

# Guia de clase

# Informática I

---

Ing. Jerónimo F. Atencio

Versión 14

# Índice

<b>Índice</b>	<b>1</b>
<b>Acerca de esta guía</b>	<b>6</b>
Modalidad de evaluación	7
Condiciones para la firma de la materia.	7
Condiciones para la promoción de la materia.	7
Realización de los parciales y recuperatorios	7
Entrega de trabajos prácticos	8
<b>1.Sistemas operativos</b>	<b>9</b>
Instalación de Lubuntu en una máquina virtual	9
<b>2. Comandos básicos de la terminal</b>	<b>20</b>
a. date: Fecha y hora	20
b. cal: Calendario	20
c. man: Manuales	21
d. pwd: Directorio actual	21
e. ls: Listar	22
f. mkdir	22
g. cd	22
h. touch	22
i. rm	22
j. rmdir	22
k. clear	23
<b>3. Instalando paquetes</b>	<b>24</b>
<b>4. Instalando herramientas de desarrollo y man pages</b>	<b>25</b>
Instalando guest addition en virtualbox	25
<b>5. Mi primer programa en C</b>	<b>28</b>
<b>6. Accediendo al repositorio</b>	<b>30</b>
Verificando los repositorios desde el navegador.	30
Configurando Git por primera vez	31
Clonando el repositorio de material.	32
Actualizando el la versión local del repositorio de material desde uno remoto (pull)	32
Clonando su repositorio personal.	33
Agregando o modificando un archivo a su repositorio. (commit, push)	34
Resumen	34
<b>7. Sistemas de numeración</b>	<b>35</b>
Clasificación de sistemas de numeración	35
● No posicionales	35
● Posicionales	35
Sistema de numeración decimal	36
Sistema de numeración binario	36

Sistema de numeración octal	36
Sistema de numeración hexadecimal	36
Símbolos para los sistemas de numeración base 2, 8, 10 y 16	37
Cambio de base decimal a binario, octal o hexadecimal	37
Cambio de base de octal a binario y binario a octal	39
Cambio de base hexadecimal a binario o de binario a hexadecimal	39
Cambio de base octal a hexadecimal o hexadecimal a octal	39
Unidad de información	40
<b>Ejercicios</b>	<b>40</b>
<b>8. Ingreso de datos e impresión en pantalla</b>	<b>41</b>
Tipos de datos	41
Secuencias de escape	41
Especificadores de formato	41
Tabla ASCII	42
Funciones utilizadas	42
Cómo obtener el manual de una función de biblioteca	42
Ejemplos	42
Ejercicios	43
<b>9. Operaciones aritmética - casteo.</b>	<b>45</b>
Operadores aritméticos	45
Funciones utilizadas	45
Ejemplos	45
Ejercicios	46
<b>10. Sentencias condicionales if y switch-case</b>	<b>48</b>
Operadores relacionales	48
Operadores lógicos	48
Sentencias utilizadas	48
Ejemplos	48
Ejercicios	53
<b>11. Sentencias de repetición for; while; do-while</b>	<b>55</b>
Operadores asignación, incremento y decremento	55
Sentencias utilizadas	55
Ejemplos	55
Ejercicios	57
<b>12. Directiva de precompilador define</b>	<b>59</b>
Directivas de precompilación	59
Ejemplos	59
Ejercicios	60
<b>13. Funciones</b>	<b>61</b>
Constantes simbólicas	61
Ejemplos	61
Ejercicios	64

<b>14. Vectores y strings</b>	<b>65</b>
Ejemplos	65
Ejercicios	68
<b>15. Punteros</b>	<b>70</b>
Forma básica de usar punteros	70
Ejemplos	71
Ejercicios	74
<b>16. Asignación dinámica de memoria</b>	<b>76</b>
Funciones utilizadas de stdlib.h	76
Ejemplos	76
Ejercicios	78
<b>17. Algoritmos integradores</b>	<b>79</b>
Funciones utilizadas de string.h	79
Funciones utilizadas de stdlib.h	79
Ejemplos	79
Ejercicios	83
<b>18. Modelo de memoria</b>	<b>88</b>
Tipos de datos (para 64 bits)	88
Ámbito de uso de una variable. (scope de una variable)	88
Mapa de memoria	90
Modificadores de variables	90
Direcciones de memoria	91
Standard streams	94
<b>19. Estructuras</b>	<b>96</b>
Operador	96
Ejemplos	96
Ejercicios	107
<b>20. Archivos I</b>	<b>110</b>
Funciones utilizadas de stdio.h	110
Modos de apertura de un archivo	110
Programas necesarios	110
Comandos	111
Ejemplo de uso de comandos.	111
Ejemplos	114
Ejercicios	121
<b>21. Archivos II</b>	<b>122</b>
Funciones utilizadas de stdio.h	122
Funciones utilizadas de sys/stat.h	122
Comandos	122
Ejemplo de uso de comandos.	122
Cada columna de lo devuelto por ls -l es:	123

Ejemplo de uso de uso de chmod, le damos al archivo permisos de lectura, escritura y ejecución para el propietario, el grupo y otros.	123
Ejemplos	124
Ejercicios	129
<b>22. Punteros, el regreso!</b>	<b>131</b>
Ejemplos	132
Ejercicios	139
<b>23. Matrices.</b>	<b>141</b>
Ejemplos	141
Ejercicios	143
<b>24. Operaciones a nivel de bits, campos de bits. Uniones. Enum</b>	<b>145</b>
Operadores a nivel de bits	145
Ejemplos	145
Ejercicios	152
<b>25. Recursividad.</b>	<b>153</b>
Ejercicios	154
<b>26. Uso de makefile: compilación y linkeo.</b>	<b>155</b>
Programas necesarios	155
Pasos de la compilación	155
Compilación de varios .c y .h	156
Makefile	158
Como instalar debugger en Atom	159
Cómo debuggear un programa en C	160
Library	162
Static library	162
Dynamic library	163
<b>27. Documentación del código: Doxygen e indentación</b>	<b>165</b>
Programas necesarios	165
Doxygen	165
Documentando con doxygen	165
Ejemplo de documentación de archivo con función main.	166
Ejemplo de documentación de archivo con funciones varias.	167
Ejemplo de documentación de archivo .h	168
<b>28. Lista simple enlazadas I</b>	<b>169</b>
Insertar un nodo al inicio (pila)	169
Imprime todos los nodos	170
Libera todos los nodos	170
Cuenta la cantidad de nodos de la lista simple enlazada	171
Inserta un nodo al final	171
Busca un nodo	171
Remueve un nodo	172
Inserta un nodo de forma ordenada	173



Función main que demuestra el uso de las funciones anteriormente descritas.	174
Ejercicios	175
<b>29. Signals</b>	<b>177</b>
Funciones utilizadas	177
Comandos	177
Programa, proceso	177
Ejemplos	178
Ejercicios	184
<b>30. Threads</b>	<b>185</b>
Funciones utilizadas	185
Ejemplos	185
<b>31. Sockets</b>	<b>189</b>
Funciones utilizadas	189
Ejemplos	189
Ejercicios	190
<b>32. Interfaces visuales: Qt</b>	<b>191</b>
Programas necesarios	191
Ejecución	191



## Acerca de esta guía

Este documento intenta guiarlo en el aprendizaje de lenguaje C sobre la plataforma Linux, abarcando la instalación del sistema operativo y las herramientas de desarrollo, para luego comenzar a realizar los primeros programas en C.

