/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 36/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

este archivo raíz, parten todos los demás. Los archivos pueden ser carpetas (directorios), de datos, aplicaciones, programas, etc.

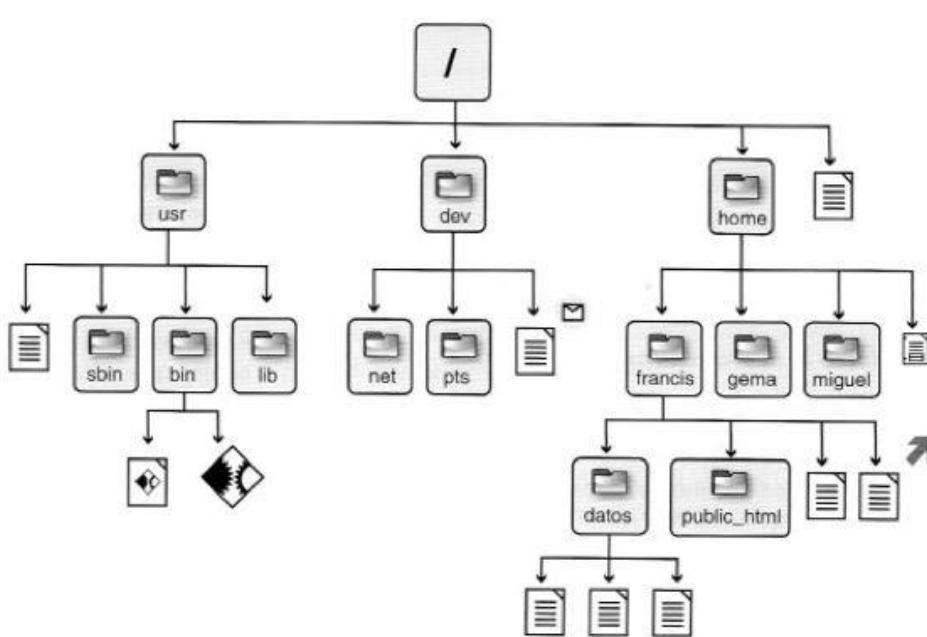


Figura 2: Una parte del sistema de archivos jerárquico en GNU/Linux.

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

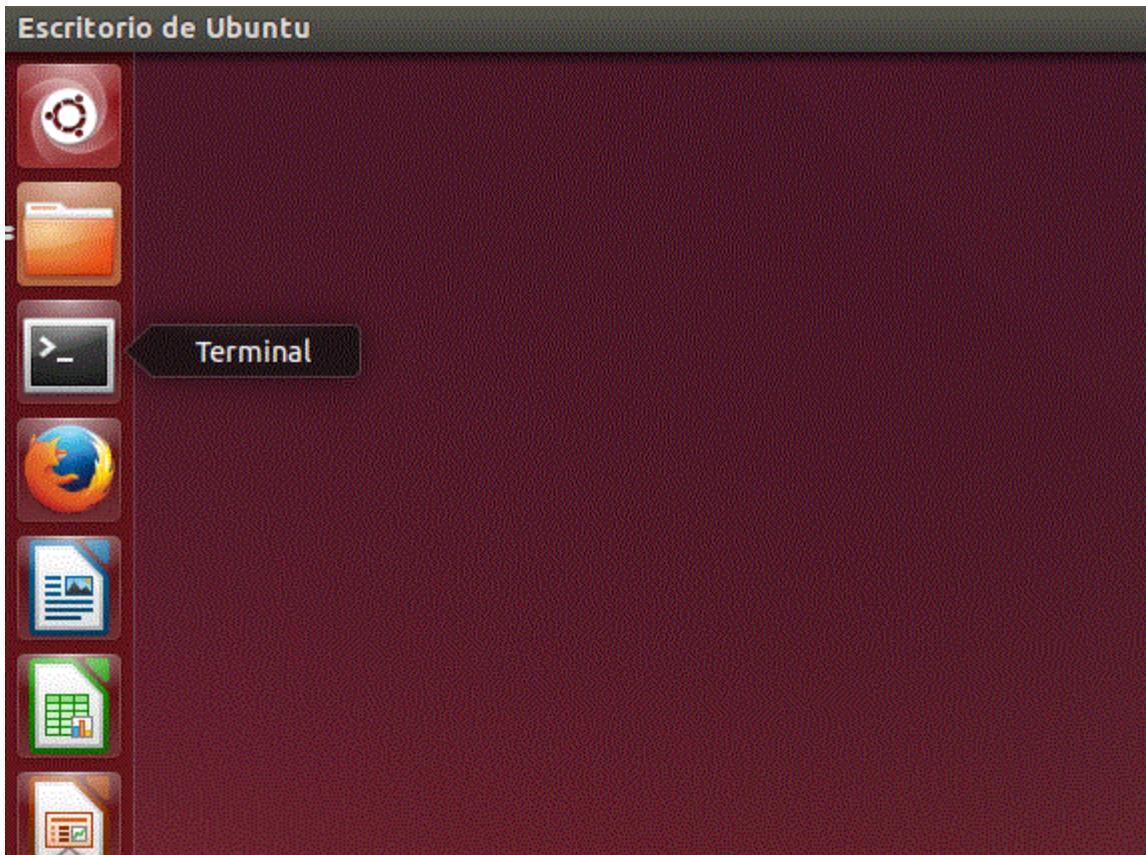
El Shell de GNU/Linux permite introducir órdenes (comandos) y ejecutar programas en el sistema operativo. Todas las órdenes de UNIX/Linux son programas que están almacenados en el sistema de archivos y a los que llamamos comandos, por lo tanto, todo en GNU/Linux se puede controlar mediante comandos.

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.

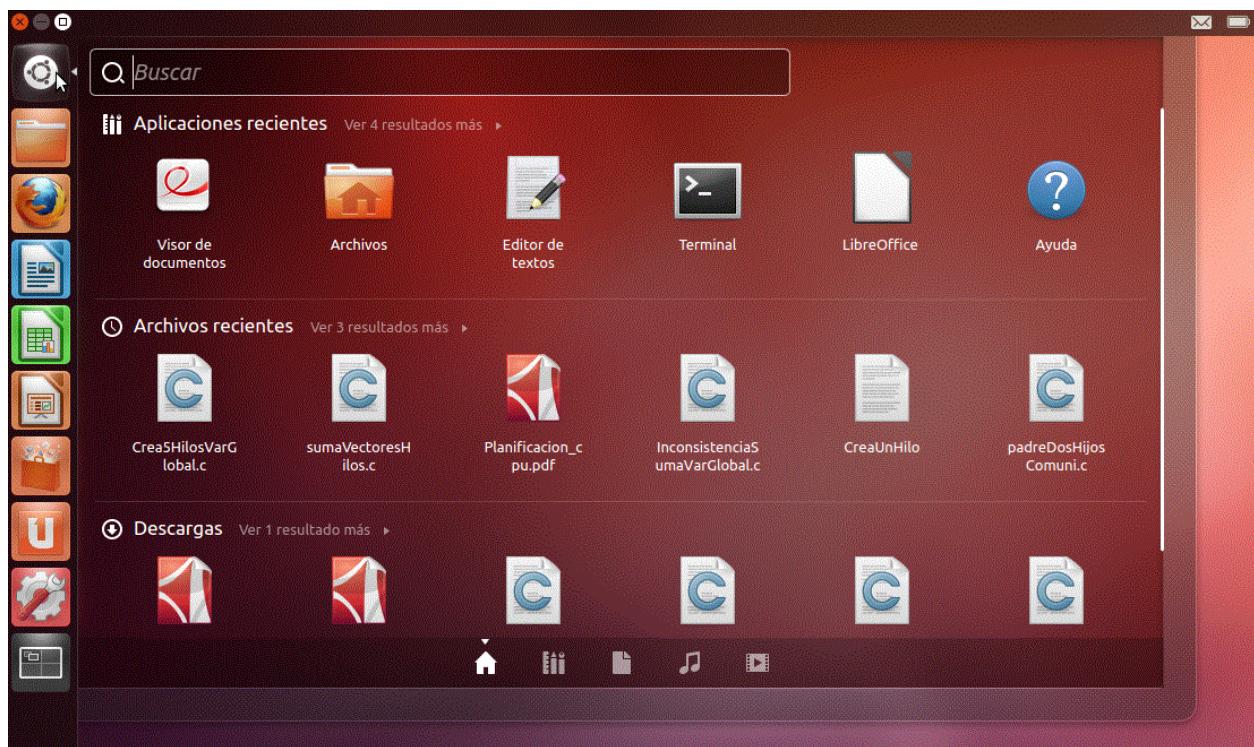
| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 37/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: 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*.



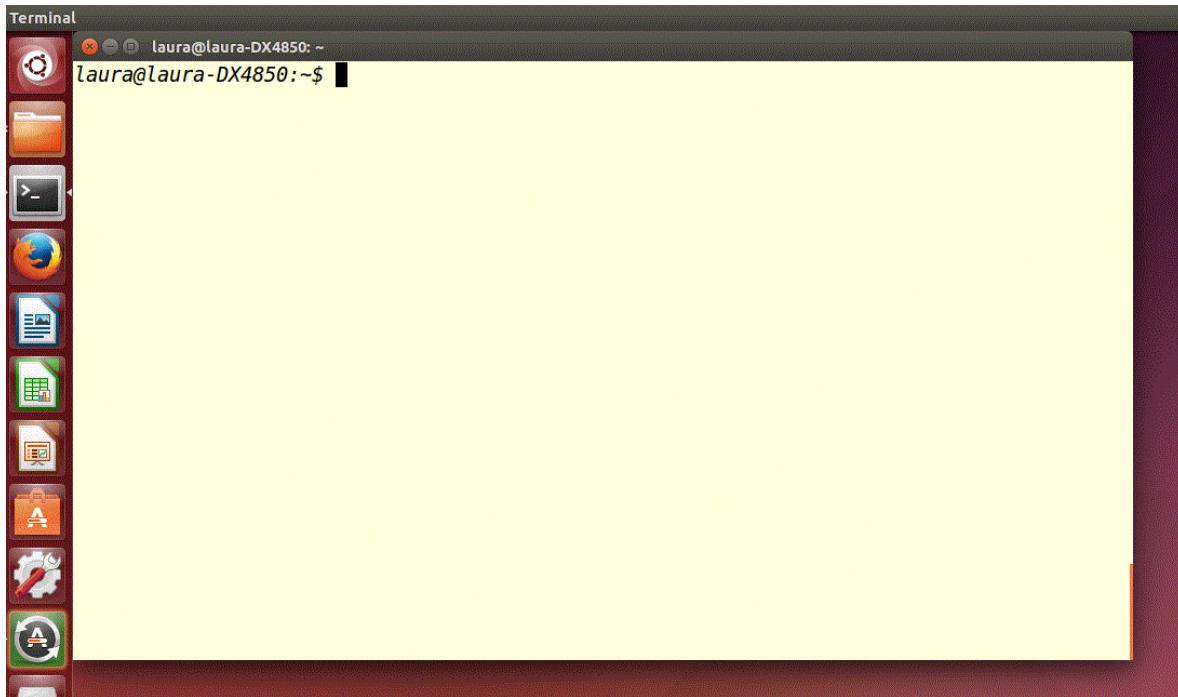
| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 38/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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.



Una vez teniendo una terminal abierta, estamos listos para introducir comandos.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 39/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |



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 del mismo 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

ls

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 40/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

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 a 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: | 01 |
| | | Página | 41/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 de 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
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 42/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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

```
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

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 43/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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:

```
find . -name tareas
```

Ejemplo (comando clear)

El comando *clear* permite limpiar la consola, 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 queremos 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:

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 44/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

```
mv ubicación_origen/archivo ubicación_destino
```

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

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_carpeta
```

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: | 01 |
| | | Página | 45/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 46/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Actualizado por:

Ing. Maricela Castañeda Perdomo
Ing. Laura Sandoval Montaño

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 47/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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:

Elaborar 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 a 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: MADO-17 Versión: 01 Página 48/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| 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.



Figura 1: Ciclo de vida del software.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 49/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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.

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.



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 01 50/207 8.3 20 de enero de 2017 |
| 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.

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

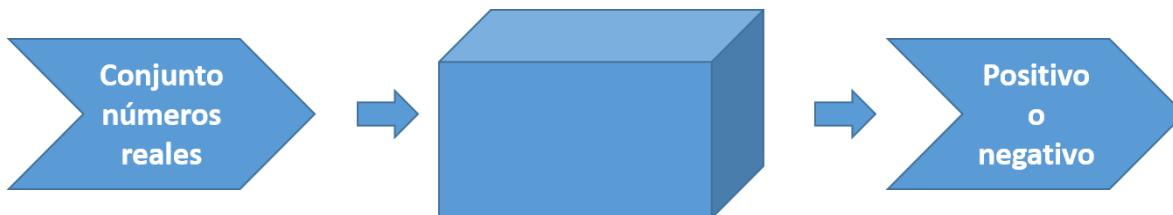
NOTA: \mathbb{R}^1 representa al conjunto de números reales de una dimensión.

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.



| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página: 51/207 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017 |
| 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.

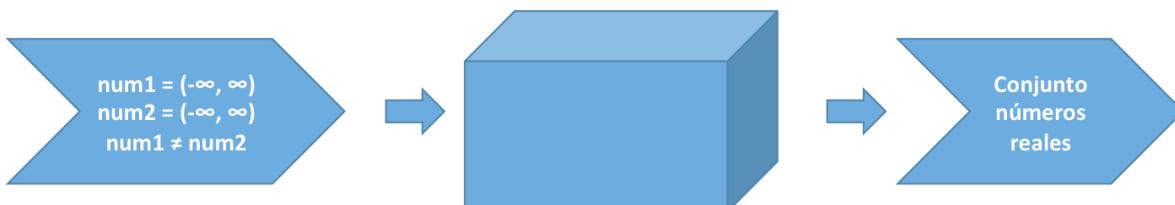
RESTRICIONES: Los números de entrada deben ser diferentes.

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 ($E' = (-\infty, \infty) \setminus \{num1\}$).

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

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.

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



| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 52/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| 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:

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

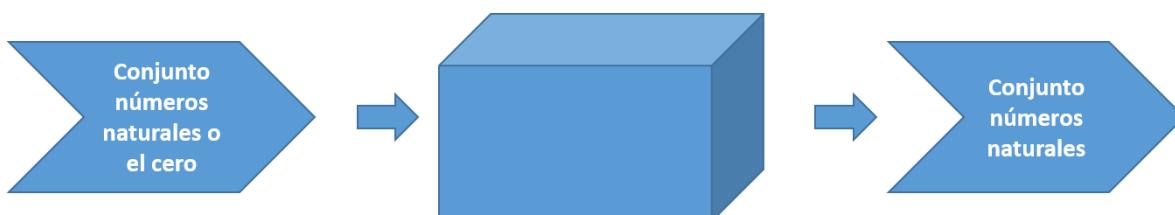
RESTRICIONES: El número de entrada debe ser entero positivo o cero. No puede ser negativo.

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

$$E \subset N^1, \text{ donde} \\ \text{num} \in E \text{ de } [1, \infty] \cup \{0\}$$

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

$$S \subset N^1; \text{ donde} \\ \text{res} \in S \text{ de } [1, \infty)$$



| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 53/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 esta debe de ser perceptible
- Debe ser sencillo y legible
- Eficiente: Realizarlo en el menor tiempo posible
- Eficaz: Que produzca el efecto esperado

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 54/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

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 2:

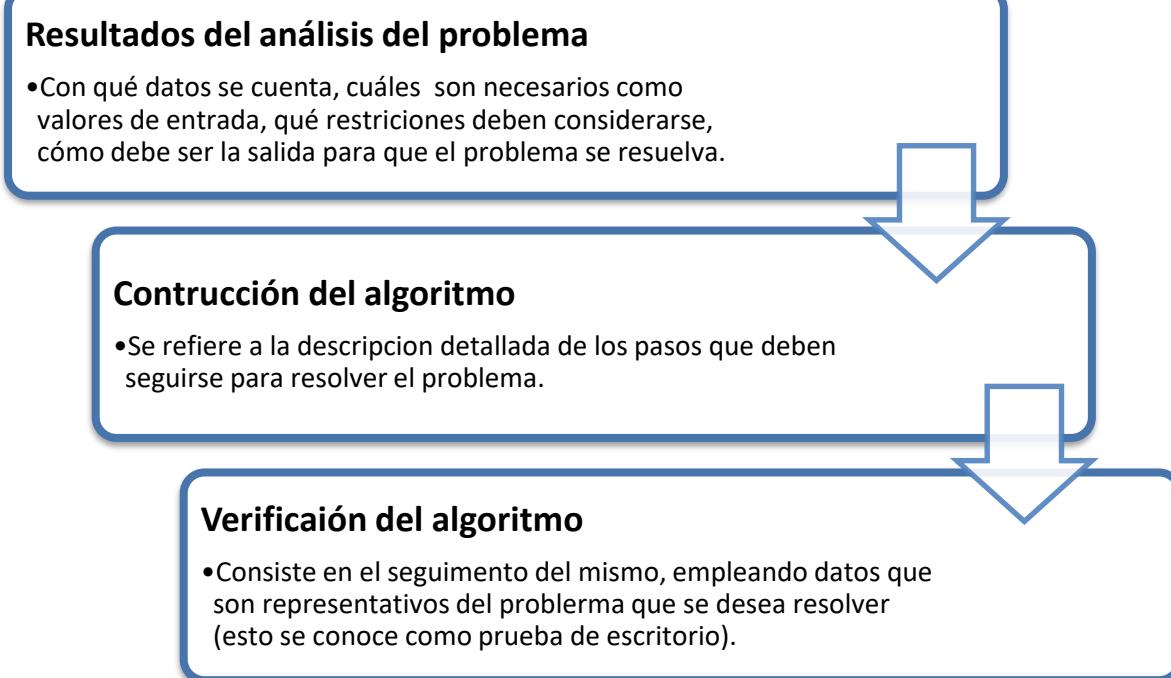
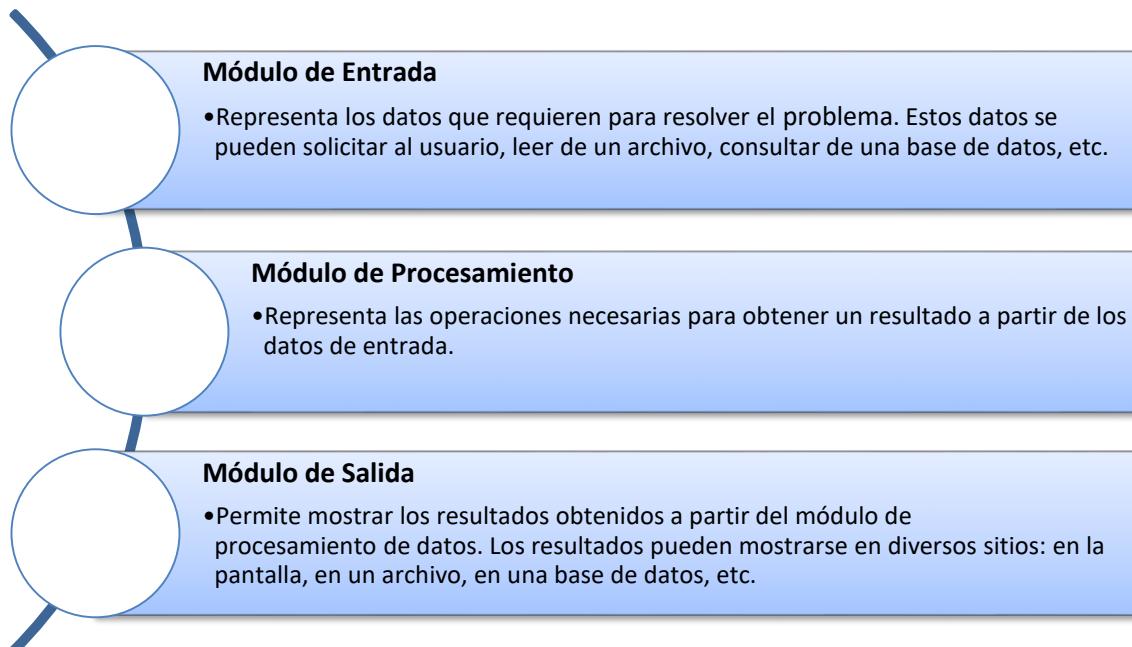


Figura 2: Elaboración de un algoritmo

| | | | |
|---|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: Versión: Página: Sección ISO Fecha de emisión | MADO-17 01 55/207 8.3 20 de enero de 2017 |
| Facultad de Ingeniería | 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:



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 validación de si el número es positivo

DOMINIO: Todos los número reales.

SOLUCIÓN:

1. Solicitar un número real.
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.

| | | | | |
|---|---|--|---------------------|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 | |
| | | Versión: | 01 | |
| | | Página | 56/207 | |
| | | Sección ISO | 8.3 | |
| | | Fecha de emisión | 20 de enero de 2017 | |
| 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

El diseño de la solución de un problema implica la creación del algoritmo y la validación del mismo. 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 programa en cada iteración. Una iteración es el número de veces que se ejecuta un código 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 el número solicitado):

| Iteración | X | Salida |
|-----------|---|-------------------------|
| 1 | 5 | El número 5 es positivo |

| Iteración | X | Salida |
|-----------|-----|--------------------------|
| 1 | -29 | El número 29 es negativo |

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

Ejemplo 2

PROBLEMA: Obtener el mayor de dos números dados.

RESTRICIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: Número real.

DATOS DE SALIDA: La impresión del número más grande.

DOMINIO: Todos los número reales.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 57/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 primer número real.
2. Solicitar un segundo número real.
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 (X es el primer número solicitado, Y es el segundo):

| Iteración | X | Y | Salida |
|-----------|---|---|-------------------------|
| 1 | 5 | 6 | El número 6 es el mayor |

| Iteración | X | Y | Salida |
|-----------|-----|--------|---------------------------|
| 1 | -99 | -222.2 | El número -99 es el mayor |

| Iteración | X | Y | Salida |
|-----------|----|----|--------------------------|
| 1 | 15 | 15 | - |
| 2 | 15 | 15 | - |
| 3 | 15 | 15 | - |
| 4 | 15 | 10 | El número 15 es el mayor |

| | | | | |
|---|---|--|---------------------|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 | |
| | | Versión: | 01 | |
| | | Página | 58/207 | |
| | | Sección ISO | 8.3 | |
| | | Fecha de emisión | 20 de enero de 2017 | |
| 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.

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

DATOS DE ENTRADA: Número entero.

DATOS DE SALIDA: La impresión del factorial del número.

DOMINIO: Todos los números naturales positivos.

SOLUCIÓN:

1. Solicitar un número entero.
2. Si el número entero es menor a cero regresar al punto 1.
3. Si el número entero es mayor a cero se crea una variable entera *contador* que inicie en 2 y una variable entera *factorial* que inicie en uno.
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 se muestra el resultado almacenado en la variable *factorial*.

Prueba de escritorio (X es el número entero del que se calculará el factorial):

| Iteración | X | factorial | contador | Salida |
|-----------|---|-----------|----------|-------------------------|
| 1 | 0 | 1 | 2 | El factorial de 0 es: 1 |

| Iteración | X | factorial | contador | Salida |
|-----------|-----|-----------|----------|--------|
| 1 | -2 | 1 | 2 | - |
| 2 | -67 | 1 | 2 | - |
| 3 | 5 | 1 | 2 | - |
| 4 | 5 | 2 | 3 | - |

| | | | | |
|---|---|--|---------------------|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 | |
| | | Versión: | 01 | |
| | | Página | 59/207 | |
| | | Sección ISO | 8.3 | |
| | | Fecha de emisión | 20 de enero de 2017 | |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | | |

| | | | | |
|---|---|-----|---|---------------------------|
| 5 | 5 | 6 | 4 | - |
| 6 | 5 | 24 | 5 | - |
| 7 | 5 | 120 | 6 | El factorial de 5 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 de 7 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 del ejecutor el buen seguimiento del algoritmo para obtener un correcto resultado.

Ejercicio 1

PROBLEMA: Seguir el algoritmo para obtener una figura

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

SALIDA: Figura correcta.

Algoritmo

- Dibuja una V invertida. Empieza desde el lado izquierdo, sube, y baja hacia el lado derecho, no levantes el lápiz.
- 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 60/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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.
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. Completa el triángulo con una línea hacia la marca superior. Así completarás el triángulo.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 61/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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, saltándote 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 62/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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: | 01 |
| | | Página | 63/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 64/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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:

Elaborar 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.

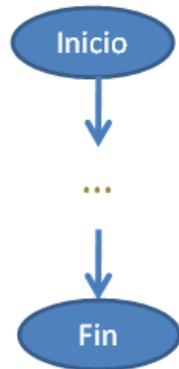
Dentro de las ciencias de la computación, un diagrama de flujo es la representación gráfica de un algoritmo. La correcta construcción de estos diagramas es fundamental para la etapa de codificación, ya que, a partir del diagrama de flujo es posible codificar un programa en algún lenguaje de programación.

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 65/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

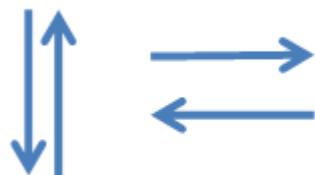
Formas de los diagramas de flujo

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.

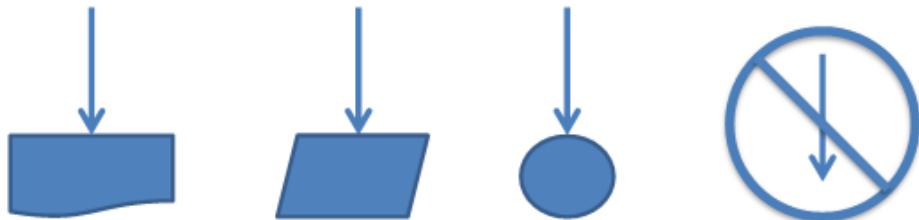


2. Las líneas utilizadas para indicar la dirección del flujo del diagrama deben ser rectas, verticales u horizontales, exclusivamente.



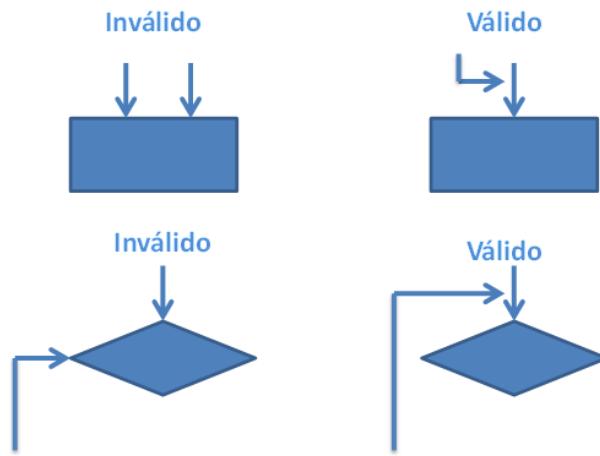
| | | | |
|---|---|--|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: Versión: Página: Sección ISO Fecha de emisión | MADO-17 01 66/207 8.3 20 de enero de 2017 |
| Facultad de Ingeniería | Area/Departamento: Laboratorio de computación salas A y B | | |
| La impresión 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.



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.

| | | |
|---|---|--|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 67/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |



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.



Representa el inicio o el fin del diagrama de flujo.

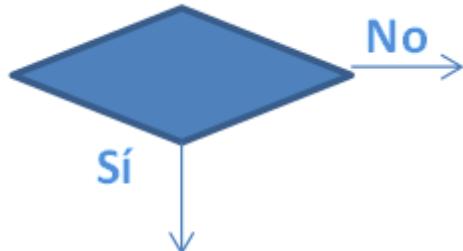


Datos de entrada. Expresa lectura de datos.

| | | |
|------------------------|---|--|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 |
| | | Versión: 01 |
| | | Página 68/207 |
| | | Sección ISO 8.3 |
| | | Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/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. Impresión del 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: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 69/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

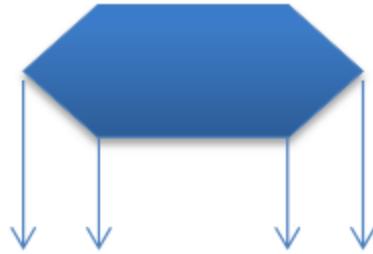
Conexión entre diferentes páginas.



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.





Manual de prácticas del Laboratorio de Fundamentos de programación

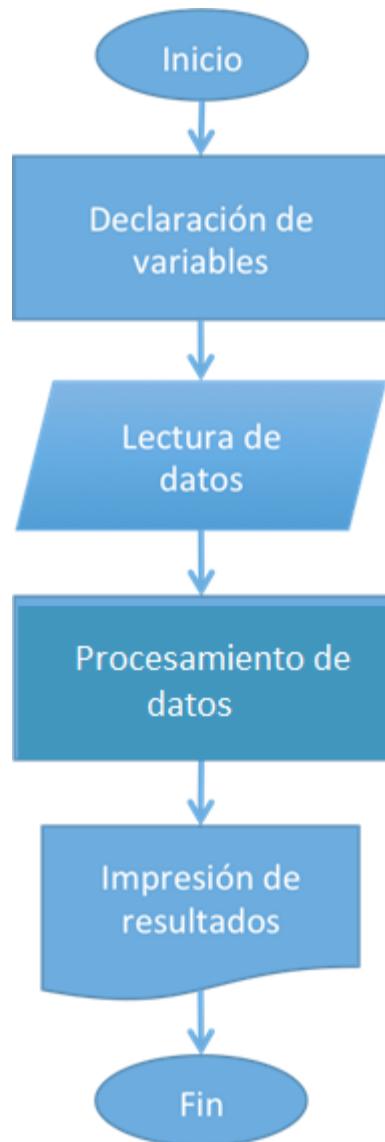
| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 70/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

El diagrama de flujo para construir un diagrama de flujo es el siguiente:



| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 71/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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

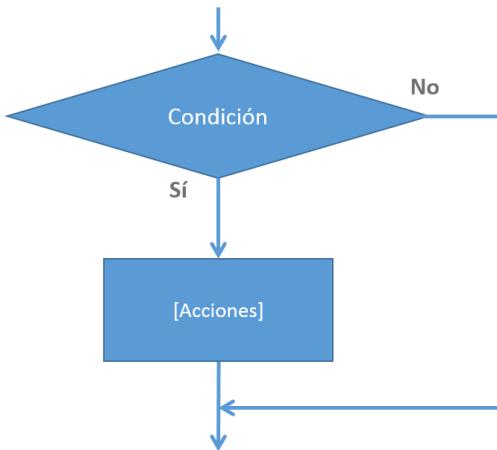
```
x: REAL
x ← 5.8
x ← x*2
```

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 ejecuta una acción o se ejecuta la otra).

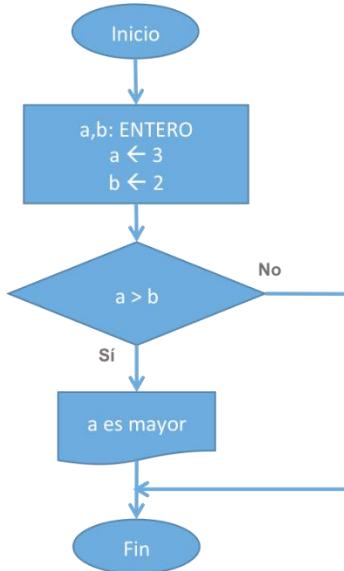
La estructura de control de flujo más simple es la estructura condicional SI (IF), su sintaxis es la siguiente:

| | | |
|---|---|--|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 72/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |



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

Ejemplo

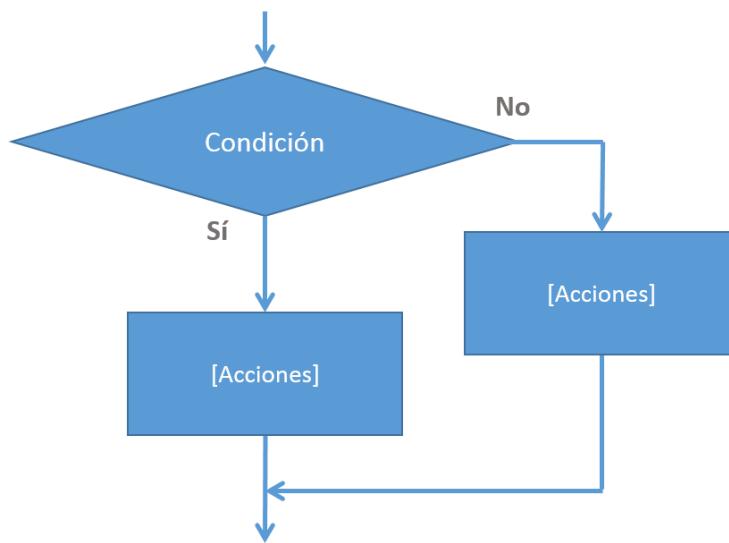


// >>> a es mayor

NOTA: La línea // >>> valor, indica el resultado que genera el ejemplo.

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 73/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

La estructura condicional completa es SI-DE LO CONTRARIO (IF-ELSE):



Se evalúa la expresión lógica y si se cumple (si la condición es verdadera) se ejecutan las instrucciones del bloque Sí. Si no se cumple la condición se ejecutan las instrucciones del bloque No. Al final el programa sigue su flujo normal.