Cada apartado de programación comienza mostrando información relevante para llevar adelante el apartado, para luego continuar con algunos ejemplos y terminar con ejercicios propuestos para realizar. Es recomendable seguir el orden de la guía y los ejercicios ya que la dificultad de los mismos va en incremento. Los ejemplos son muy básicos, pero sirven para mostrar algún aspecto del lenguaje y/o como base para la resolución de los ejercicios, por lo cual es recomendable transcribirlos y probarlos.

Se recomienda que los programas de ejemplo los transcriba evitando copiar y pegar para ir tomando práctica en la sintaxis del lenguaje. Trate de leer los manuales (use el comando `man`) de las funciones , sentencias y comandos utilizados para interiorizarse en su uso más allá de la aplicación particular que se haga en esta guía.

## Modalidad de evaluación

La evaluación de la materia se realizará con dos parciales y cada uno tendrá dos instancias de recuperación.

### Condiciones para la firma de la materia.

1. Obtener 6 o más puntos en el primer parcial. Se puede recuperar en diciembre y/o en febrero.
2. Obtener 6 o más puntos en el segundo parcial. Se puede recuperar en diciembre y/o en febrero.
3. Tener aprobado el 75% de los trabajos prácticos.

### Condiciones para la promoción de la materia.

1. Obtener 7 o más puntos en el primer parcial.
2. Obtener 8 o más puntos en el primer parcial.
3. Tener aprobado el 75% de los trabajos prácticos.

En este caso si alguno de los parciales no alcanza la condición para promoción, se podrá recuperar sólo uno en la fecha de diciembre. La nota que queda es la de la última evaluación realizada.

Ejemplos:

Parciales		Recuperatorio diciembre		Recuperatorio febrero		Trabajo práctico aprobado	Resultado
Primer parcial	Segundo parcial	Primer parcial	Segundo parcial	Primer parcial	Segundo parcial		
7	8	-	-	-	-	Si	Promociona
6	8	7	-	-	-	Si	Promociona
6	8	6	-	-	-	Si	Firma
7	7	-	5	-	6	Si	Firma
10	7	-	-	-	-	Si	Firma
10	7	-	7	-	8	Si	Firma
7	7	-	4	-	2	Si	Recurso
10	10	-	-	-	-	No	Recurso

## Realización de los parciales y recuperatorios

Las fechas de los parciales y los recuperatorios las podrá encontrar en la planificación. En cada parcial entran todos los temas vistos hasta la clase anterior dicho parcial. Ambos parciales serán presenciales así como también los recuperatorios.



## Entrega de trabajos prácticos

En la planificación encontrará las fechas límites para la entrega de los ejercicios de cada capítulo y cuales son entregables.

Todos los ejercicios deberán ser subidos al repositorio en una carpeta con la siguiente convención de nombres ejercicios\_AA donde

- AA: Es el número de capítulo

Por ejemplo: para el capítulo 8 la carpeta se llamara se llamará ejercicios\_08

Al realizar los ejemplos salvo que indique lo contrario utilizaremos la siguiente convención de nombres ejemplo\_AA\_BB.c donde

- AA: Es el número de capítulo
- BB: Es el número de ejemplo

Por ejemplo: el ejemplo 1 del capítulo 8 se llamará ejemplo08\_01.c

Al realizar los ejemplos salvo que indique lo contrario utilizaremos la siguiente convención de nombres ejercicio\_AA\_BB.c donde

- AA: Es el número de capítulo
- BB: Es el número de ejercicio

Por ejemplo: el ejercicio 10 del capítulo 9 se llamará ejercicios 09\_10.c



# 1.Sistemas operativos

El sistema operativo es el software que gestiona los recursos de hardware y software además de proveer una interfaz para los programas que se ejecutarán en él. Existen numerosos sistema operativos entre los cuales podemos mencionar:

- Linux
- Windows
- MacOS
- Unix

En esta guía utilizaremos Linux como sistema operativo, el cual posee varias distribuciones (distro), como por ejemplo:

- Debian (<https://www.debian.org>)
- Ubuntu (<https://ubuntu.com>)
- Lubuntu (<https://lubuntu.net>)
- Mint (<https://linuxmint.com>)
- Fedora (<https://getfedora.org>)
- Arch (<https://archlinux.org/>)

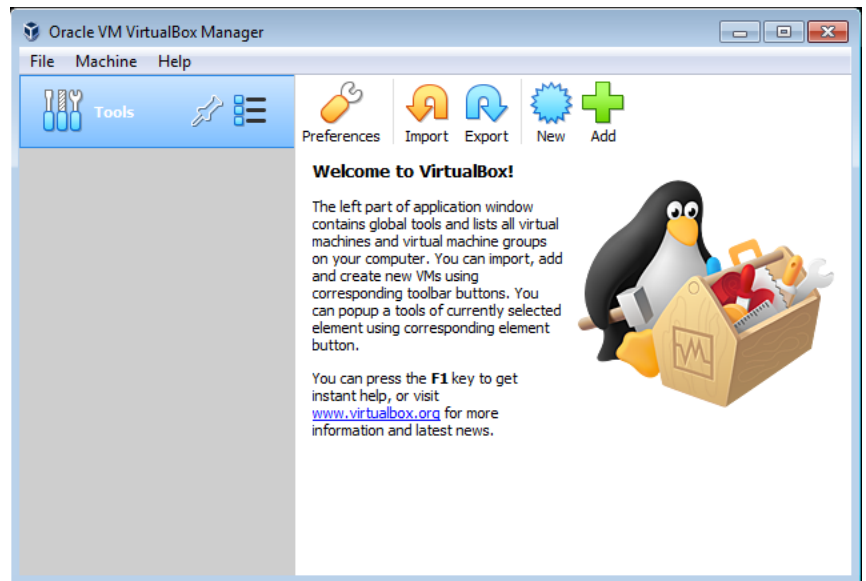
Una distribución utiliza el Kernel de linux (<https://www.kernel.org/>) y está compuesta por software adicional como por ejemplo un editor de texto, software para reproducir música o video, browser, un manejador de archivos, etc. Además de documentación, las herramientas GNU, un gestor de paquetes y sus bibliotecas junto con alguna interfaz de usuario que puede ser una manejador de ventanas o una terminal. La elección de la distribución suele depender del uso que se dará y de los gustos del usuario, en el caso de esta guía utilizaremos Lubuntu.

## Instalación de Lubuntu en una máquina virtual

Si no tiene experiencia en instalación de sistemas operativos, instalarlo en una máquina virtual es una forma sencilla de aprender a hacerlo y segura, dado que ante cualquier error podría borrar la imagen y volver a intentarlo sin afectar sus sistema operativo nativo. Cuando adquiera alguna experiencia en Linux es conveniente que lo instale de forma nativa en su computadora. Los siguientes pasos asumen que su sistema nativo es Windows.