Manual de prácticas del Laboratorio de Fundamentos de programación

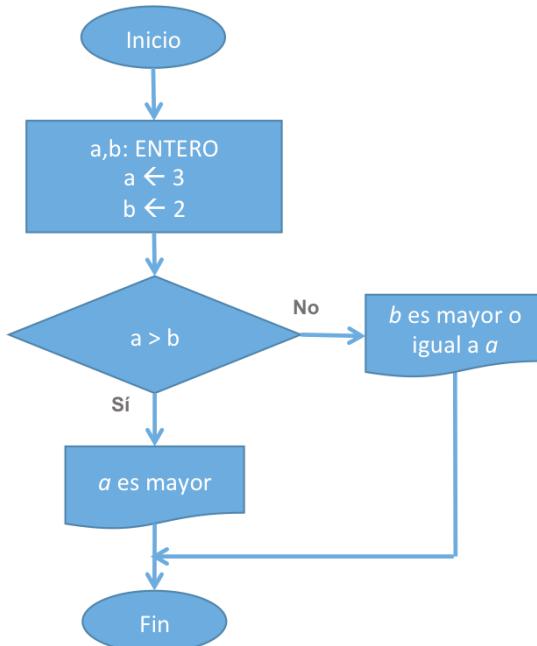
| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 74/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Ejemplo



// >>> b es mayor

La estructura condicional SELECCIONAR-CASO valida el valor de la variable que está en el hexágono y comprueba si es igual al valor que está definido en cada caso (líneas queemanan del hexágono). Si la variable no tiene el valor de algún caso se va a la instrucción por defecto (*).



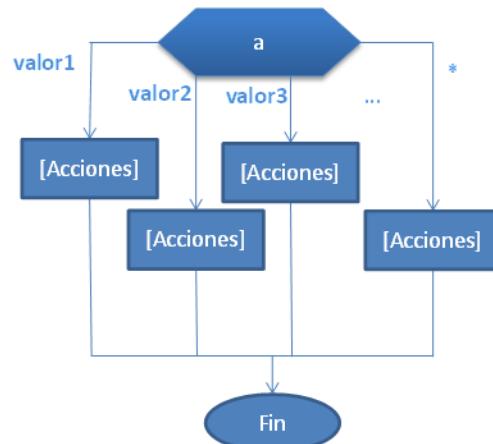
Manual de prácticas del Laboratorio de Fundamentos de programación

| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 75/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

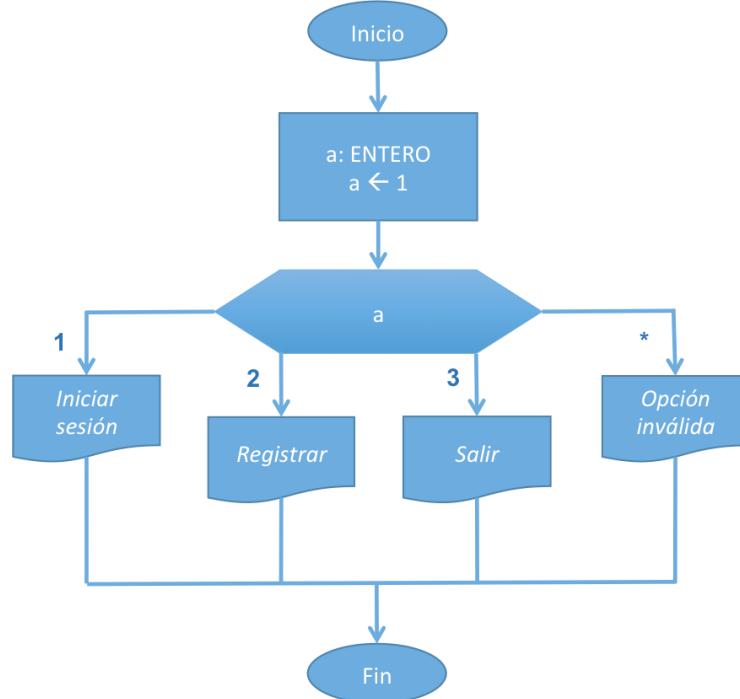
Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada



Ejemplo



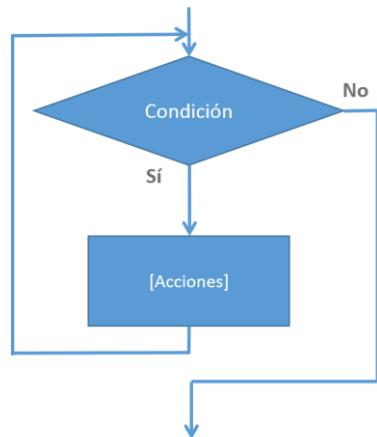
// >>> "Iniciar sesión"

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 76/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

Estructuras de control iterativas o repetitivas

Las estructuras de control de flujo **iterativas o repetitivas** (también llamadas cíclicas) permiten ejecutar 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 procede a ejecutar el bloque de instrucciones de la estructura, de lo contrario rompe el ciclo y continúa el flujo normal del programa.





Manual de prácticas del Laboratorio de Fundamentos de programación

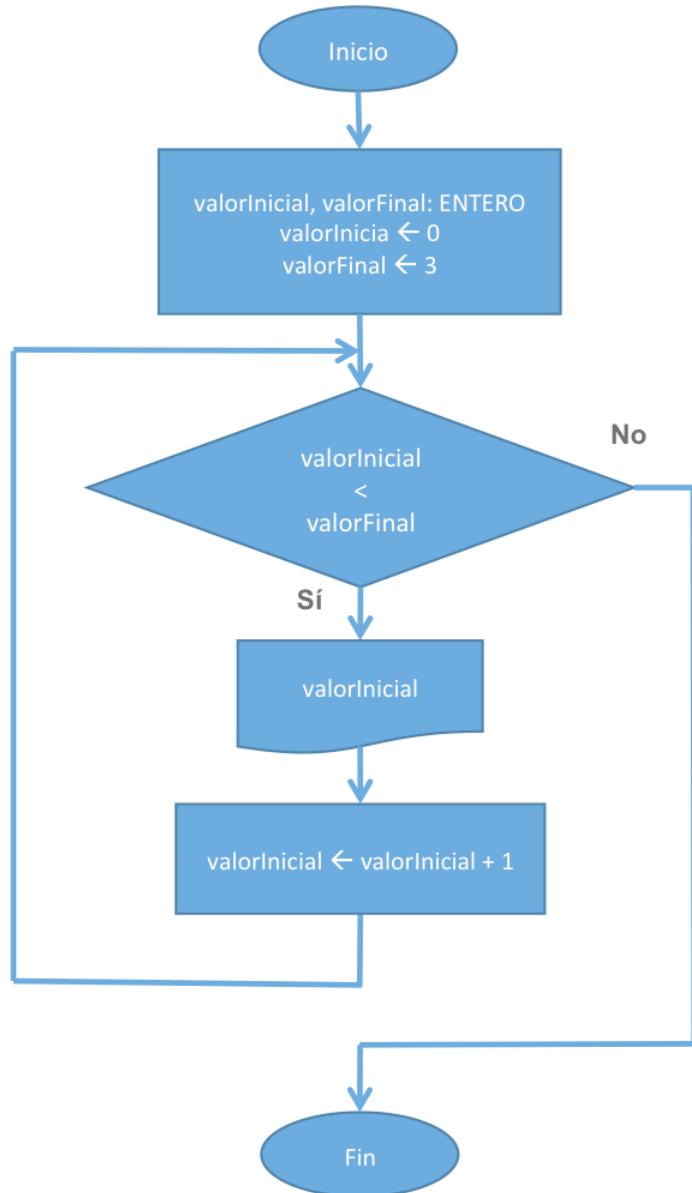
| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 77/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

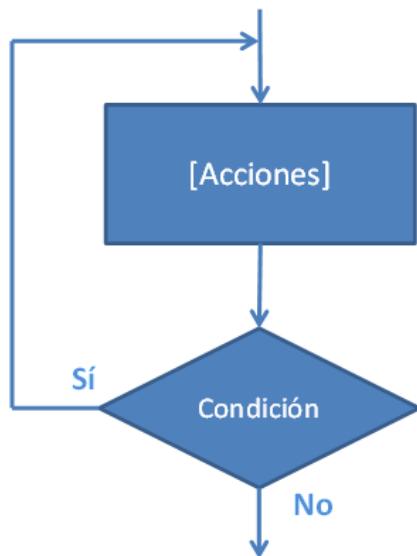
Ejemplo



```
//>>> 0  
//>>> 1  
//>>> 2
```

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 78/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

La estructura HACER-MIENTRAS primero ejecuta las instrucciones descritas en la estructura y al final valida la expresión lógica.



Si la condición se cumple vuelve a ejecutar las instrucciones de la estructura, de lo contrario rompe el ciclo y sigue el flujo del algoritmo. Esta estructura asegura que, por lo menos, se ejecuta una vez el bloque de la estructura, ya que primero ejecuta y después pregunta por la condición.



Manual de prácticas del Laboratorio de Fundamentos de programación

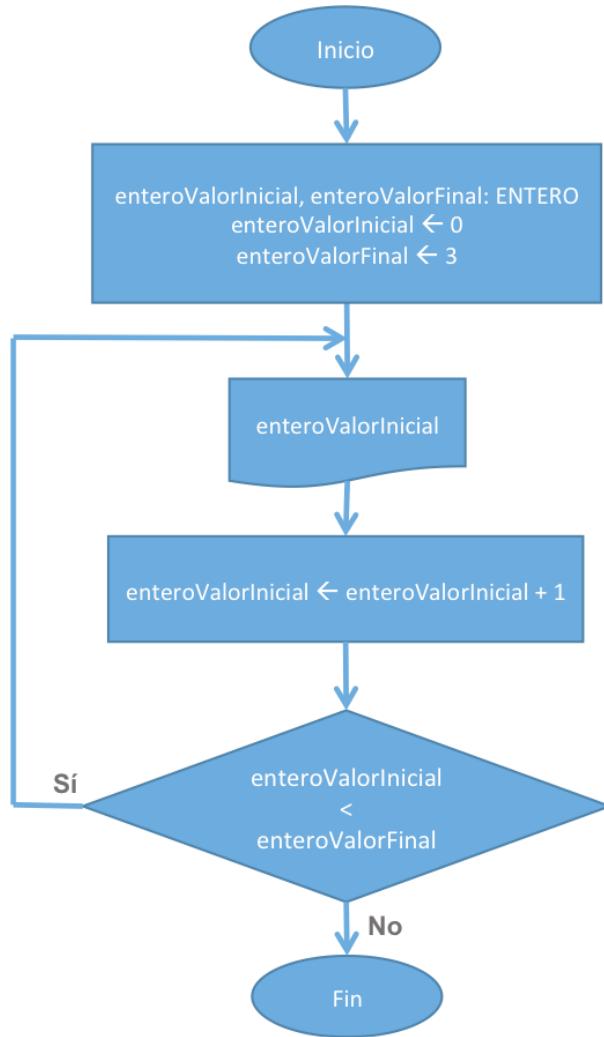
| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 79/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

Facultad de Ingeniería

Área/Departamento:
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Ejemplo



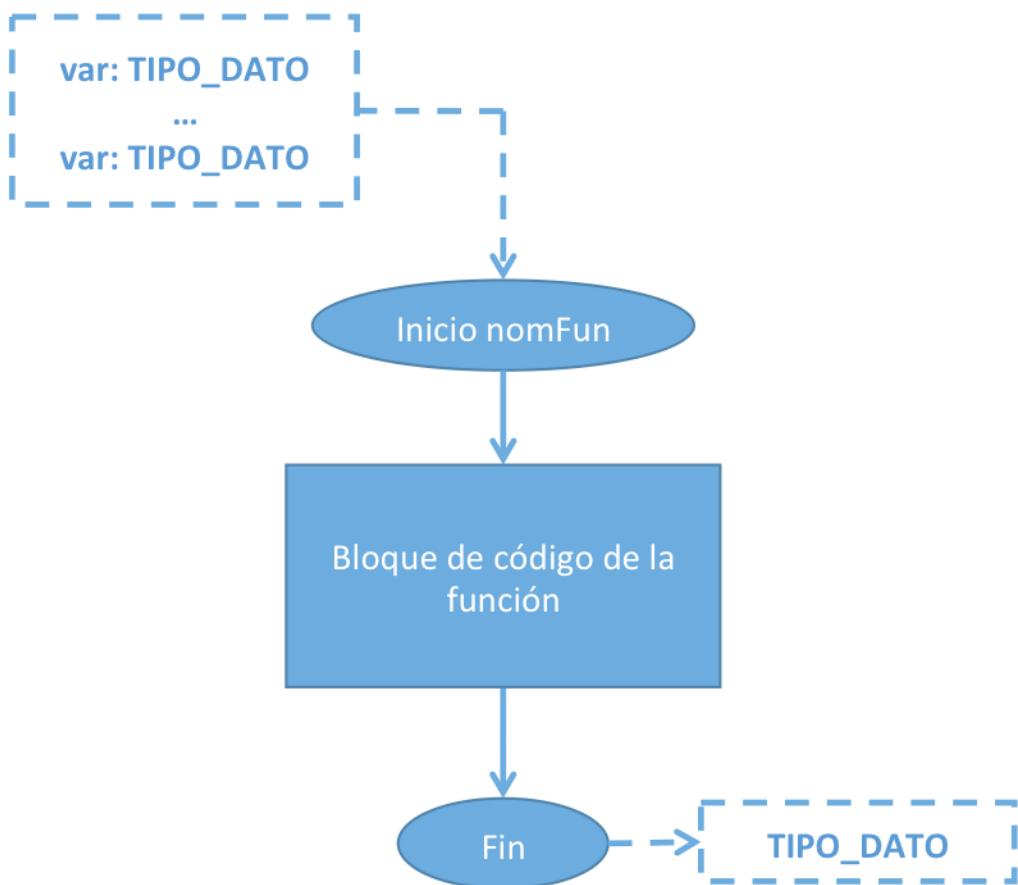
```
// >>> 0
// >>> 1
// >>> 2
```

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 80/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

Funciones

Cuando la solución de un problema es muy compleja se suele ocupar el diseño descendente (divide y vencerás). Este diseño implica la división de un problema en varios subprocessos más sencillos que juntos forman la solución completa. A estos subprocessos les llaman módulos o funciones.

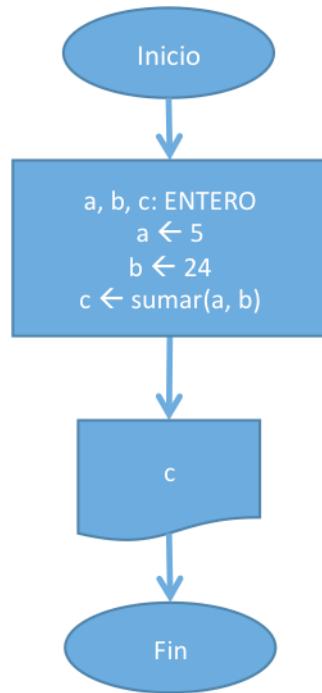
Una función está constituida por un identificador de función (nombre), de cero a n parámetros de entrada y un valor de retorno:



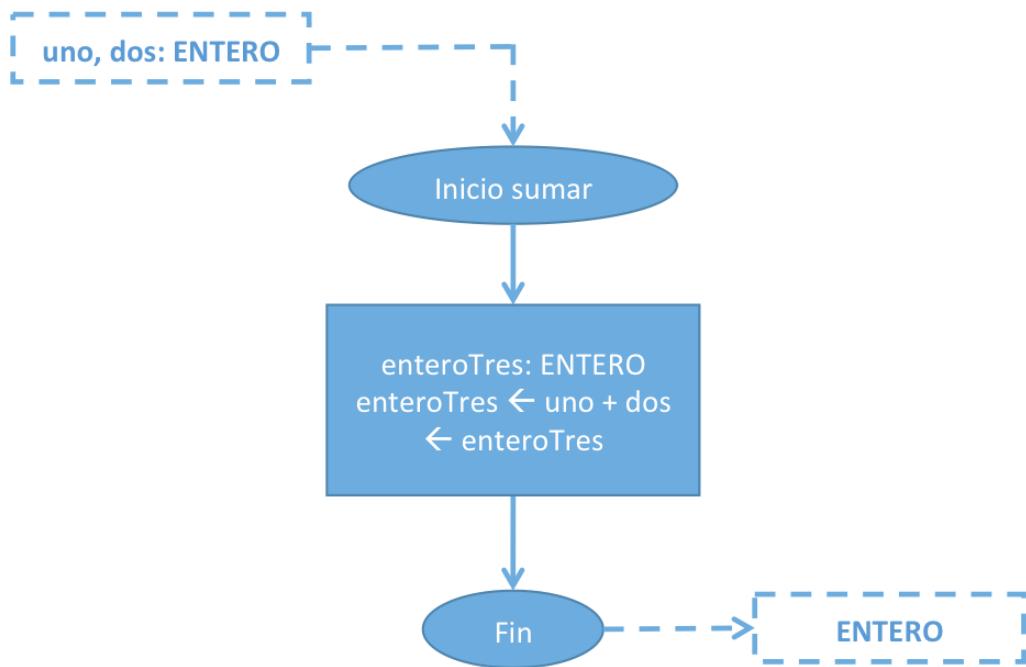
| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 81/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

nomFun es el nombre con el que llama a la función. Las funciones pueden o no recibir algún parámetro (tipo de dato) como entrada, si la función recibe alguno se debe incluir en el recuadro inicial (el que apunta al nombre de la función). Todas las funciones pueden regresar un valor al final de su ejecución (un resultado) para ello se debe definir el dominio del conjunto de salida (tipo de dato).

Ejemplo



| | | |
|---|---|--|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 82/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |



// >>> 29

Descripción

La primera función que se ejecuta es 'principal', ahí se crean las variables (uno y dos) y, posteriormente, se manda llamar a la función 'sumar'. La función 'sumar' recibe como parámetros dos valores enteros y devuelve como resultado un valor de tipo entero, que es la suma de los valores que se enviaron como parámetro.

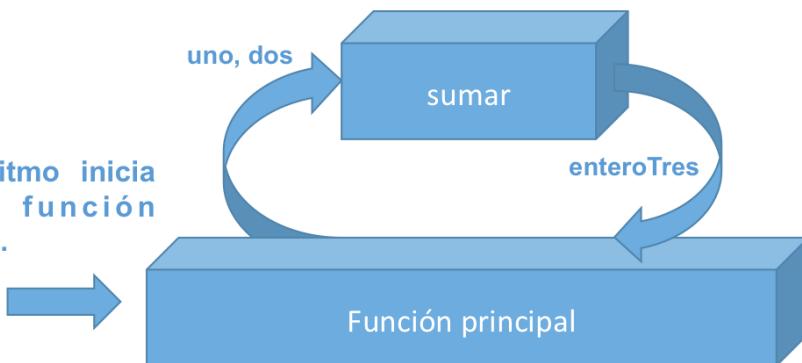
Para la función 'principal' los pasos que realiza la función 'sumar' son transparentes, es decir, solo manda a llamar a la función y espera el parámetro de retorno.

La siguiente figura permite analizar la función a través del tiempo. El algoritmo inicia con la función principal, dentro de esta función se hace una llamada a una función externa (sumar). Sumar realiza su proceso (ejecuta su algoritmo) y devuelve un valor a la función principal, la cual sigue su flujo hasta que su estructura secuencial llega a su fin.

| | | |
|---|---|--|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 83/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

La función sumar realiza su proceso (Acciones) y regresa el resultado (enteroTres) a la función que la mandó llamar (la función principal).

El algoritmo inicia con la función principal.



La función principal puede mandar llamar a otras funciones. En este caso llama a la función suma.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 84/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 85/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 86/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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:

Elaborar 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 del mismo. 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: | 01 |
| | | Página | 87/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 programa.
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. La lectura de datos se realiza, por defecto, desde el teclado, que es la entrada estándar del sistema. La escritura de datos se realiza, por defecto, en la pantalla, que es la salida estándar del sistema.

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>:<tipoDeDatos>
```

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

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 88/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Ejemplo

contador: ENTERO

producto: REAL

continuar: BOOLEANO

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 ".".

Ejemplo

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

```
usuario:REG domicilio // variable llamada usuario de tipo registro
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 89/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

```

usuario.calle := "Av. Imán"
usuario.número := 3000
usuario.ciudad := "México"

```

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 real (/), división entera (div), 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 AND, OR y NOT se describe a continuación:

| 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 camelio. Para nombrar variables y nombres de funciones se debe hacer uso de la notación de camelio.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 90/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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 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

```
// variables
realAreaDelTriangulo: REAL    // lower camel case
EnteroRadioCirculo: REAL      // upper camel case

// funciones
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
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 91/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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 ejecuta una acción o se ejecuta la otra)

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

```
SI condición ENTONCES
    [Acción]
FIN SI
```

Se evalúa la expresión lógica y si se cumple (si la condición es verdadera) se ejecutan las instrucciones del bloque [Acción]. Si no se cumple la condición, se continúa con el flujo normal del programa.

Ejemplo

```
INICIO
    a,b: ENTERO
    a := 3
    b := 2
    SI a > b ENTONCES
        ESCRIBIR "a es mayor"
    FIN SI
FIN

// >>> a es mayor
```

NOTA: La línea `//>>> valor`, indica el resultado que genera el ejemplo.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 92/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

La estructura condicional completa es SI-DE LO CONTRARIO:

```

SI cond_booleana ENTONCES
    [Acciones SI]
FIN SI
DE LO CONTRARIO
    [Acciones DE LO CONTRARIO]
FIN DE LO CONTRARIO

```

Se evalúa la expresión lógica y si se cumple (si la condición es verdadera) se ejecutan las instrucciones del bloque SI [Acciones SI]. Si no se cumple la condición se ejecutan las instrucciones del bloque DE LO CONTRARIO [Acciones DE LO CONTRARIO]. Al final el pseudocódigo sigue su flujo normal.

Ejemplo

```

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

```

La estructura condicional SELECCIONAR-CASO 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 -> [Acción]
    CASO valor2 -> [Acción]
    CASO valor3 -> [Acción]
    DEFECTO -> [Acción]
FIN SELECCIONAR

```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 93/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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

// >>> "Iniciar sesión"

```

Estructuras de control iterativas o repetitivas

Las estructuras de control de flujo **iterativas o repetitivas** (también llamadas cíclicas) permiten ejecutar 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 ejecutar el bloque de instrucciones de la estructura, de lo contrario rompe el ciclo y continúa el flujo normal del pseudocódigo.

```

MIENTRAS condición ENTONCES
    [Acción]
FIN MIENTRAS

```

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

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 94/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Ejemplo

```

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

//>>> 0
//>>> 1
//>>> 2

```

La estructura HACER-MIENTRAS primero ejecuta las instrucciones descritas en la estructura y al final valida la expresión lógica.

```

HACER
    [Acción]
    MIENTRAS condición

```

Si la condición se cumple vuelve a ejecutar las instrucciones de la estructura, de lo contrario rompe el ciclo y sigue el flujo del pseudocódigo. Esta estructura asegura que, por lo menos, se ejecuta una vez el bloque de la estructura, ya que primero ejecuta 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

```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 95/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

```
// >>> 0
// >>> 1
// >>> 2
```

Funciones

Cuando la solución de un problema es muy compleja se suele ocupar el diseño descendente (divide y vencerás). Este diseño implica la división de un problema en varios subprocessos más sencillos que juntos forman la solución completa. A estos subprocessos se les llaman métodos o funciones.

Una función está constituida por un identificador de función (nombre), de cero a n parámetros de entrada y un valor de retorno:

```
INICIO
  FUNC identificador (var:TipoDato,..., var:TipoDato) RET: TipoDato
    [Acciones]
  FIN FUNC
FIN
```

El identificador es el nombre con el que llama a la función. Las funciones pueden o no recibir algún(os) parámetro(s) (tipo(s) de dato(s)) como entrada; si la función recibe alguno se debe incluir entre los paréntesis. Todas las funciones pueden regresar un valor al final de su ejecución (el resultado).

Todas las estructuras de control de flujo (secuencial, condicional y repetitivas o iterativas) deben ir dentro de alguna función.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 96/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Ejemplo

INICIO

```

FUNC principal (vacío) RET: vacío
    a, b, c: ENTERO
    a := 5
    b := 24
    c := sumar(a, b)
    ESCRIBIR c
FIN FUNC

```

FIN

INICIO

```

** Función que suma dos números enteros
FUNC sumar (uno:ENTERO, dos: ENTERO) RET: ENTERO
    enteroTres: ENTERO
    enteroTres:= uno + dos
    RET enteroTres
FIN FUNC

```

FIN

// >>> 29

NOTA: Los dos asteriscos (**) dentro de un pseudocódigo se utilizan para hacer un comentario y, por tanto, lo que esté en la misma línea y después de los ** no es parte del algoritmo y no se toma en cuenta. Es una buena práctica realizar comentarios sobre una función o sobre un bloque del algoritmo para guiar sobre el funcionamiento del mismo.

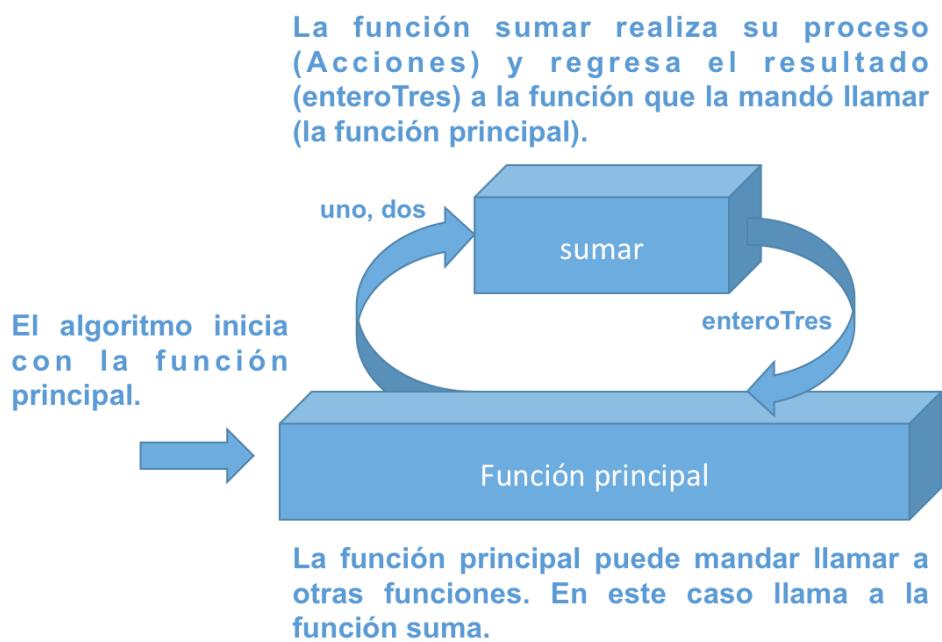
Descripción

La primera función que se ejecuta es 'principal', ahí se crean las variables (uno y dos) y, posteriormente, se manda llamar a la función 'sumar'. La función 'sumar' recibe como parámetros dos valores enteros y devuelve como resultado un valor de tipo entero, que es la suma de los valores que se enviaron como parámetro.

Para la función 'principal' los pasos que realiza la función 'sumar' son transparentes, es decir, solo manda a llamar a la función y espera el parámetro de retorno.

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 97/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

La siguiente figura permite analizar el pseudocódigo a través del tiempo. El algoritmo inicia con la función principal, dentro de esta función se hace una llamada a una función externa (sumar). Sumar realiza su proceso (ejecuta su algoritmo) y devuelve un valor a la función principal, la cual sigue su flujo hasta que su estructura secuencial (las instrucciones del pseudocódigo) llega a su fin.



| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 98/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 99/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 de C (editores, compilación y ejecución)



Elaborado por:

Ing. Laura Sandoval Montaño
Juan Francisco De reza Trujillo

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 100/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 de C (editores, compilación y ejecución)

Objetivo:

Conocer y usar los ambientes y herramientas para el desarrollo y ejecución de programas en Lenguaje C, como editores y compiladores en diversos sistemas operativos.

Actividades:

- Utilizando un editor de GNU/Linux, crear un archivo de texto
- Modificar/actualizar un archivo ya existente con un editor GNU/Linux.
- Crear, compilar y ejecutar un programa simple escrito en C en GNU/Linux
- En algún entorno de desarrollo de Windows, crear, compilar y ejecutar un programa simple escrito en C.

Introducción

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

Este es muy utilizado ya que la forma de dar instrucciones es muy cercana a lo que un humano podría abstraer, es decir, las instrucciones no son tal cual las que una computadora podría entender, para ello se necesitaría conocer a fondo el microprocesador, el sistema operativo entre otros aspectos. Por esta razón, C es conocido como un lenguaje de alto nivel, esto significa a que las instrucciones podrían ser entendidas fácilmente por un humano. En contraparte, un lenguaje de bajo nivel, son instrucciones que son cercanas a lo que la máquina puede entender y difícilmente pueden ser comprendidas por una persona que no tenga conocimientos de la máquina en que operarán. Algunos autores consideran al lenguaje C como un lenguaje de mediano nivel, ya que no es totalmente transparente sino tiene elementos que tienen que ver con la arquitectura de la máquina a la hora de programar.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 101/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Otra característica de C, es que es muy poderoso en el aspecto de combinar 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 a 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 correr en cualquier máquina siempre y cuando exista un compilador de C hecho para tal.

Para realizar un programa usando el lenguaje C, es necesario pensar primero en el sistema operativo que corre sobre la máquina y posteriormente, si este sistema cuenta con interfaz gráfica o sólo posee línea de comandos. A veces, se puede pensar siempre en sólo usar sistemas operativos con interfaz gráfica dado a que su manejo es más sencillo, sin embargo, esta se encuentra limitada para operar toda la funcionalidad del sistema operativo además de que consume recursos de cómputo que pueden ser indispensables para equipos donde el rendimiento es imprescindible. Una vez que se han seleccionado estos elementos, se necesita buscar qué opciones de editores y compiladores están disponibles.

Editores de C

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 de 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, etc., y será interpretado hasta que se haga una lectura de éste. Un

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 102/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

procesador de texto permite dar formato al texto, a la hoja donde está escrito, incrustar imágenes, etc., 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” puede ser el nombre del archivo a editar o el nombre de un archivo nuevo que se creará con VI. Es válido incluir la ruta donde se localiza o localizará el archivo. Existen más métodos de apertura para usuarios más avanzados. VI tiene tres modos de operación.

Modo comando

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.
- **ndd** 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.

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 103/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

- **p** pega un contenido copiado o borrado.
- **u** deshace el último cambio.

```
Ejemplo del editor de textos VI.
~
```

1, 32 Todo

Figura 1: VI en modo comando.

Modo de última línea

Se puede acceder a él desde el *modo de última línea*. Es muy similar al *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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 104/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Ejemplo del editor de textos VI.

:sg!

Figura 2: VI en modo de última línea.

Modo insertar

Este modo permite insertar texto. Las teclas presionadas ya no harán una acción como en el *modo comando* sino será el contenido que formará el texto del documento. Se puede desplazar con las **flechas del teclado** y borrar con la tecla de **retroceso** o de **suprimir**.

Para ingresar al *modo insertar* existen varios comandos:

- **i** pasa al modo insertar poniendo el texto a la izquierda del cursor.
 - **a** pasa al modo insertar poniendo el texto a la derecha del cursor.
 - **A** pasa al modo insertar colocando el texto al final de la línea donde el cursor se encuentra.
 - **I** pasa al modo insertar colocando el texto al principio de la línea donde el cursor se encuentra.
 - **O** coloca una línea arriba de la línea seleccionada por el cursor y pasa al modo insertar.
 - **o** coloca una línea debajo de la seleccionada por el cursor y pasa al modo insertar.

Para salir del modo *insertar* debe presionarse la tecla **Esc**. Para verificar que se encuentra en modo insertar es se puede ver -- **insertar** -- en la última línea del editor.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 105/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

```
Ejemplo del editor de textos VI.

-- INSERTAR --          2,1          Todo
```

Figura 3: VI en modo de insertar.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 106/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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 sino proporciona una interfaz que describe los comandos básicos.

NANO es un editor clon de otro editor llamado PICO.

Para iniciar NANO, debe taclearse desde la línea de comandos:

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

Donde “nombre_archivo” puede ser 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.

Modificado |

Ejemplo del editor NANO.

```
^G Ver ayuda ^O Guardar ^R L Fichero ^Y PÃ;g Ant ^K CortarTxt ^C Pos act a
^J Justificar ^W Buscar ^V PÃ;g Sig ^U UnCut Text ^T OrtografÃ;a
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 107/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Figura 4: El editor de texto NANO.