- a. Descargar una versión LTS (Long Term Support) de la distribución Lubuntu (<https://lubuntu.net>)
- b. Descargar el software VirtualBox (<https://www.virtualbox.org/>) que es el que usaremos para hacer la máquina virtual. Debe descargar el que dice Windows Host.
- c. Instale el VirtualBox, es recomendable reiniciar la computadora al finalizar la instalación.

- d. Una vez abierto el VirtualBox, cree una nueva máquina virtual presionando en el botón **New**



- e. En la siguiente ventana se configura lo siguiente
- El nombre de la máquina virtual
  - La carpeta donde estará el disco virtual (Dejamos la que está configurada)
  - El tipo de sistema operativo, seleccionamos linux
  - La versión (distribución) del sistema operativo

← Create Virtual Machine

### Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name:

Machine Folder:

Type:

Version:

- f. Se selecciona la memoria que se reserva para nuestra máquina virtual, inicialmente se puede dejar el valor propuesto para la instalación, en el caso de que funcione lento se puede luego asignarle más. Se debe tener en cuenta que la memoria que reservamos para la máquina virtual es memoria que la máquina física no usará.

?

×

← Create Virtual Machine

Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024 MB**.

4 MB

8192 MB

1024 MB

Next

Cancel

- g. Se le secciona la opción para usar un disco rígido virtual nuevo

?

×

← Create Virtual Machine

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

☐ Do not add a virtual hard disk

☒ Create a virtual hard disk now

☐ Use an existing virtual hard disk file

Create

Cancel

- h. Selección del tipo de disco rígido virtual.

? ×

← Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

☒ VDI (VirtualBox Disk Image)

☐ VHD (Virtual Hard Disk)

☐ VMDK (Virtual Machine Disk)

Expert Mode

Next

Cancel

- i. Se selecciona la opción para que cree el disco rígido virtual de tamaño fijo.