```
nano help text
```

The nano editor is designed to emulate the functionality and ease-of-use of the UW Pico text editor. There are four main sections of the editor. The top line shows the program version, the current filename being edited, and whether or not the file has been modified. Next is the main editor window showing the file being edited. The status line is the third line from the bottom and shows important messages. The bottom two lines show the most commonly used shortcuts in the editor.

The notation for shortcuts is as follows: Control-key sequences are notated with a caret (^) symbol and can be entered either by using the Control (Ctrl) key or pressing the Escape (Esc) key twice. Escape-key sequences are notated with the Meta (M) symbol and can be entered using either the Esc, Alt, or Meta key depending on your keyboard setup. Also, pressing Esc twice and then typing a three-digit decimal number from 000 to 255 will enter the character with the corresponding value. The following keystrokes are available in the main editor window. Alternative keys are shown in parentheses: █

```
█ Pág Sig █ Línea siguiente M-/ Última línea
```

Figura 5: Editor de texto NANO. Ayuda del Editor.

GEDIT

Es un editor de texto de entorno gráfico para sistemas basados en UNIX y se encuentra por defecto en el entorno de GNOME.

Permite todas las funciones básicas que se pueden encontrar en la mayoría de los editores como copiar, pegar, guardar, etcétera, con las mismas combinaciones de teclas que ya son conocidas por los usuarios. También cuenta con menús en la parte superior que permite acceder a todas las funciones del editor por lo que no requiere conocimientos avanzados para poderlo usar.

A diferencia de los editores de línea de comandos, la mayoría de los editores con interfaz gráfica tienen funciones como enumerar cada línea, sangría automática (en los lenguajes de programación se suele dejar tabulaciones antes de iniciar cada línea que ayudan a jerarquizar un programa codificado con el fin de aumentar la legibilidad y en algunos cuantos son de vital importancia), corrector ortográfico y coloreado sintáctico (colorear los distintos componentes sintácticos de ciertos lenguajes de programación con la finalidad de facilitar el desarrollo con elementos visuales).

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 108/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Notepad

Es un editor de texto plano de entorno gráfico disponible en todas las ediciones de Windows. A diferencia de GEDIT es muy limitado en funcionalidad, la cual es más parecida a un editor de línea de comandos, de hecho, es la evolución de uno de ellos, EDIT que era el editor para MS-DOS. A pesar de lo anterior, es conveniente usarlo cuando no existen herramientas adicionales para editar archivos de texto plano.

Notepad++

Es un editor de texto plano diseñado para ejecutarse en entorno gráfico con sistema operativo Windows; es de código libre. Éste tiene gran variedad de funciones que ayudan a los desarrolladores escribir programas de manera eficaz, como el autocompletado, corrector ortográfico, coloreado sintáctico, edición múltiple de archivos, resaltado de paréntesis, etcétera. Soporta gran variedad de lenguajes de programación y permite instalar aditamentos para aumentar su funcionalidad.

GitHub Atom

Es otro editor de texto de reciente aparición y de gran versatilidad para programadores porque tiene varias funciones además de soporte para distintos lenguajes de programación.

Tiene licencia MIT, está hecho para correr en entornos gráficos en sistemas operativos Linux, Windows y Mac OS X.

La mayoría de usuarios prefieren usar Notepad++ como su editor base a pesar de que Atom puede ser una competencia directa para éste. Atom además está diseñado para usuarios experimentados mientras que Notepad++ es tanto para iniciantes como avanzados.

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 y depende totalmente del hardware de la computadora y el sistema operativo que corre sobre ella.

Recordando, un programa en C es universal, por lo que cada una de las instrucciones que lo conforman debe poder entenderlas muchos de los equipos en el mercado, aunque su naturaleza sea distinta. Por ello, un compilador depende del equipo, porque es un traductor que transforma ese lenguaje universal a un programa ejecutable que sólo puede correr ese equipo.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 109/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Un programa en C, tampoco puede ser escrito de manera arbitraria sino respetando 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 correr 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 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 en un futuro. La mayoría de estos errores provocan que el compilador no pueda generar el programa ejecutable y muestra 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 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 volver a repetir este paso de manera continua.

A veces el compilador arroja advertencias durante el proceso, se generará el archivo ejecutable, pero puede tener problemas a la hora de ejecución por lo que es mejor investigar de qué tratan o porqué se generaron.

A continuación, se presentan dos compiladores que pueden ser de utilidad.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 110/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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 Linux. El compilador trabaja en línea de comandos.

Existe también una versión modificada que puede correr 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*):

```
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 está 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
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 111/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

LCC

Es un compilador similar a GCC de uso libre diseñado para ejecutarse en sistemas operativos Windows, sean de 64 bits o 32 bits.

Para poder hacer uso de LCC debe haber sido instalado previamente y agregado al PATH del sistema (es la ruta que sigue el sistema operativo para encontrar la ubicación de un ejecutable al ser llamado desde el símbolo de sistema). Se tiene que agregar al PATH por lo menos *lcc.exe* y *lcclnk.exe* localizados por defecto en *C:\lcc\bin*.

A diferencia de GCC, la compilación consiste en dos pasos, el primero genera un archivo objeto y el segundo a partir de éste genera el programa ejecutable. Existen opciones adicionales para usuarios avanzados a la hora de invocar al compilador.

Si se tiene un programa llamado *calculadora.c* se debe establecer primero la ruta donde se encuentra el archivo y luego generar el archivo objeto *calculadora.obj* con el siguiente comando:

```
lcc calculadora.c
```

Después se tiene que generar el programa ejecutable *calculadora.exe* por medio de:

```
lcclnk calculadora.obj
```

TCC

Es un compilador de uso libre diseñado para ejecutarse en sistemas operativos Windows.

Para poder hacer uso de TCC debe haber sido instalado previamente y agregado al PATH del sistema al igual que LCC.

Se tiene también como en GCC y LCC diferentes opciones para usuarios avanzados.

Si se tiene un programa llamado *calculadora.c* se tiene que establecer primero la ruta donde se encuentra el archivo. El siguiente comando permite generar el programa ejecutable y ejecutarlo:

```
tcc -run calculadora.c
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 112/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Otros compiladores e IDE

Un compilador simplemente es un programa que tiene como entrada un archivo de texto con el programa codificado en C y cuya salida es el programa ejecutable y la salida para marcar errores o advertencias.

Una IDE significa entorno de desarrollo integrado por sus siglas en inglés y combina un editor de textos con un compilador además de varias herramientas que facilitan la programación haciendo todo lo mencionado en esta práctica invisible para el programador. Incluso la ejecución es más sencilla desde una IDE.

Un IDE famoso diseñado para MS-DOS fue Borland Turbo C. LCC, al instalarse, incluye un IDE llamado wedit en el que basta escribir el programa en su editor y compilar y ejecutar el programa con la tecla F9.

En la siguiente lista se presentan varios compiladores e IDE para varios sistemas operativos y arquitecturas. Algunos compiladores soportan otros lenguajes además de C.

- GCC
- Embarcadero (Borland)
- Ch
- Code::Blocks,
- Cygwin
- Dev-C++
- DJGPP
- TCC
- EMX/RSXNT
- LCC para Windows de 32 o 64 bits
- Microsoft Visual C/C++
- MinGW para Windows de 32 o 64 bits
- Miracle C
- Orange C
- Pelles C
- Watcom

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 recomienda exhaustivamente que se haga desde *símbolo de sistema*. Como el programa realizado es para línea de comandos, si se ejecuta en entorno gráfico en Windows el programa sólo se abrirá y se cerrará sin poder ver los resultados de la ejecución, aunque el programa haga sus funciones. Existen métodos para evitar que el programa se cierre, pero suelen no

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 113/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

corresponder a ANSI C por lo que se desaconseja. Es mejor ejecutar el programa en el símbolo de sistema porque, aunque el programa finalice su ejecución, los resultados continuarán siendo visibles en la consola.

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
```

Y en Windows, teniendo un programa llamado *calculadora.exe* debe teclearse en símbolo de sistema:

```
calculadora.exe
```

En ambos casos localizándose previamente en la ruta donde se encuentra el ejecutable. En Windows a veces puede omitirse mencionar *.exe*.

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

```
calculadora 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 a lo que ANSI C señala, pero lo que se programó 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 aparente funcionar, 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 114/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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. No obstante, este programa tiene que correr en un equipo con 512 MB de memoria principal. El sistema operativo y otros procesos están ocupando ya una parte de esta memoria, aunque el sistema operativo haga lo posible por ejecutar el programa, será con un rendimiento pobre o simplemente el sistema se colgará.

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.

Bibliografía

- Dr. Pedro Alberto Enríquez Palma. Editor VI. Consulta: septiembre de 2016. Disponible en: <http://www.unirioja.es/cu/enriquez/docencia/Quimica/vi.pdf>
- Francisconi Hugo Adrian. Nano. Consulta: septiembre de 2016. Disponible en: <http://francisconi.org/linux/comandos/nano>
- G2 Crowd. ATOM vs. Notepad++. Consulta: septiembre de 2016. Disponible en: <https://www.g2crowd.com/compare/atom-vs-notepad>
- Gerald Pfeifer (GCC team). GCC, the GNU Compiler Collection. Consulta: septiembre de 2015. Disponible en: <https://gcc.gnu.org>
- MinGW.org. MinGW - Minimalist GNU for Windows. Consulta: septiembre de 2015. Disponible en: <http://www.mingw.org>
- White-Hat Hacking. Uso de gcc bajo Linux. Consulta: septiembre de 2016. Disponible en: <https://whitehathacking.wordpress.com/2010/10/31/uso-de-gcc-bajo-linux/>
- Willus.org. Win32/64 C/C++ Compilers Page. Consulta: septiembre de 2016. Disponible en: <http://www.willus.com/ccomp.shtml>
- Fabrice Bellard. Tiny C Compiler. Consulta: septiembre de 2015. Disponible en: <http://bellard.org/tcc/>

| | | | |
|---|---|--|---------------------|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 115/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Fundamentos de Lenguaje C



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 116/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Fundamentos de Lenguaje C

Objetivo:

Elaborar 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 programa en lenguaje C que tenga definidas variables de varios tipos, se les asigne valores adecuados (por lectura o asignación directa) y muestre su valor en la salida estándar.
- En un programa en C, asignar valores a variables utilizando expresiones aritméticas; algunas con uso de cambio de tipo (cast)
- 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, como lo son Pascal, Python, Fortran o PHP. 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 dentro de la etapa de *codificación* del problema. Esta etapa va muy unida a la etapa de *pruebas*:

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 117/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

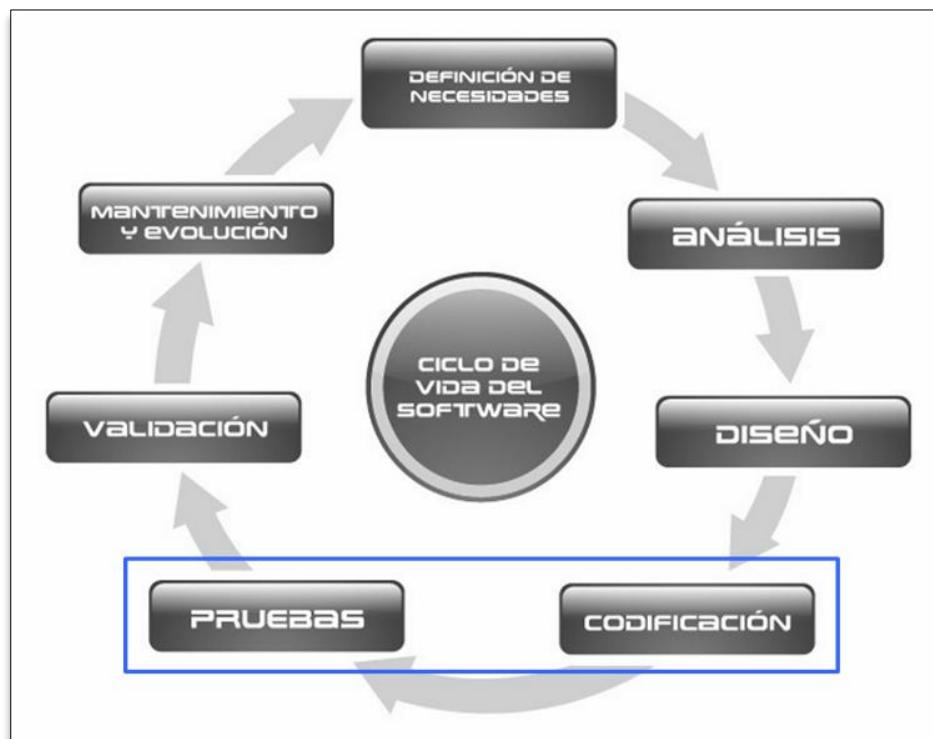


Figura 1: Ciclo de vida del software, resaltando las etapas de *codificación* y *pruebas*, las cuales se cubrirán en esta práctica.

Lenguaje de programación C

El proceso de desarrollo del lenguaje C se origina con la creación de un lenguaje llamado BCPL, que fue desarrollado por Martin Richards.

BCPL tuvo influencia en un lenguaje llamado B, el cual se usó en 1970 y fue inventado por Ken Thompson, esto permitió el desarrollo de C en 1971, el cual lo inventó e implementó Dennis Ritchie.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 118/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

C es un lenguaje de programación de propósito general que ofrece como ventajas economía de expresión, control de flujo y estructuras de datos y un conjunto de operadores.

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
- Iteración

Por otro lado, C es un lenguaje compilado, es decir, existe un programa (llamado compilador) que, a partir de un código en lenguaje C, genera un código objeto (ejecutable).

Para crear un programa en C se siguen tres etapas principales: edición, compilación y ejecución.

- Edición: Se escribe el código fuente en lenguaje C desde algún editor de textos.
- Compilación: A partir del código fuente (lenguaje C) se genera el archivo en lenguaje máquina (se crea el programa objeto o ejecutable).
- Ejecución: El archivo en lenguaje máquina se puede ejecutar en la arquitectura correspondiente.

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 ejecutarán únicamente las instrucciones que estén definidas dentro de la función principal. La función principal 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: | 01 |
| | | Página | 119/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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/>.
 *
 * Author: Jorge A. Solano
 *
 */
```

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 termine 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: | 01 |
| | | Página | 120/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Ejemplo

```
// Comentario por línea
// Otro comentario por línea
/* Comentario por bloque */
/* Comentario por bloque
   que puede ocupar
   varios renglones */
```

Código con comentarios

```
#include <stdio.h>

main() {
    // 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
    */
}
```

NOTA. Al iniciar el programa se deben agregar todas las bibliotecas que se van a utilizar en el mismo, es decir, funciones externas necesarias para ejecutar el programa. En lenguaje C la biblioteca estándar de entrada y salida está definida en 'stdio.h' (standard in out) y provee, entre otras, funciones para lectura y escritura de datos que se verán a continuación.

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 121/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

Declaración de variables

Para declarar variables en C se sigue la siguiente sintaxis:

[modificadores] tipoDeDatos identificador [= valor];

Por lo tanto, una variable puede tener modificadores (éstos se analizarán más adelante y son opcionales), debe declarar el tipo de dato que puede contener la variable, debe declarar el identificador (nombre o etiqueta) con el que se va a manejar el valor y se puede asignar un valor inicial a la variable (opcional).

También es posible declarar varios identificadores de un mismo tipo de dato e inicializarlos en el mismo renglón, lo único que se tiene que hacer es separar cada identificador por comas.

tipoDeDatos identificador1 [= valor], identificador2 [= valor];

Tipos de datos

Los tipos de datos básicos en C son:

- Caracteres: codificación definida por la máquina.
- Enteros: números sin punto decimal.
- Flotantes: números reales de precisión normal.
- Dobles: números reales de doble precisión.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 122/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Las variables enteras que existen en lenguaje C son:

| <i>Tipo</i> | <i>Bits</i> | <i>Valor Mínimo</i> | <i>Valor Máximo</i> |
|-----------------------|-------------|---------------------------|----------------------------|
| <i>signed char</i> | 8 | -128 | 127 |
| <i>unsigned char</i> | 8 | 0 | 255 |
| <i>signed short</i> | 16 | -32 768 | 32 767 |
| <i>unsigned short</i> | 16 | 0 | 65 535 |
| <i>signed int</i> | 32 | -2,147,483,648 | 2 147 483 647 |
| <i>unsigned int</i> | 32 | 0 | 4 294 967 295 |
| <i>signed long</i> | 64 | 9 223 372 036 854 775 808 | 9 223 372 036 854 775 807 |
| <i>unsigned long</i> | 64 | 0 | 18 446 744 073 709 551 615 |
| <i>enum</i> | 16 | -32 768 | 32 767 |

Si se omite el clasificador por defecto se considera 'signed'.

Las variables reales que existen en lenguaje C son:

| <i>Tipo</i> | <i>Bits</i> | <i>Valor Mínimo</i> | <i>Valor Máximo</i> |
|--------------------|-------------|---------------------|---------------------|
| <i>float</i> | 32 | 3.4 E-38 | 3.4 E38 |
| <i>double</i> | 64 | 1.7 E-308 | 1.7 E308 |
| <i>long double</i> | 80 | 3.4 E-4932 | 3.4 E4932 |

Las variables reales siempre poseen signo.

Para poder acceder al valor de una variable se requiere especificar el tipo de dato. Los especificadores que tiene lenguaje C para los diferentes tipos de datos son:

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 123/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

| <i>Tipo de dato</i> | <i>Especificador de formato</i> |
|-----------------------------|---------------------------------|
| <i>Entero</i> | %d, %i, %ld, %li, %o, %x |
| <i>Flotante</i> | %f, %lf, %e, %g |
| <i>Carácter</i> | %c, %d, %i, %o, %x |
| <i>Cadena de caracteres</i> | %s |

El especificador de dato se usa para guardar o imprimir el valor de una variable.

Para imprimir un valor entero en base 10 se pueden usar los especificadores %d o %i (%ld ó %li para enteros largos), para imprimir un valor entero en base 8 se utiliza el especificador %o y para imprimir un valor entero en base 16 se utiliza el especificador %x.

Un valor real se puede imprimir con el especificador %f para reales de precisión simple, %lf para reales de doble precisión, %e (notación científica) y %g (redondea la parte fraccionaria a 3 dígitos significativos).

Una variable de tipo carácter se puede imprimir con el especificador %c, con los especificadores %i o %d para imprimir el valor del código ASCII del carácter en base 10, con el especificador %o para imprimir el valor del código ASCII del carácter en base 8 o con el especificador %x para imprimir el valor del código ASCII del carácter en base 16.

El lenguaje C también permite manejar cadenas de caracteres y éstas se pueden imprimir con el especificador %s.

NOTA: Las cadenas de caracteres se manejarán cuando se aborde el tema de arreglos.

Identificador

Un identificador es el nombre con el que se va a almacenar en memoria un tipo de dato. Los identificadores siguen las siguientes reglas:

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 124/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

- Debe iniciar con una letra [a-z].
- Puede contener letras [A-Z, a-z], números [0-9] y el carácter guion bajo (_).

NOTA: A pesar de que variables como 'areadeltriangulo' o 'perimetro_del_cuadrado' son declaraciones válidas como identificadores, es una buena práctica utilizar la notación de camelCase para nombrar las variables como convención.

En la notación de camelCase los nombres de cada palabra empiezan con mayúscula y el resto se escribe con minúsculas (a excepción de la primera palabra, la cual inicia también con minúscula). No se usan puntos ni guiones para separar las palabras. Además, las palabras de las constantes se escriben con mayúsculas y se separan con guion bajo.

Código declaración de variables

```
#include <stdio.h>
/*
Este programa muestra la manera en la que se declaran y asignan variables
de diferentes tipos: numéricas (enteras y reales) y caracteres.
*/
int main() {
    // Variables enteras
    short enteroNumero1 = 32768;
    signed int enteroNumero2 = 55;
    unsigned long enteroNumero3 = -789;
    char caracterA = 65;           // Convierte el entero (ASCII) a carácter
    char caracterB = 'B';

    // Variables reales
    float puntoFlotanteNumero1 = 89.8;
    // La siguiente sentencia genera un error al compilar
    // porque los valores reales siempre llevan signo
    // unsigned double puntoFlotanteNumero2 = -238.2236;
    double puntoFlotanteNumero2 = -238.2236;
    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 125/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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

printf es una función para imprimir con formato, es decir, se tiene que especificar entre comillas el tipo de dato que se desea imprimir, también se puede combinar la impresión de un texto predeterminado:

```
printf("El valor de la variable real es: %lf", varReal);
```

scanf es una función que sirve para leer datos de la entrada estándar (teclado), para ello únicamente se especifica el tipo de dato que se desea leer entre comillas y en qué variable se quiere almacenar. Al nombre de la variable le antecede un ampersand (&), esto indica que el dato recibido se guardará en la localidad de memoria asignada a esa variable.

```
scanf ("%i", &varEntera);
```

Para imprimir con formato también se utilizan algunas secuencias de caracteres de *escape*, C maneja los siguientes:

- \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

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 126/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Código almacenar e imprimir variables

```
#include <stdio.h>
/*
Este programa muestra la manera en la que se declaran y asignan variables
de diferentes tipos: numéricas (enteras y reales) y caracteres, así como
la manera en la que se imprimen los diferentes tipos de datos.
*/
int main() {
    /* Es recomendable al inicio declarar
       todas las variables que se van a utilizar
       en el programa */
    // variables enteras
    int enteroNumero;
    char caracterA = 65;           // Convierte el entero a carácter (ASCII)

    // Variable reales
    double puntoFlotanteNumero;

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

    // Imprimir los valores con formato
    printf("\nImprimiendo las variables enteras:\a\n");
    printf("\tValor de enteroNumero = %i\n", enteroNumero);
    printf("\tValor de caracterA = %c\n", caracterA);
    printf("\tValor de puntoFlotanteNumero = %lf\n", puntoFlotanteNumero);

    printf("\nValor de enteroNumero en base 16 = %x\a\n", enteroNumero);
    printf("\tValor de caracterA en código hexadecimal = %i\n", enteroNumero);
    printf("\tValor de puntoFlotanteNumero en notación científica = %e\n",
        puntoFlotanteNumero);
    // La función getchar() espera un carácter para continuar la ejecución
    getchar();
    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 127/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Modificadores de alcance

Los modificadores que se pueden agregar al inicio de la declaración de variables son **const** y **static**.

El modificador **const** impide que una variable cambie su valor durante la ejecución del programa, es decir, permite para crear constantes. Por convención, las constantes se escriben con mayúsculas y se deben inicializar al momento de declararse.

Ejemplo

```
const int NUM_MAX = 1000;
const double PI = 3,141592653589793 3;
```

El modificador **static** indica que la variable permanece en memoria desde su creación y durante toda la ejecución del programa, es decir, permanece estática en la memoria.

Ejemplo

```
static int num;
static float resultado = 0;
```

NOTA: Cuando se llegue al tema de funciones se verá la utilidad de las variables estáticas.

Código variables estáticas y constantes

```
#include <stdio.h>
/*
Este programa muestra la manera en la que se declaran y asignan
las variables estáticas y las constantes.
*/
int main() {
    const int constante = 25;
    static char a = 'a';
    printf("Valor constante: %i\n", constante);
    printf("Valor estático: %c\n", a);
    // El valor de la variable declarada como constante no puede cambiar.
    // La siguiente línea genera un error al compilar si se quita el comentario:
    // constante = 30;
    // las variables estáticas sí pueden cambiar de valor
    a = 'b';
    printf("\nValor estático: %c\n", a);
    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 128/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Operadores

Los operadores aritméticos que maneja lenguaje C se describen en la siguiente tabla:

| <i>Operador</i> | <i>Operación</i> | <i>Uso</i> | <i>Resultado</i> |
|-----------------|------------------|-----------------|------------------|
| + | 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 |

Los operadores lógicos a nivel de bits que maneja el lenguaje C se describen en la siguiente tabla:

| <i>Operador</i> | <i>Operación</i> | <i>Uso</i> | <i>Resultado</i> |
|-----------------|----------------------------|------------|------------------|
| >> | 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 |

Moldeo o cast

El resultado de una operación entre dos tipos de datos iguales puede dar como resultado un tipo de dato diferente, en esos casos es necesario moldear el resultado. A este proceso se le conoce como *cast*.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 129/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Ejemplo:

```
// Si se tienen 2 enteros
int cinco = 5, dos = 2;

// La operación de división entre dos enteros
// genera un valor real, en este caso hay que
// moldear (cast) el resultado del lado derecho del
// igual para que corresponda con el lado izquierdo
// y se pueda asignar.
double res = (double)cinco/dos;

// Si no se hiciese el cast el resultado se truncaría.
```

Código operadores

```
#include <stdio.h>

/*
Este programa muestra la manera en la que se realiza un moldeo o cast.
También muestra como 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
*/
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");
    double res = (double)cinco/dos;           // Cast
    printf("5 / 2 = %lf\n",res);
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 130/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

```

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;
}

```

Expresiones lógicas

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

Los operadores de relación permiten comparar elementos numéricos, alfanuméricos, constantes o variables.

| <i>Operador</i> | <i>Operación</i> | <i>Uso</i> | <i>Resultado</i> |
|-----------------|------------------|------------|------------------|
| == | Igual que | '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 |

Los operadores lógicos permiten formular condiciones complejas a partir de condiciones simples.

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 131/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Area/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

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

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

El operador `++` agrega una unidad (1) a su operando. Es posible manejar preincrementos (`++n`) o posincrementos (`n++`).

El operador `--` resta una unidad (1) a su operando. Se pueden manejar predecrementos (`--n`) o posdecrementos (`n--`).

NOTA: Lenguaje C maneja los resultados booleanos (Verdadero o falso) con números enteros, cuando el resultado de una comparación es falso el valor regresado es cero, cuando la comparación es verdadera el valor regresado es 1.

Código expresiones lógicas

```
#include <stdio.h>
/*
Este programa ejemplifica el manejo de operaciones relacionales y los
operadores lógicos. También muestra la manera de incrementar o decrementar
una variable y la diferencia entre hacerlo antes (pre) o después (pos).
*/
int main (){
    int num1, num2, res;
    char c1, c2;
    double equis = 5.5;
    num1 = 7;
    num2 = 15;

    c1 = 'h';
    c2 = 'H';
```

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 132/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

```

printf("Expresiones de relación\n");
printf("¿ num1 es menor a num2 ? -> \t%d\n",num1<num2);
printf("¿ c1 es igual a c2 ?      -> \t%d\n",c1==c2);
printf("¿ c1 es diferente a c2 ? -> \t%d\n",c1!=c2);

printf("\nExpresiones lógicas\n");
printf("¿ num1 es menor a num2 Y c1 es igual a 'h' ? -> \t%d\n",
num1<num2 && c1 == 'h');
printf("¿ c1 es igual a 's' O c2 es igual a 'H' ?      -> \t%d\n",
c1 == 's' || c2 == 'H');
printf("¿ c1 es igual a 's' O c2 es igual a 'S' ?      -> \t%d\n",
c1 == 's' || c2 == 'S');

printf("\nIncrementos y decrementos\n");
printf("num1++ -> \t%d\n",num1++);
printf("--num1 -> \t%d\n",--num1);
printf("++equis -> \t%lf\n",++equis);

return 0;
}

```

Depuración de programas

Cuando un programa falla (no termina su ejecución de manera correcta) y la información enviada por el compilador es muy general, se puede ejecutar el programa en un contexto controlado para saber, exactamente, dónde está fallando. Se revisará este tema en la guía práctica de estudio **“Depuración de programas”** para conocer las diferentes herramientas que nos ayudan a encontrar los errores de un programa.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 133/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Referencias

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



- 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-del-software-sqa>

| | | | |
|---|---|--|---------------------|
| | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 134/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 08: Estructuras de selección



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 135/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 08: Estructuras de selección

Objetivo:

Elaborar 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 136/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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
 *
*/
```

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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 137/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

NOTA 1: Si el bloque de código a ejecutar consta de una solo 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 0 indica que la expresión lógica es falsa y cualquier número diferente de 0 indica que la expresión lógica es verdadera.

Código (estructura de control selectiva if)

```
#include<stdio.h>

/*
Este programa valida si el número a es mayor al número b.
*/

int main (){
    int a, b;
    a = 3;
    b = 2;

    if (a > b) {
        printf("\ta (%d) es mayor a b (%d).\n",a,b);
    }

    printf("\t\tEl programa sigue su flujo.\n");

    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 138/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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)

```
#include<stdio.h>

/*
Este programa comprueba que las condiciones son numéricas
0 -> falso
≠ 0 -> Verdadero
*/

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;
}
```

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
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 139/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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)

```
#include <stdio.h>

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

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: | 01 |
| Página | 140/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

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)

```
#include <stdio.h>

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

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: | 01 |
| | | Página | 141/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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. Al final de cada caso se ejecuta la instrucción break, 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 142/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 switch-case)

```
#include <stdio.h>

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

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;
}
```

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 143/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| 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 switch-case)

```
#include <stdio.h>

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

int main(){
    int op = 0;

    printf("\tMenú\n\n");
    printf("Elegir la opción deseada\n");
    printf("1) Ingresar\n");
    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;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 144/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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};
```

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};
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 145/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Código (variables tipo enumeración)

```
#include <stdio.h>

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

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: | 01 |
| | | Página | 146/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Código (variables tipo enumeración)

```
#include <stdio.h>

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

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 0;           // Valor entero en hexadecimal
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 147/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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)

```
#include <stdio.h>

/*
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.
*/

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: | 01 |
| | | Página | 148/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 149/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Estructuras de repetición



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 150/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 09: Estructuras de repetición

Objetivo:

Elaborar programas en C para la resolución de problemas básicos que incluyan las estructuras de repetición y la directiva *define*.

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*.
- Usar la directiva *define* para elaboración de código versátil.

Introducción

Las estructuras de repetición son las llamadas 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 151/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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
 *
 */
```

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 {}. Si 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 *n* veces. Su sintaxis es la siguiente:

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

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 152/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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

Código (estructura de repetición while)

```
#include <stdio.h>

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

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;
}
```

Código (estructura de repetición while)

```
#include <stdio.h>
/*
Este programa genera un ciclo infinito.
*/
int main(){

    // Al igual que en la estructura if-else
    //     0 -> falso
    // 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: 01 Página 153/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| 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 ene 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 ';'.

Código (estructura de repetición do-while)

```
#include <stdio.h>
/*
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'.
*/
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');

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

    return 0;
}
```



Manual de prácticas del Laboratorio de Fundamentos de programación

| | |
|------------------|---------------------|
| Código: | MADO-17 |
| Versión: | 01 |
| Página | 154/207 |
| Sección ISO | 8.3 |
| Fecha de emisión | 20 de enero de 2017 |

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 repetición do-while)

```
#include <stdio.h>
/* Este programa genera una calculadora básica. */
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");
                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);

    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 155/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 de repetición for

Lenguaje C posee la estructura de repetición *for* la cual permite realizar repeticiones cuando se conoce el número de elementos que se quiere recorrer. 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 antes o 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 parte 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 parte es opcional. La tercera parte 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 parte también es opcional.

Código (estructura de repetición for)

```
#include <stdio.h>
/*
Este programa genera un arreglo unidimensional de 5 elementos y
* accede a cada elemento del arreglo a través de un ciclo for.
*/
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;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 156/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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 <nombree> <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)

```
#include <stdio.h>
#define MAX 5

/*
 * 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.
 */

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

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 157/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

valor definido para MAX y en automático todos los arreglos y el recorrido de los mismos adquieren el nuevo valor (Mientras se use MAX para definir el o los arreglos y para realizar los recorridos).

Break

Algunas veces es conveniente tener la posibilidad de abandonar un ciclo. La proposición **break** proporciona una salida anticipada dentro de una estructura de repetición, tal como lo hace en un switch. Un *break* provoca que el ciclo que lo encierra termine inmediatamente.

Código (break)

```
#include <stdio.h>

/*
 * Este programa hace una suma de números. Si la suma rebasa la cantidad
 * de 50 el programa se detiene.
 */

#define VALOR_MAX 5

int main (){
    int enteroSuma = 0;
    int enteroNumero = 0;
    int enteroContador = 0;
    while (enteroContador < VALOR_MAX){
        printf("Ingrese un número:");
        scanf("%d", &enteroNumero);
        enteroSuma += enteroNumero;
        enteroContador++;
        if (enteroSuma > 50){
            printf("Se rebasó la cantidad límite.\n");
            break;
        }
    }
    printf("El valor de la suma es: %d\n", enteroSuma);
}
return 0;
}
```

Cuando se compila el programa, MAX se sustituye por 5.

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 158/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

Continue

La proposición **continue** provoca que inicie la siguiente iteración del ciclo de repetición que la contiene.