? ×

← Create Virtual Hard Disk

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

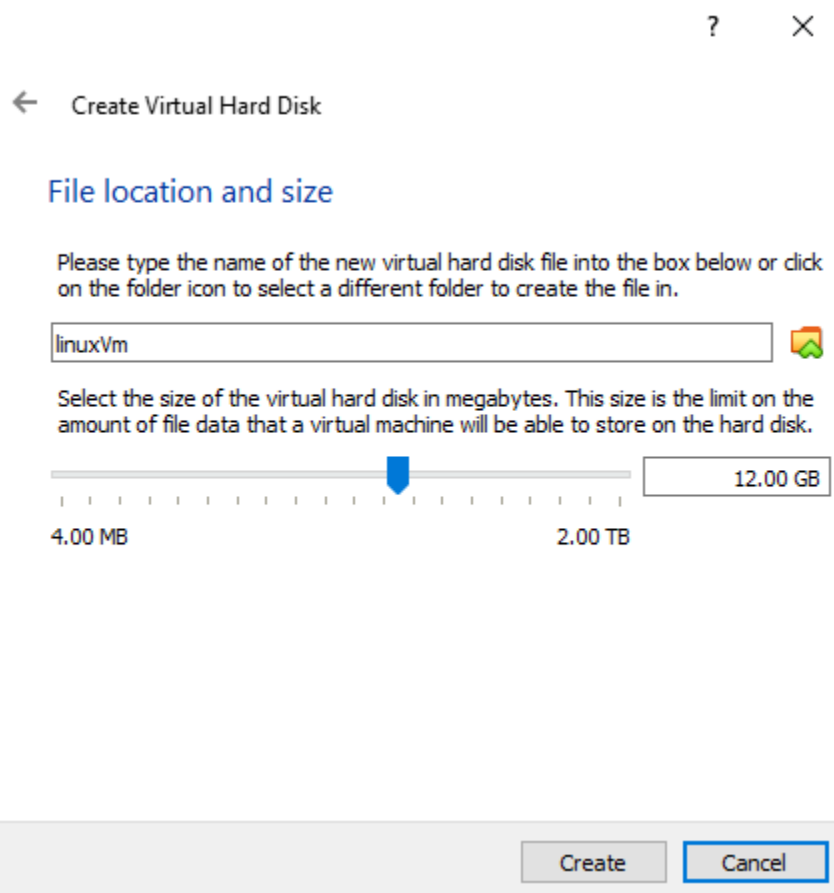
☐ Dynamically allocated

☒ Fixed size

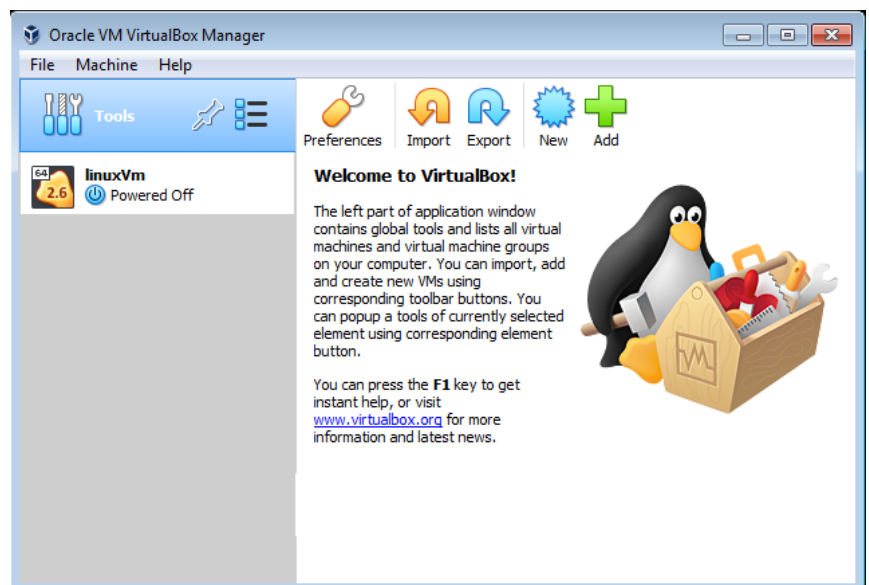
Next

Cancel

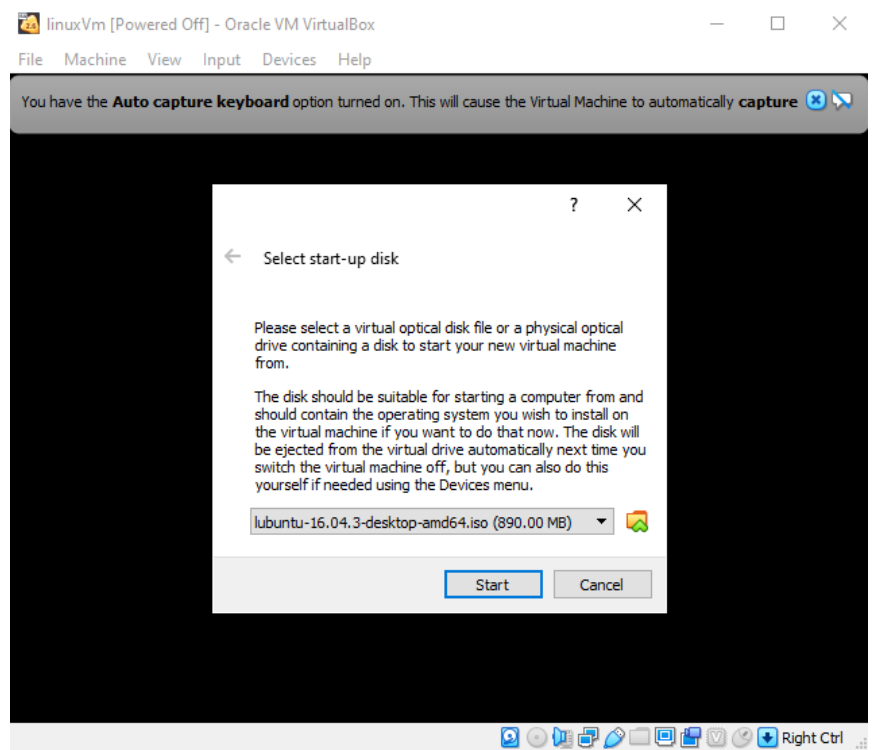
- j. Se selecciona el tamaño del disco rígido virtual, con al menos 12GB debería alcanzar para una instalación básica.



- k. Al finalizar la creación del disco rígido virtual se llega a una pantalla como esta. Se selecciona la máquina virtual creada y se le da start.

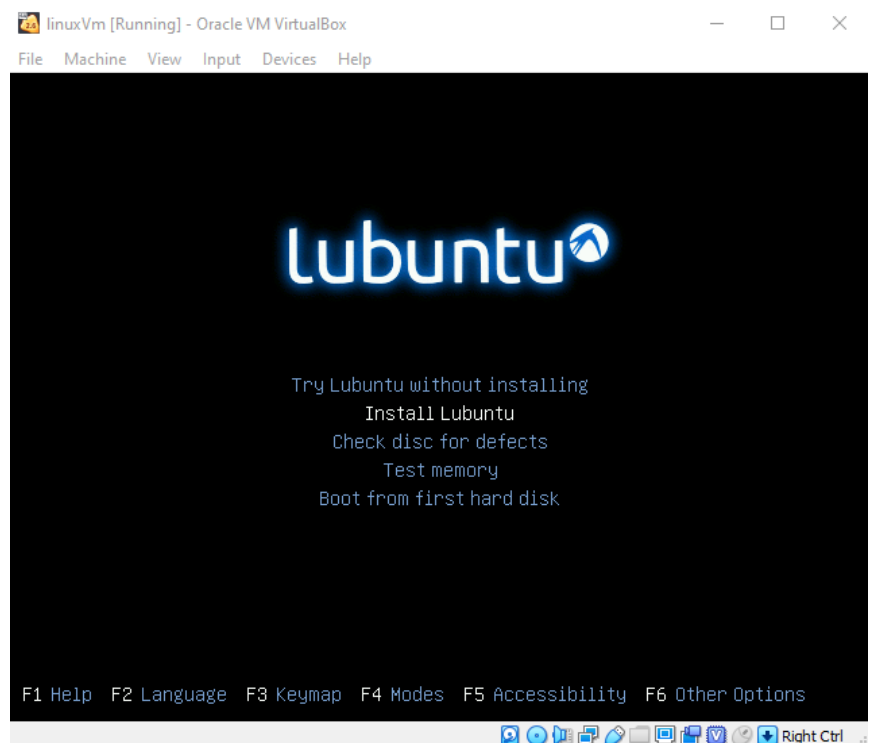


- I. Se selecciona la imagen de Linux que se desea instalar. La imagen a instalar suele tener extensión .iso

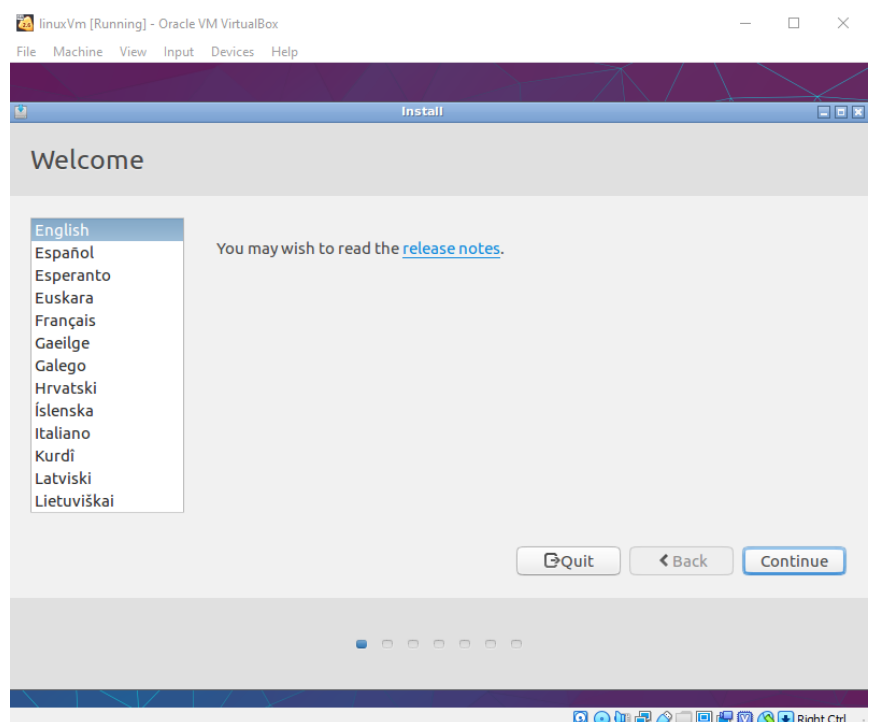


A partir de este punto se realiza la instalación del sistema operativo

- m. Una vez que inicio el instalador, se selecciona la opción **Install Lubuntu**