Código (continue)

```
#include <stdio.h>

/*
 * Este programa obtiene la suma de un LIMITE de números pares ingresados
 */

#define LIMITE 5

int main (){
    int enteroContador = 1;
    int enteroNumero = 0;
    int enteroSuma = 0;
    while (enteroContador <= LIMITE){
        printf("Ingrese número par %d:", enteroContador);
        scanf("%d",&enteroNumero);

        if (enteroNumero%2 != 0){
            printf("El número insertado no es par.\n");
            continue;
        }

        enteroSuma += enteroNumero;
        enteroContador++;
    }

    printf("La suma de los números es: %d\n", enteroSuma);

    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 159/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 160/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Depuración de programas



Elaborado por:

Ing. Laura Sandoval Montaño
Juan Francisco de Reza Trujillo

Revisado por:

Ing. Laura Sandoval Montaño

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 161/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Depuración de programas

Objetivo

Aprender las técnicas básicas de depuración de programas en C para revisar de manera precisa el flujo de ejecución de un programa y el valor de las variables; en su caso, corregir posibles errores.

Actividades:

- Revisar, a través de un depurador, los valores que va tomando una variable en un programa escrito en C, al momento de ejecutarse.
- Utilizando un depurador, revisar el flujo de instrucciones que se están ejecutando en un programa en C, cuando el flujo depende de los datos de entrada.

Introducción

Depurar un programa significa someterlo a un ambiente de ejecución controlado por medio de herramientas dedicadas a ello. Este ambiente permite conocer exactamente el flujo de ejecución del programa, el valor que las variables adquieren, la pila de llamadas a funciones, entre otros aspectos. Es importante poder compilar el programa sin errores antes de depurarlo.

Antes de continuar, es necesario conocer las siguientes definiciones (extraídas del Glosario IEEE610) ya que son parte latente del proceso de Desarrollo de Software:

Error. Se refiere a una acción humana que produce o genera un resultado incorrecto.

Defecto (Fault). Es la manifestación de un error en el software. Un defecto es encontrado porque causa una Falla (failure).

Falla (failure). Es una desviación del servicio o resultado esperado.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 162/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

La depuración de un programa es útil cuando:

- Se desea optimizar el programa: no basta que el programa se pueda compilar y se someta a pruebas que demuestren que funciona correctamente. Debe realizarse un análisis exhaustivo del mismo en ejecución para averiguar cuál es su flujo de operación y encontrar formas de mejorarlo (reducir el código, utilizar menos recursos llegando a los mismos resultados, hacer menos rebuscado al algoritmo), o bien, encontrar puntos donde puede fallar con ciertos tipos de entrada de datos.
- El programa tiene algún fallo: el programa no muestra los resultados que se esperan para cierta entrada de datos debido a que el programador cometió algún error durante el proceso de diseño. Muchas veces encontrar este tipo de fallos suele ser difícil, ya sea porque la percepción del programador no permite encontrar la falla en su diseño o porque la errata es muy pequeña, pero crucial. En este caso es de mucha utilidad conocer paso a paso cómo se ejecutan las estructuras de control, qué valor adquieren las variables, etc.
- El programa tiene un error de ejecución o defecto: cuando el programa está ejecutándose, éste se detiene inesperadamente. Suele ocurrir por error en el diseño o implementación del programa en las que no se contemplan las limitaciones del lenguaje de programación o el equipo donde el programa se ejecuta. Como el programa se detiene inesperadamente, no se conoce la parte del programa donde se provoca el defecto, teniendo que recurrir a la depuración para encontrarlo. El más común de este tipo de defecto es la “violación de segmento”.

Algunas funciones básicas que tienen en común la mayoría de los depuradores son las siguientes:

- Ejecutar el programa: se procede a ejecutar el programa en la herramienta de depuración ofreciendo diversas opciones para ello.
- Mostrar el código fuente del programa: muestra cuál fue el código fuente del programa con el número de línea con el fin de emular la ejecución del programa sobre éste, es decir, se indica qué parte del código fuente se está ejecutando a la hora de correr el programa.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 163/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

- Punto de ruptura: también conocido por su traducción al inglés *breakpoint*, sirve para detener la ejecución del programa en algún punto indicado previamente por medio del número de línea. Como la ejecución del programa es más rápida de lo que podemos visualizar y entender, se suelen poner puntos de ruptura para conocer ciertos parámetros de la ejecución como el valor de las variables en determinados puntos del programa. También sirve para verificar hasta qué punto el programa se ejecuta sin problemas y en qué parte podría existir el error, esto es especialmente útil cuando existe un error de ejecución.
- Continuar: continúa con la ejecución del programa después del punto de ruptura.
- Ejecutar la siguiente instrucción: cuando la ejecución del programa se ha detenido por medio del depurador, esta función permite ejecutar una instrucción más y detener el programa de nuevo. Esto es útil cuando se desea estudiar detalladamente una pequeña sección del programa. Si en la ejecución existe una llamada a función se ingresará a ella.
- Ejecutar la siguiente línea: es muy similar a la función anterior, pero realizará todas las instrucciones necesarias hasta llegar a la siguiente línea de código. Si en la ejecución existe una llamada a función se ignorará.
- Ejecutar la instrucción o línea anterior: deshace el efecto provocado por alguna de las funciones anteriores para volver a repetir una sección del programa.
- Visualizar el valor de las variables: permite conocer el valor de alguna o varias variables.

Dependiendo de la herramienta usada para compilar el programa, si es de consola o de terminal, su uso y las funciones disponibles variarán.

En las IDE (Entornos de Desarrollo Interactivo), suelen existir herramientas de depuración integradas de manera gráfica. Es muy común que existan dos modos de desarrollar un programa y producir el archivo ejecutable que son "Debug" y "Release". El primer modo se recomienda exclusivamente durante el desarrollo del programa para poder depurarlo continuamente durante cualquier prueba de ejecución. El segundo modo se establece cuando el programa ha sido terminado y totalmente probado.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 164/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Depuración de programas escritos en C con GCC y GDB

Para depurar un programa usando las herramientas desarrolladas por GNU, éste debe compilarse con información para depuración por medio del compilador GCC.

Para compilar, por ejemplo, un programa llamado *calculadora.c* con GCC con información de depuración, debe realizarse en una terminal con el siguiente comando:

```
gcc -g -o calculadora calculadora.c
```

El parámetro *-g* es quien indica que el ejecutable debe producirse con información de depuración.

Una vez hecho el paso anterior, debe usarse la herramienta GDB, la cual, es el depurador para cualquier programa ejecutable realizado por GCC.

Para depurar un ejecutable debe invocarse a GDB en la terminal indicando cuál es el programa ejecutable a depurar, por ejemplo, para depurar *calculadora*:

```
gdb ./calculadora
```

Al correr GDB se entra a una línea de comandos. De acuerdo al comando es posible realizar distintas funciones de depuración:

list o l: Permite listar diez líneas del código fuente del programa, si se desea visualizar todo el código fuente debe invocarse varias veces este comando para mostrar de diez en diez líneas. Se puede optar por colocar un número separado por un espacio para indicar a partir de qué línea desea mostrarse el programa. También es posible mostrar un rango de líneas introduciendo el comando y de qué línea a qué línea separadas por una coma. Ejemplo: *list 4,6*

b: Establece un punto de ruptura para lo cual debe indicarse en qué línea se desea establecer o bien también acepta el nombre de la función donde se desea realizar dicho paso. Ejemplo: *b 5*

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 165/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

- *d o delete*: Elimina un punto de ruptura, indicando cuál es el que debe eliminarse usando el número de línea. Ejemplo: *d 5*
- *clear*: Elimina todos los puntos de ruptura. Ejemplo: *clear*
- *info line*: Permite mostrar información relativa a la línea que se indique después del comando. Ejemplo: *info line 8*
- *run o r*: Ejecuta el programa en cuestión. Si el programa tiene un punto de ruptura se ejecutará hasta dicho punto, de lo contrario se ejecutará todo el programa.
- *c*: Continúa con la ejecución del programa después de un punto de ruptura.
- *s*: Continúa con la siguiente instrucción después de un punto de ruptura.
- *n*: Salta hasta la siguiente línea de código después de un punto de ruptura.
- *p o print*: Muestra el valor de una variable, para ello debe escribirse el comando y el nombre de la variable separados por un espacio. Ejemplo: *p suma_acumulada*
- *ignore*: Ignora un determinado punto de ruptura indicándolo con el número de línea de código. Ejemplo: *ignore 5*
- *q o quit*: Termina la ejecución de GDB.

GDB tiene más opciones disponibles que pueden consultarse con comandos como *help* o invocando desde la terminal del sistema *man gdb*.

Depuración de programas escritos en C con Dev-C++ 5.0.3.4

Dev-C++, es una IDE especializada para desarrollar programas escritos en C o C++. Si bien incorpora un editor de textos y un compilador integrados, también posee un depurador. Cabe destacar que por defecto Dev-C++ se basa en el compilador GCC y el depurador en GDB, aunque de manera gráfica ello es transparente para el usuario ya que todo simula una sola herramienta.

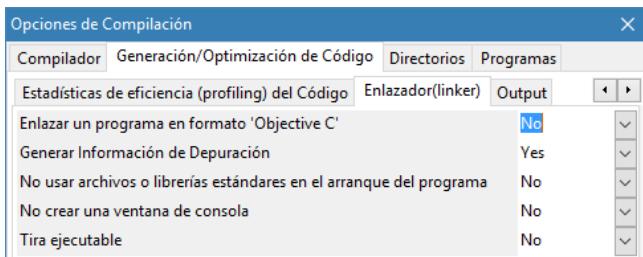
Cabe señalar, que si se desea utilizar Dev-C++ como herramienta de desarrollo de programas en C, debe estar instalado adecuadamente en el equipo para que funcione tanto el compilador como las herramientas de depuración. Si se usa sistema operativo Windows,

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 166/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

se recomienda encarecidamente usar la versión que se proporciona en <http://lcp02.fib.unam.mx> en la sección de *Servicios*.

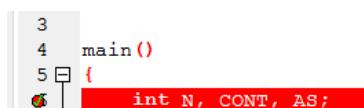
Antes de iniciar la depuración debe tenerse a la mano el archivo con el programa escrito en C o proceder a escribirlo en la misma IDE. Para ello debe usarse el menú *Archivo → Nuevo → Código Fuente* si se piensa usar la IDE para escribirlo o en su lugar *Archivo → Abrir Proyecto o Archivo* si ya existía el código fuente.

Una vez que se tiene el programa, debe activarse la opción de compilación generando información para el depurador. Para activar esta opción debe abrirse el menú *Herramientas → Opciones del Compilador* y acceder a la pestaña *Generación/Optimización de Código* y finalmente, en la subpestaña Enlazador (linker), activar la opción *Generar Información de Depuración*:



Después de realizar lo anterior, el programa realizado puede compilarse y ejecutarse con lo que ofrece el menú *Ejecutar*.

Para agregar puntos de ruptura, debe hacerse clic en la línea de código donde se desea colocar y ésta se volverá en color rojo. Para retirarlo se hace clic de nuevo en la línea y volverá a su color normal. Lo anterior se ve en la imagen siguiente:



```

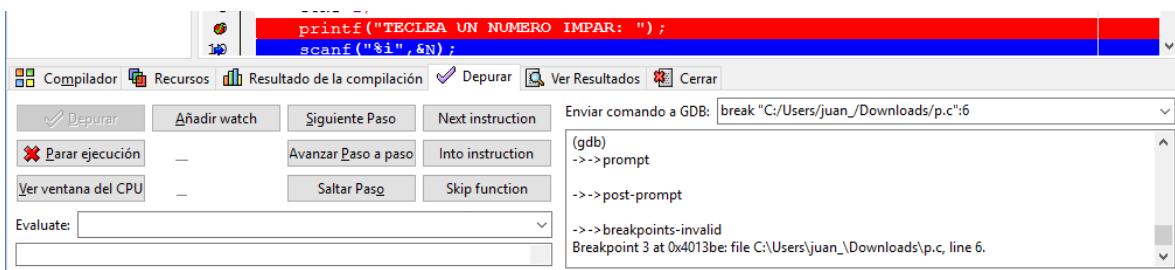
3
4 main()
5 {
    int N, CONT, AS;

```

Para depurar el programa, primero se debe compilar con el menú *Ejecutar → Compilar* y luego depurar con *Depurar → Depurar*. El programa se abrirá y se ejecutará hasta el primer punto de ruptura seleccionado. También se abrirá un cuadro de herramientas en la parte inferior del programa que tiene las principales herramientas de depuración en la parte

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 167/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

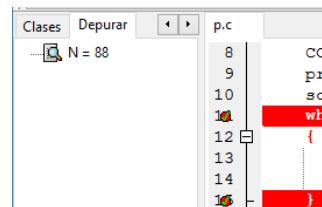
derecha. Cabe destacar que la línea que se ejecuta actualmente es la que se resalta en color azul:



Cuando se llega al punto de ruptura, se tienen diversas opciones, *Siguiente Paso* ejecuta la siguiente línea (si existe una iteración o función, la saltará), *Avanzar Paso a Paso* ejecuta instrucción por instrucción (una función o iteración serán ejecutadas instrucción por instrucción), *Saltar Paso* ejecuta hasta el siguiente punto de ruptura.

Para detener la depuración puede seleccionarse la opción *Parar ejecución*. La opción *Ver ventana del CPU* permite ver a detalle las instrucciones enviadas al procesador, registros de memoria involucrados y valor de cada una de las banderas en el procesador.

Finalmente, para estudiar el valor de cada variable, se puede recurrir a la función *Añadir Watch* y escribir el nombre de la variable. En un cuadro a la izquierda, se verá el nombre de la variable y su valor hasta el punto donde se está ejecutando el programa:



Cuando se utiliza esta IDE, se recomienda usar los accesos directos mencionados en los propios menús, lo cual permite usarla de manera óptima. El nombre de las funciones y menús pueden variar según el idioma en el que se haya instalado la IDE.

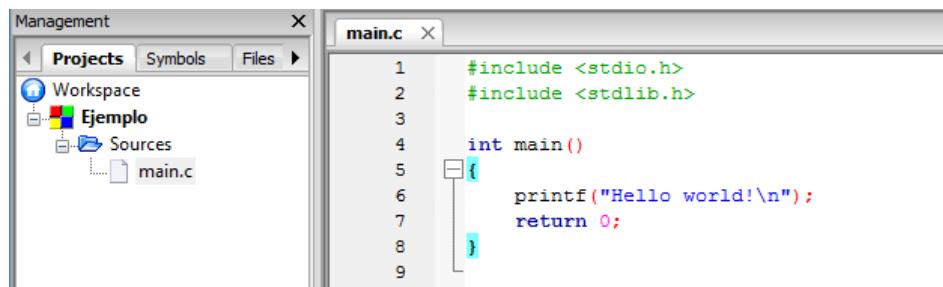
| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página: 168/207 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

Depuración de programas escritos en C con Code::Blocks 13.12

Code::Blocks, es otra IDE de código abierto que puede basarse en las mismas herramientas GNU que Dev-C++. Permite cambiarse por otros motores de compilación si se desea. Proporciona un editor de texto, un compilador integrado, herramientas de depuración, etc.

Para poder depurar, es necesario crear un nuevo proyecto desde el menú *File → New → Project* y elegir en el cuadro que aparece *Console Application* (recordar que, por ahora, todos los programas son desarrollados en modo de consola). Seguir el asistente para crear el proyecto eligiendo que se usará el lenguaje C, luego seleccionar un título adecuado para el proyecto, la ruta donde se creará y el título del archivo asociado al proyecto. Posteriormente, en el mismo asistente seleccionar *Create "Debug" Configuration* y *Create "Release" Configuration* en el mismo asistente con compilador *GNU GCC Compiler*. Nótese que existen dos carpetas asociadas que son */bin/debug* y */bin/release* que son donde se crearán los ejecutables de depuración y el final respectivamente. Se debe finalizar el asistente.

En la parte izquierda, se encontrará el nombre del archivo fuente del proyecto. Para ello navegar como se indica en la siguiente imagen y dar clic en *main.c*:



Dicho archivo debe editarse para formar el programa deseado. Cuando se está desarrollando es importante que esté seleccionado el modo de depuración, localizado en la barra de herramientas que se muestra:



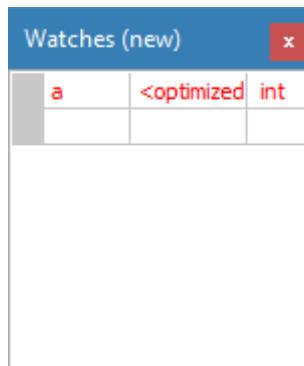
Como puede observarse, hay un menú despegable que tiene la opción *Debug* y *Release*, debe estar siempre seleccionada la primera opción hasta no haber terminado el programa

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 169/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

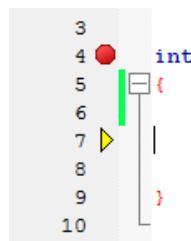
a desarrollar. Cuando todo esté listo, se cambiará a la segunda opción y se usará el archivo ejecutable que se localiza en la `/bin/reléase`, jamás el localizado en `/bin/debug` que solo tiene efectos de desarrollo.

A la izquierda del menú, se encuentran las opciones de compilación y ejecución del programa en el modo seleccionado. A la derecha se encuentran las opciones de depuración. La primera opción, *Debug/Continue*, permite ejecutar el programa en modo de depuración y reanudar la ejecución después de un punto de ruptura. La opción *Stop debugger*, detiene la depuración y permite continuar editando. Existen otras herramientas adicionales, entre ellas correr el programa hasta donde se encuentre el cursor en el texto, ejecutar la siguiente línea o la siguiente instrucción, todas ellas estudiadas anteriormente.

Para visualizar una variable, debe hacerse clic sobre ella con el depurador corriendo, y dar clic en *Watch 'variable'*, aparecerá un pequeño cuadro con la tabla de las variables que se desean visualizar y su valor:



Finalmente, para agregar un punto de ruptura, se tiene que hacer clic derecho sobre el número de línea de código y agregarlo, aparecerá un punto rojo en la línea, para quitarlo se tiene que hacer el mismo procedimiento.



| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 170/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

La parte del código que se está ejecutando por el depurador, se indica por medio de una flecha amarilla en el número de línea correspondiente.

Ejercicios propuestos

Para el siguiente código fuente, utilizar algún entorno de depuración para encontrar la utilidad del programa y la funcionalidad de los principales comandos de depuración, como puntos de ruptura, ejecución de siguiente línea o instrucción.