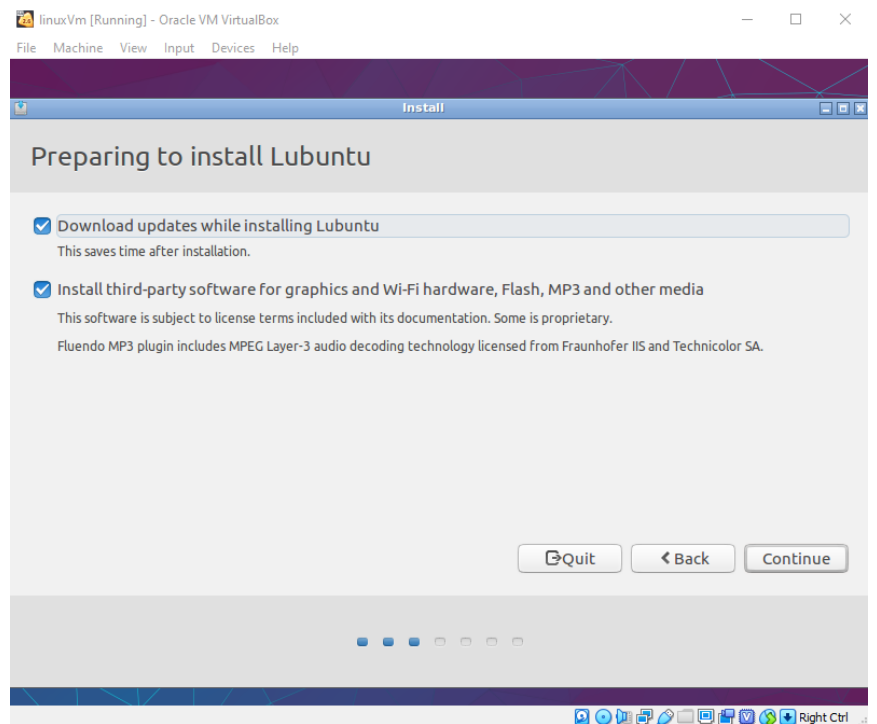


- n. Se selecciona el idioma para la instalación.

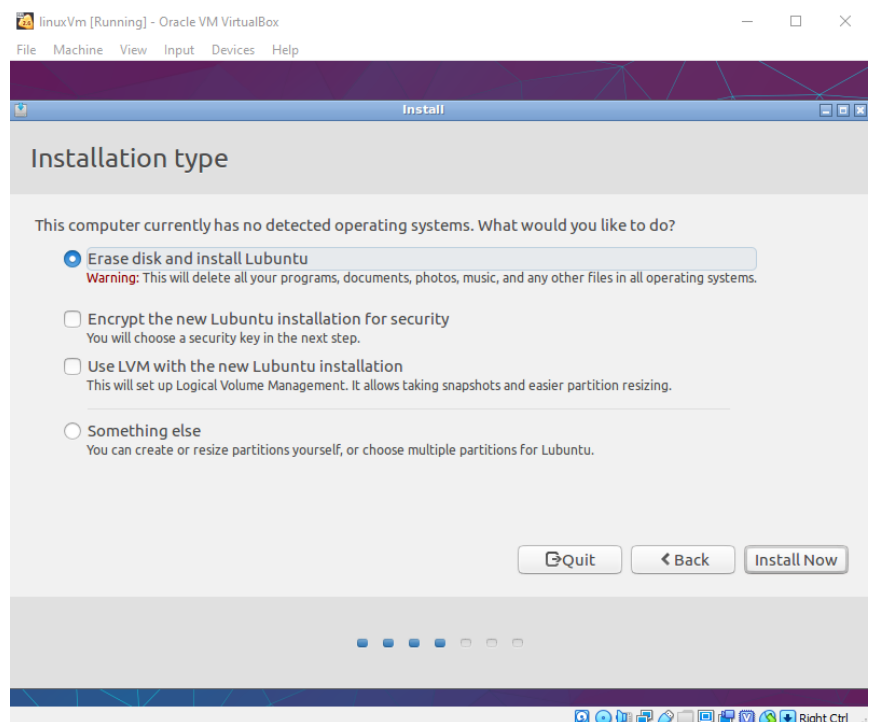




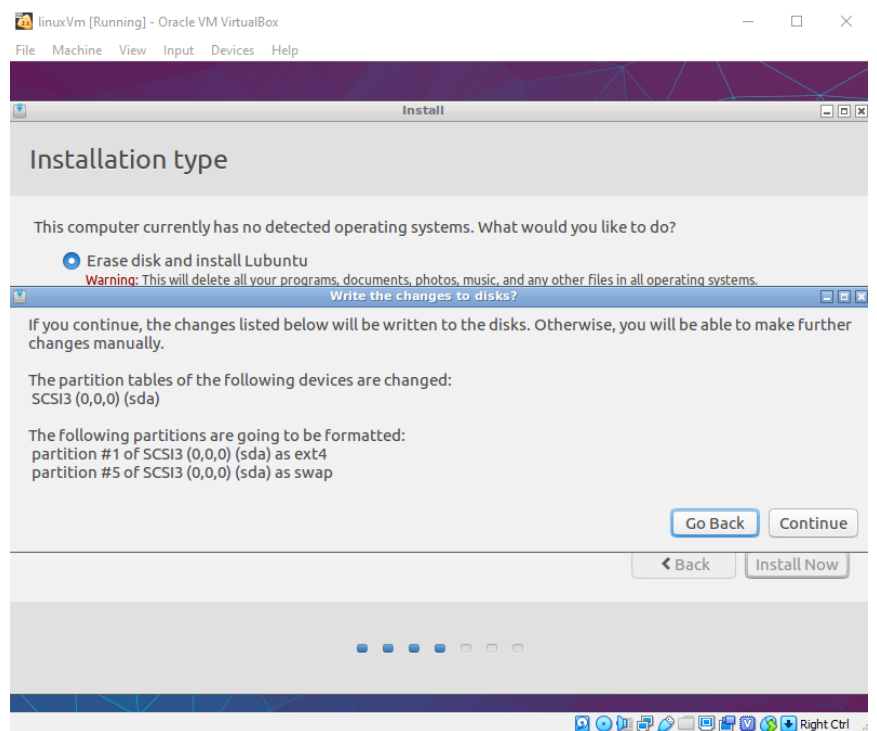
- o. La primera opción seleccionada es para que las actualizaciones se descarguen durante la instalación. La segunda opción es para permitir que se instale software de terceros.



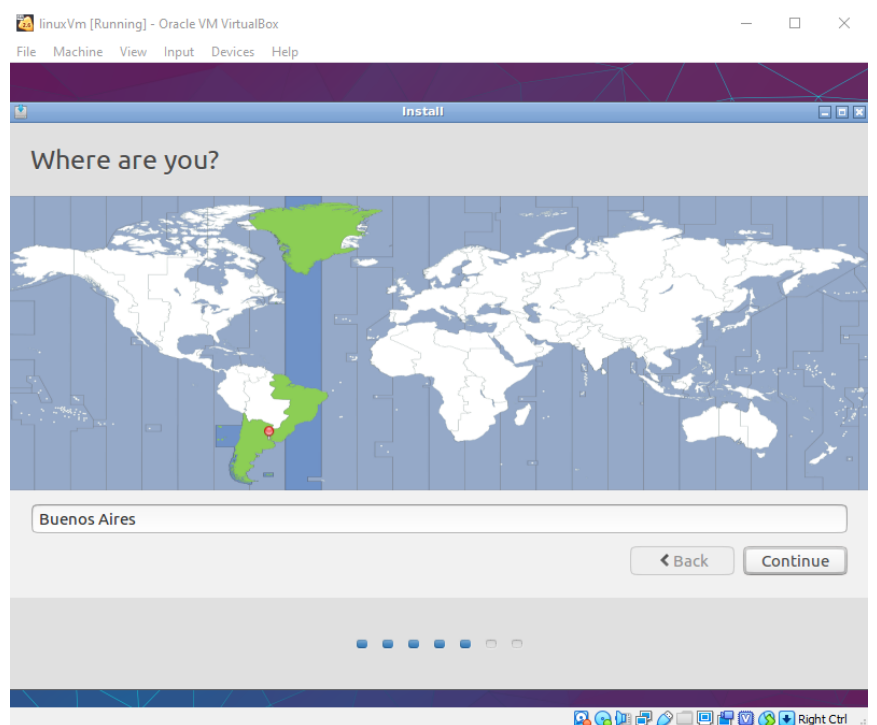
- p. Dado que el disco rígido virtual no tiene nada instalados, se selecciona la opción de borrar el disco entero e instalar Ubuntu



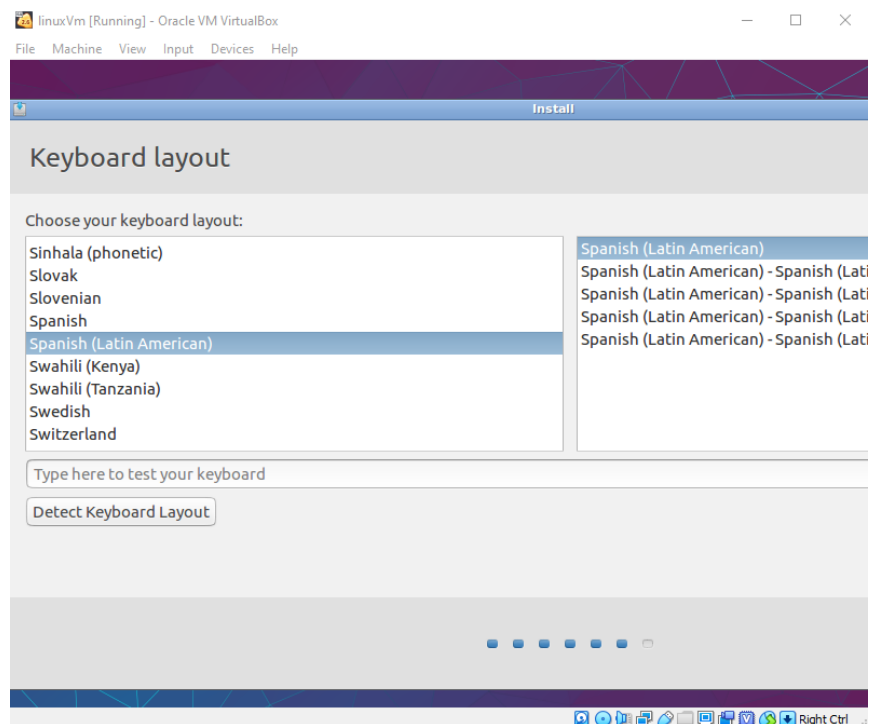
- q. Solicita confirmación acerca de los cambios a realizar en el disco rígido virtual.



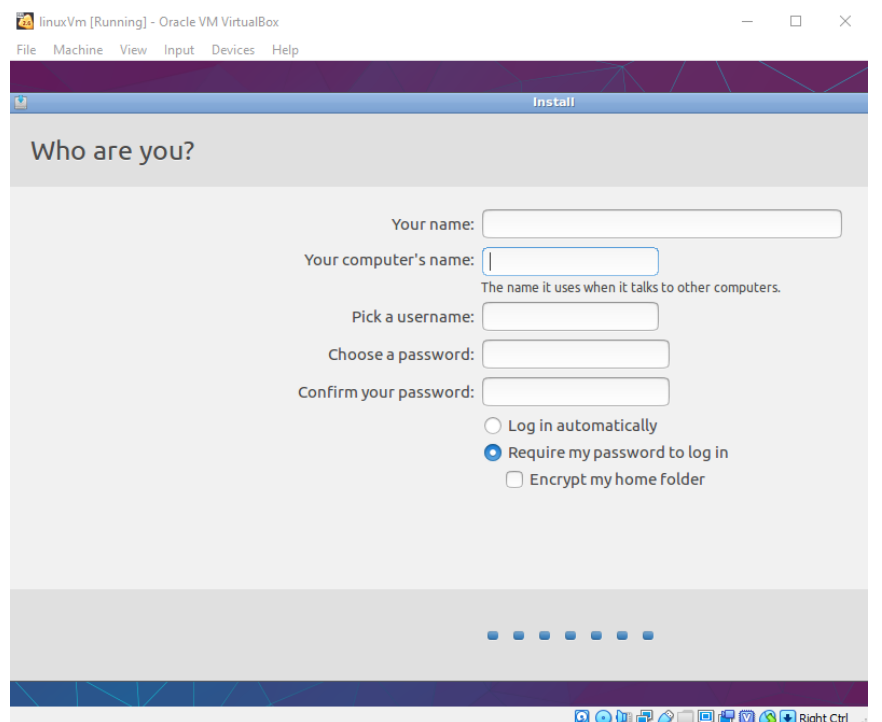
- r. Se indica el lugar donde estamos para las configuraciones regionales.



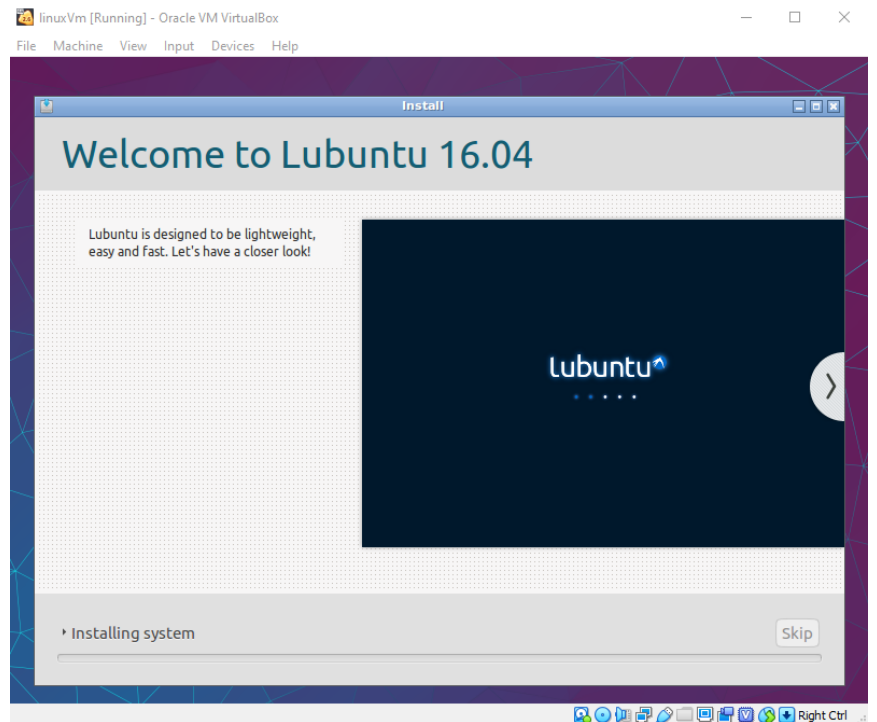
- s. Se configura el layout del teclado.