```
#include <stdio.h>

void main()
{
    int N, CONT, AS;
AS=0;
CONT=1;
printf("TECLEA UN NUMERO: ");
scanf("%i",&N);
while(CONT<=N)
{
    AS=(AS+CONT);
    CONT=(CONT+2);
}
printf("\nEL RESULTADO ES %i\n", AS);
}
```

Funcionalidad del programa:

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 171/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

El siguiente programa debe mostrar las tablas de multiplicar desde la del 1 hasta la del 10. En un principio no se mostraba la tabla del 10, luego después de intentar corregirse sin un depurador dejaron de mostrarse el resto de las tablas. Usar un depurador de C para averiguar el funcionamiento del programa y corregir ambos problemas.

```
#include <stdio.h>
void main()
{
    int i, j;

    for(i=1; i<10; i++)
    {
        printf("\nTabla del %i\n", i);
        for(j=1; j==10; j++)
        {
            printf("%i X %i = %i\n", i, j, i*j);
        }
    }
}
```

El siguiente programa muestra una *violación de segmento* durante su ejecución y se interrumpe; usar un depurador para detectar y corregir la falla.

```
#include <stdio.h>
#include <math.h>
void main()
{
    int K, X, AP, N;
    float AS;
    printf("EL TERMINO GENERICO DE LA SERIE ES: X^K/K!");
    printf("\nN=");
    scanf("%d", N);
    printf("X=");
    scanf("%d", X);
    K=0;
    AP=1;
    AS=0;
    while(K<=N)
    {
        AS=AS+pow(X, K)/AP;
        K=K+1;
        AP=AP*K;
    }
    printf("SUM=%le", AS);
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 172/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Bibliografía

- Gutiérrez Rodríguez, Javier Jesús. *Primeros pasos con GDB*. Consulta: octubre de 2016. Disponible en: http://www.lsi.us.es/~javierj/ssoo_ficheros/GuiaGDB.htm
- Ferreira, Amelia. *Depurador gdb*. Consulta: octubre de 2016. Disponible en: <http://learnassembler.com/gdbesp.html>
- Ferreira, Amelia. *Depurador gdb - uso de la opción -g de gcc*. Consulta: octubre de 2016. Disponible en: <http://learnassembler.com/opc.html>
- Gutiérrez, Erik Marín. *Depuración de programas Dev C++*. Consulta: octubre de 2016. Disponible en: <http://programacionymetodos.blogspot.mx/2012/05/depuracion-de-programas-dev-c.html>
- González Cárdenas, Miguel Eduardo; Marín Lara, Claudia Lorena; Noguerón Pérez, Pedro. *Apuntes De Computadoras Y Programación*. Universidad Nacional Autónoma de México.
- Pozo Coronado, Salvador. Primeros pasos con GDB. Consulta: octubre de 2016. Disponible en: <http://www.c.conclase.net/devcpp/?cap=depurar>

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 173/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Arreglos unidimensionales y multidimensionales



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Actualizado y revisado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 174/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Arreglos unidimensionales y multidimensionales

Objetivo:

Reconocer la importancia y utilidad de los arreglos, en la elaboración de programas que resuelvan problemas que requieran agrupar datos del mismo tipo, así como trabajar con arreglos tanto unidimensionales como multidimensionales.

Actividades:

- Elaborar un programa en lenguaje C que emplee arreglos de una dimensión.
- Resolver un problema que requiera el uso de un arreglo de dos dimensiones, a través de un programa en lenguaje C.
- Manipular arreglos a través de índices y apuntadores.

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. Los arreglos se utilizan para hacer más eficiente el código de un programa.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 175/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

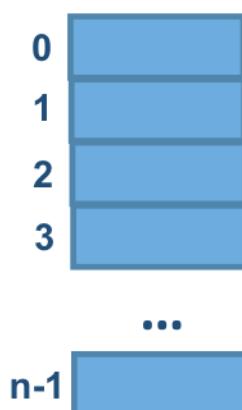
Licencia GPL de GNU

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
 *
 */
```

Arreglos unidimensionales

Un arreglo unidimensional de n elementos en la memoria se almacena de la siguiente manera:



| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 176/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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 en lenguaje C es la siguiente:

tipoDeDatos 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. Un arreglo puede ser de los tipos de dato entero, real, carácter o estructura.

NOTA: Los tipos de datos estructuras no se abordarán en esta práctica.

Código (arreglo unidimensional while)

```
#include <stdio.h>

/*
Este programa genera un arreglo unidimensional de 5 elementos y los
accede a cada elemento del arreglo a través de un ciclo while.
*/

int main (){
    #define TAMANO 5
    int lista[TAMANO] = {10, 8, 5, 8, 7};

    int indice = 0;

    printf("\tLista\n");
    while (indice < 5 ){
        printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
        indice += 1;          // análogo a indice = indice + 1;
    }

    printf("\n");

    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 177/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 for)

```
#include <stdio.h>

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

int main (){
#define TAMANO 5
int lista[TAMANO] = {10, 8, 5, 8, 7};

printf("\tLista\n");
for (int indice = 0 ; indice < 5 ; indice++){
    printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
}

printf("\n");

return 0;
}
```

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.

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

```
TipoDeDatos *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 acceder a la dirección de memoria de la misma (es lo que pasa cuando se lee un dato con scanf).

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 178/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

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 *.

Código (apuntadores)

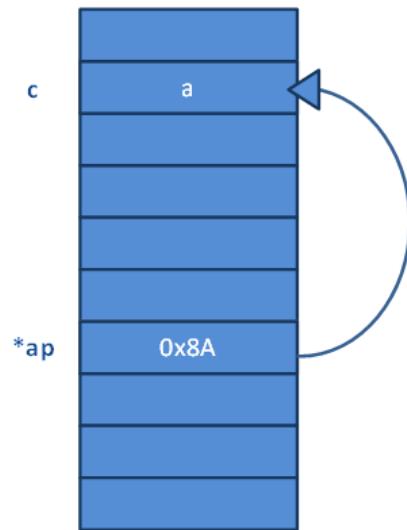
```
#include <stdio.h>

/*
Este programa crea un apuntador de tipo carácter.
*/

int main () {
    char *ap, c = 'a';
    ap = &c;

    printf("Carácter: %c\n",*ap);
    printf("Código ASCII: %d\n",*ap);
    printf("Dirección de memoria: %d\n",ap);

    return 0;
}
```



Un apuntador almacena la dirección de memoria de la variable a la que apunta

| | | |
|---|---|--|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página: 179/207 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

Código (apuntadores)

```
#include<stdio.h>

/*
Este programa accede a las localidades de memoria de distintas variables a
través de un apuntador.
*/

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");

    b = *apEnt;
    printf("b = *apEnt \t-> b = %i\n", b);

    b = *apEnt +1;
    printf("b = *apEnt + 1 \t-> b = %i\n", b);

    *apEnt = 0;
    printf("*apEnt = 0 \t-> a = %i\n", a);

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

    return 0;
}
```

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;
```

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 180/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B |
| La impresión de este documento es una copia no controlada | | |

Código (apuntadores)

```
#include <stdio.h>

/*
Este programa trabaja con aritmética de apuntadores para acceder a los
valores de un arreglo.
*/

int main () {
    int arr[] = {5, 4, 3, 2, 1};
    int *apArr;
    apArr = arr;

    printf("int arr[] = {5, 4, 3, 2, 1};\n");
    printf("apArr = &arr[0]\n");

    int x = *apArr;
    printf("x = *apArr \t -> x = %d\n", x);

    x = *(apArr+1);
    printf("x = *(apArr+1) \t -> x = %d\n", x);

    x = *(apArr+2);
    printf("x = *(apArr+2) \t -> x = %d\n", x);

    return 0;
}
```

Código (apuntadores en ciclo for)

```
#include <stdio.h>

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

int main (){
#define TAMANO 5
    int lista[TAMANO] = {10, 8, 5, 8, 7};
    int *ap = lista;
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 181/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

```

printf("\tLista\n");
for (int indice = 0 ; indice < 5 ; indice++){
    printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
}

printf("\n");

return 0;
}

```

Código (apuntadores en cadenas)

```

#include <stdio.h>

/*
Este programa muestra el manejo de cadenas en lenguaje C.
*/

int main(){
    char palabra[20];
    int i=0;

    printf("Ingrese una palabra: ");
    scanf("%s", palabra);
    printf("La palabra ingresada es: %s\n", palabra);

    for (i = 0 ; i < 20 ; i++){
        printf("%c\n", palabra[i]);
    }

    return 0;
}

```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 182/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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). Los tipos de dato que puede tolerar un arreglo multidimensional son: 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.

Código (arreglos multidimensionales)

```
#include<stdio.h>

/*
Este programa genera un arreglo de dos dimensiones (arreglo
multidimensional) y accede a sus elementos a través de dos ciclos
for, uno anidado dentro de otro.
*/

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

    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 183/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Código (arreglos multidimensionales con apuntadores)

```
#include<stdio.h>

/* Este programa genera un arreglo de dos dimensiones (arreglo
multidimensional) y accede a sus elementos a través de un apuntador utilizando
un ciclo for.
*/

int main(){
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, cont=0, *ap;
    ap = matriz;

    printf("Imprimir Matriz\n");
    for (i=0 ; i<9 ; i++){
        if (cont == 3){
            printf("\n");
            cont = 0;
        }
        printf("%d\t",*(ap+i));
        cont++;
    }
    printf("\n");

    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 184/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 185/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Funciones



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado y actualizado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 186/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: Funciones

Objetivo:

Elaborar programas en C donde la solución del problema se divide en funciones. Distinguir lo que es el prototipo o firma de una función y la implementación de ella, así como manipular 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, se inicia con 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 187/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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
 *
 */
```

Funciones

La sintaxis básica para definir una función es la siguiente:

```
valorRetorno 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 188/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Una función puede recibir parámetros de entrada, 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.

El valor de retorno de una función indica el tipo de dato que va a regresar la función al terminar el bloque de código de la misma. 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 regresar el elemento vacío (void).

El compilador C revisa 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:

valorRetorno nombre (*parámetros*);

La firma de una función está compuesta por tres elementos: el nombre de la función, los parámetros que recibe la función y el valor de retorno de 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 189/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Código (funciones)

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

/*
Este 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.
*/

// 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.

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 190/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Como ya se vio, un programa en C puede contener varias funciones. Las variables que se declaran 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.

```
void sumar() {
    int x;
    // ámbito de la variable x
}
```

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.

```
#include <stdio.h>

int resultado;

void multiplicar() {
    resultado = 5 * 4;
}
```

Código (Ámbito de las variables)

```
#include <stdio.h>

/*
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.
*/

void incremento();

// La variable enteraGlobal es vista por todas
// las funciones (main e incremento)
int enteraGlobal = 0;
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 191/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

```

int main(){
    // La variable cont es local a la función main
    for (int cont=0 ; cont<5 ; cont++){
        incremento();
    }

    return 999;
}

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 anteriormente, la firma de una función está compuesta por tres elementos: el nombre de la función, los parámetros que recibe la función y el valor de retorno de la función.

La función main también puede recibir parámetros. Debido a que la función main es la primera que se ejecuta en un programa, los parámetros de la función hay que enviarlos al ejecutar el programa. La firma completa de la función main es:

```
int main (int argc, char ** argv);
```

La función main puede recibir como parámetro de entrada un arreglo de cadenas al ejecutar el programa. 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 parámetros, el programa se debe ejecutar de la siguiente manera:

- En plataforma Linux/Unix
`./nombrePrograma arg1 arg2 arg3 ...`
- En plataforma Windows
`nombrePrograma.exe arg1 arg2 arg3 ...`

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 192/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Esto es, el nombre del programa seguido de los argumentos de entrada. Estos argumentos son leídos como cadenas de caracteres dentro del *argument vector*, donde en la posición 0 se encuentra el nombre del programa, en la posición 1 el primer argumento, en la posición 2 el segundo argumento y así sucesivamente.

Código (argumentos función main)

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

/*
Este programa permite manejar los argumentos enviados al ejecutarlo.
*/

int main (int argc, char** argv){
    if (argc == 1){
        printf("El programa no contiene argumentos.\n");
        return 88;
    }

    printf("Los elementos del arreglo argv son:\n");
    for (int cont = 0 ; cont < argc ; cont++ ){
        printf("argv[%d] = %s\n", cont, argv[cont]);
    }

    return 88;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 193/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

Estático

Lenguaje C permite definir elementos estáticos. La sintaxis para declarar elementos estáticos es la siguiente:

```
static tipoDato nombre;
static valorRetorno 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 las mismas.

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.

Código (variable estática)

```
#include <stdio.h>

/*
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.
*/

void llamarFuncion();

int main (){
    for (int j=0 ; j < 5 ; j++){
        llamarFuncion();
    }
}
```

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 194/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

```

void llamarFuncion(){
    static int numVeces = 0;
    printf("Esta función se ha llamado %d veces.\n",++numVeces);
}

```

Una vez declarada una variable estática, esta permanece en memoria a lo largo de la ejecución del programa, por lo tanto, la segunda vez que se llama a la función ya no se vuelve a crear la variable, si no que se utiliza la que está en la memoria y por eso conserva su valor.

Código (función estática)

Este ejemplo consta de dos archivos: funcEstatica.c y calculadora.c.

```

//##### funcEstatica.c #####
#include <stdio.h>

/*
Este programa contiene las funciones de una calculadora básica: suma, resta, producto y
cociente.
*/

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: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 195/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

```

//##### calculadora.c #####
#include <stdio.h>

/*
Este programa contiene el método principal, el cual invoca a las funciones
del archivo funcEstatica.c.
*/

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 (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: | 01 |
| | | Página | 196/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 197/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 13: Lectura y escritura de datos



Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado por:

Ing. Laura Sandoval Montaño

Autorizado por:

M.C. Alejandro Velázquez Mena

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 198/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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 13: Lectura y escritura de datos

Objetivo:

Elaborar 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.

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: | 01 |
| | | Página | 199/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | 01 |
| | | Página | 200/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 201/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| 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)

```
#include<stdio.h>

/*
Este programa permite abrir un archivo en modo de lectura, de ser posible.
*/

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;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 202/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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);
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)

```
#include<stdio.h>

/*
Este programa permite leer el contenido de un archivo, de ser posible, a
través de la función fgets.
*/

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: Versión: Página: Sección ISO Fecha de emisión | MADO-17 01 203/207 8.3 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

Código (fputs)

```
#include<stdio.h>

/*
Este programa permite escribir una cadena dentro de un archivo, de ser
posible, a través de la función fputs.
*/

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 exactamente como printf() (imprimir) y scanf() (leer), excepto que operan sobre archivo. Sus estructuras son:

```
int fprintf(FILE *apArch, char *formato, ...);
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.

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 204/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

Código (fscanf)

```
#include<stdio.h>
/*
Este programa permite leer el contenido de un archivo,
de ser posible, a través de la función fscanf.
*/
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;
}
```

Código (fprintf)

```
#include<stdio.h>
/*
Este programa permite escribir dentro de un archivo,
de ser posible, a través de la función fprintf.
*/
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 /
escriutura.\n");
    }
    return 0;
}
```

| | | | |
|---|---|--|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 205/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | | |

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.

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)

```
#include <stdio.h>

/*
Este programa muestra el contenido de un archivo de texto. El
nombre del archivo se recibe como argumento de la
función principal.
*/

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;
    }
}
```

| | | | |
|---|---|------------------|---------------------|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: | MADO-17 |
| | | Versión: | 01 |
| | | Página | 206/207 |
| | | Sección ISO | 8.3 |
| | | Fecha de emisión | 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | | |
| La impresión de este documento es una copia no controlada | | | |

```
// 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;
}
```

Código (fwrite)

```
#include <stdio.h>

/*
Este programa realizar una copia exacta de dos archivos. Los
nombres de los archivos (origen y destino) se reciben como
argumentos de la función principal.
*/

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");
```

| | | |
|---|---|---|
|  | Manual de prácticas del Laboratorio de Fundamentos de programación | Código: MADO-17 Versión: 01 Página 207/207 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017 |
| Facultad de Ingeniería | Área/Departamento: Laboratorio de computación salas A y B | |
| La impresión de este documento es una copia no controlada | | |

```

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))
    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 Educación 1991.