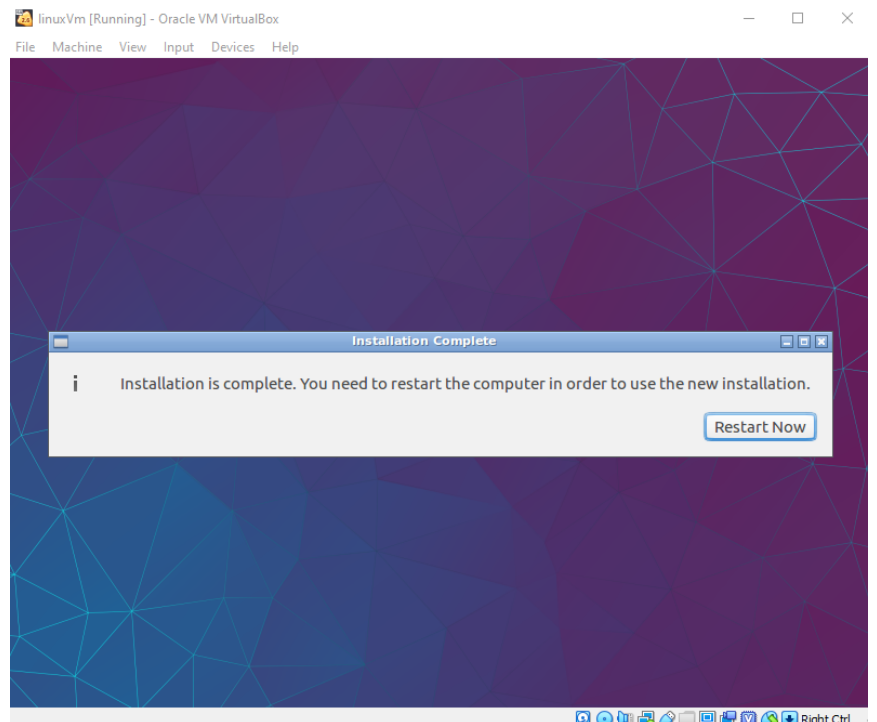
- t. Configurar el nombre de usuario, nombre de la computadora, passwords (Tratar de no olvidar los passwords)



- u. Esperamos a que transcurra la instalación.



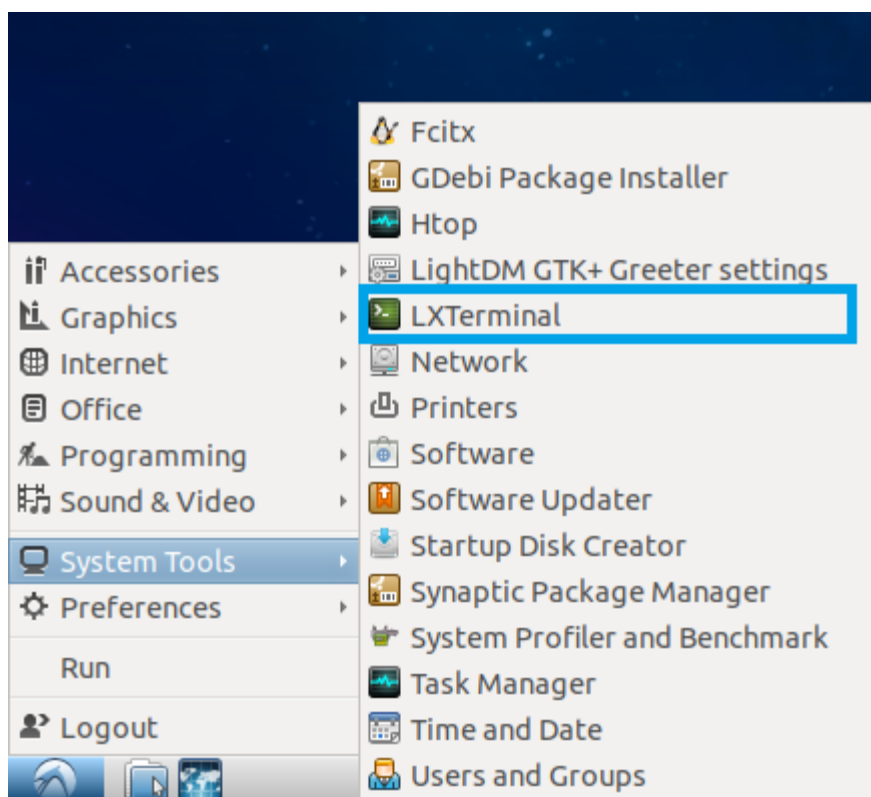
- v. Una vez finalizada la instalación, reiniciamos la computadora.



Advertencia:  
Recuerde el password y usuario configurados.

## 2. Comandos básicos de la terminal

En esta sección se exploran algunos comandos útiles de la terminal. Para abrir la terminal puede usar el shortcut CTRL+ALT+T o el menú que se observa en la siguiente imagen.



### a. date: Fecha y hora

Con la opción -a Muestra todos los manuales que contienen intro

```
jerome@linuxVm:~$ date
dom mar 21 21:45:40 -03 2021
```

### b. cal: Calendario

Muestra el calendario del mes.

```
jerome@linuxVm:~$ cal
Marzo 2021
do lu ma mi ju vi sá
  1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

Muestra el calendario de un mes y un año particular.

```
jerome@linuxVm:~$ cal -m 12 2001
    Diciembre 2001
do lu ma mi ju vi sá
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

### c. man: Manuales

Muestra el manual de un comando o función que es pasado como parámetro. El manual dispone de siete secciones que se describen a continuación.

1. Programas ejecutables o comandos del shell
2. Llamadas del sistema (Funciones provistas por el kernel)
3. Llamadas a biblioteca.
4. Archivos especiales. (Los encontrados en /dev)
5. Formatos y convenciones de archivos.
6. Juegos
7. Misceláneos
8. Comandos de administración de sistema
9. Rutinas de kernel

Para obtener el manual del manual, una vez dentro del manual si desea salir deberá presionar la letra **q**

```
jerome@linuxVm:~$ man man
```

Para obtener el manual date

```
jerome@linuxVm:~$ man date
```

Para buscar en el manual 1 date

```
jerome@linuxVm:~$ man 1 date
```

Con la opción -a Muestra todos los manuales que contienen intro

```
jerome@linuxVm:~$ man -a intro
```

### d. pwd: Directorio actual

Imprime el nombre del directorio actual

```
jerome@linuxVm:~$ pwd
/home/jerome
```

## e. ls: Listar

Lista el contenido de un directorio

```
jerome@linuxVm:~$ ls
Desktop Documents Music Pictures Public Templates Videos
```

Lista el contenido de un directorio incluyendo la fecha y hora de última modificación entre otros datos.

```
jerome@linuxVm:~$ ls -l
total 32
drwxr-xr-x 2 info1 info1 4096 abr  6  2020 Desktop
drwxr-xr-x 4 info1 info1 4096 mar 24 00:18 Documents
drwxr-xr-x 2 info1 info1 4096 abr  6  2020 Music
drwxr-xr-x 2 info1 info1 4096 abr  6  2020 Pictures
drwxr-xr-x 2 info1 info1 4096 abr  6  2020 Public
drwxr-xr-x 2 info1 info1 4096 abr  6  2020 Templates
drwxr-xr-x 2 info1 info1 4096 abr  6  2020 Videos
```

## f. mkdir

Crea un nuevo directorio. Crea un directorio llamado pruebaDir

```
jerome@linuxVm:~$ mkdir pruebaDir
jerome@linuxVm:~$ ls
Desktop Documents Music Pictures pruebaDir Public Templates
Videos
```

## g. cd

Me permite moverse entre directorios.

Me muevo al directorio con el nombre pruebaDir

```
jerome@linuxVm:~$ cd pruebaDir
jerome@linuxVm:~/pruebaDir$
```

## h. touch

Crea un archivo vacío o actualiza la fecha de último acceso o última modificación.

```
jerome@linuxVm:~/pruebaDir$ touch pruebaArchivo.txt
jerome@linuxVm:~/pruebaDir$ ls
pruebaArchivo.txt
```

## i. rm

Elimina un archivo

```
jerome@linuxVm:~/pruebaDir$ rm pruebaArchivo.txt
```

## j. rmdir

Elimina un directorio.

Antes de borrar un directorio debo salir de él, para ello uso el comando **cd**

```
jerome@linuxVm:~/pruebaDir$ cd ..  
jerome@linuxVm:~$ rmdir pruebaDir
```

#### k. clear

Limpia la pantalla

```
jerome@linuxVm:~/pruebaDir$ clear
```

■ ■ ■



### 3. Instalando paquetes

Cada distribución de linux posee un gestor de paquetes en el caso de Lubuntu es **apt** o **apt-get**. Un gestor de paquetes es un software automatiza el proceso de instalación, actualización, configuración y desinstalación de software en su computadora.

Como ejemplo vamos a instalar VLC, que es un reproductor de audio y video.

```
jerome@linuxVm:~$ apt install vlc
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13:
Permission denied)
E: Unable to acquire the dpkg frontend lock
(/var/lib/dpkg/lock-frontend), are you root?
```

Luego de ejecutar el comando **apt install vlc** nos devuelve un error indicándonos que no tenemos los permisos suficientes, por ello debemos usar **sudo** antes del comando. Luego de escribir **sudo apt install vlc** y ejecutarlo nos pedirá nuestro password. Finalmente nos indicará el espacio necesario para la instalación, luego nos preguntará y deseamos continuar entonces presionamos **Y** Comenzando con la instalación

```
jerome@linuxVm:~$ sudo apt install vlc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  fonts-freefont-ttf vlc-bin vlc-l10n vlc-plugin-notify vlc-plugin-qt
vlc-plugin-samba
  vlc-plugin-skins2 vlc-plugin-video-splitter vlc-plugin-visualization
The following NEW packages will be installed:
  fonts-freefont-ttf vlc vlc-bin vlc-l10n vlc-plugin-notify
vlc-plugin-qt vlc-plugin-samba
  vlc-plugin-skins2 vlc-plugin-video-splitter vlc-plugin-visualization
0 upgraded, 10 newly installed, 0 to remove and 410 not upgraded.
Need to get 11,7 MB of archives.
After this operation, 57,0 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Para verificar que la instalación se realizó correctamente

```
jerome@linuxVm:~$ vlc
```

■ ■ ■

## 4. Instalando herramientas de desarrollo y man pages

Para comenzar instalaremos las herramientas de desarrollo básicas.

- El compilador gcc
- El software de control de versiones git
- El editor de texto atom (<https://flight-manual.atom.io/getting-started/sections/installing-atom/>)

Para realizar la instalación ejecutaremos el siguiente comando, en el incluimos todos los paquetes a instalar y la opción -Y para que no nos pregunte si estamos de acuerdo con la instalación.

```
jerome@linuxVm:~$ sudo apt install gcc git -y
```

Para el caso de la instalación del editor de texto deberemos seguir los siguientes comandos, que aparecen en dos líneas por que no entran en este documento en la misma línea.

```
jerome@linuxVm:~$ wget -qO - https://packagecloud.io/AtomEditor/atom/gpgkey  
| sudo apt-key add -  
  
jerome@linuxVm:~$ sudo sh -c 'echo "deb [arch=amd64]  
https://packagecloud.io/AtomEditor/atom/any/ any main" >  
/etc/apt/sources.list.d/atom.list'  
  
jerome@linuxVm:~$ sudo apt-get update
```

Finalmente ejecutamos

```
jerome@linuxVm:~$ sudo apt install atom -y
```

Para instalar los manuales de desarrollo deberá ejecutar el siguiente comando

```
jerome@linuxVm:~$ sudo apt install manpages-dev manpages-posix-dev -y
```

Si además necesitamos los manuales en español puede ejecutar el siguiente comando

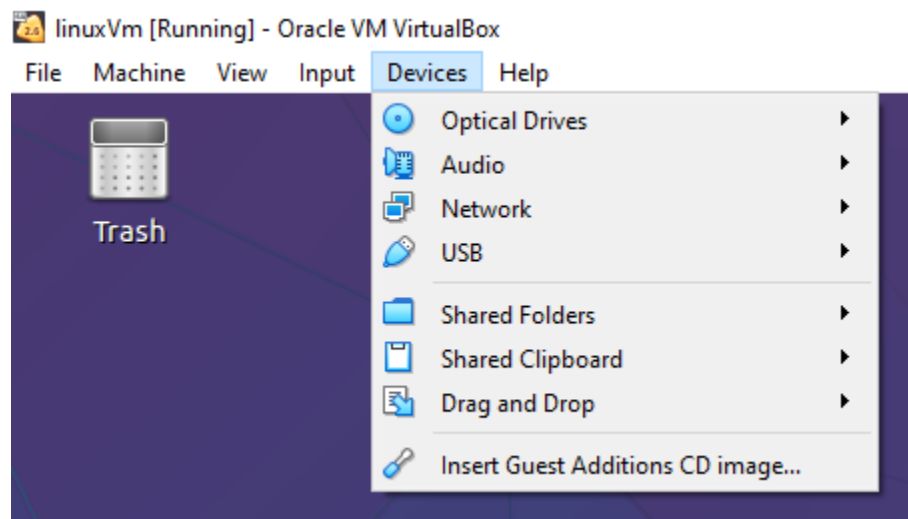
```
jerome@linuxVm:~$ sudo apt install manpages-es manpages-es-extra -y
```

### Instalando guest addition en virtualbox

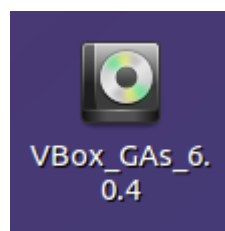
Los guest additions es un paquete de software para agregar algunas funciones a la máquina virtual, como por ejemplo poder usarla en pantalla completa o compartir el portapapeles.

Para instalarlos tenemos que realizar el siguiente procedimiento

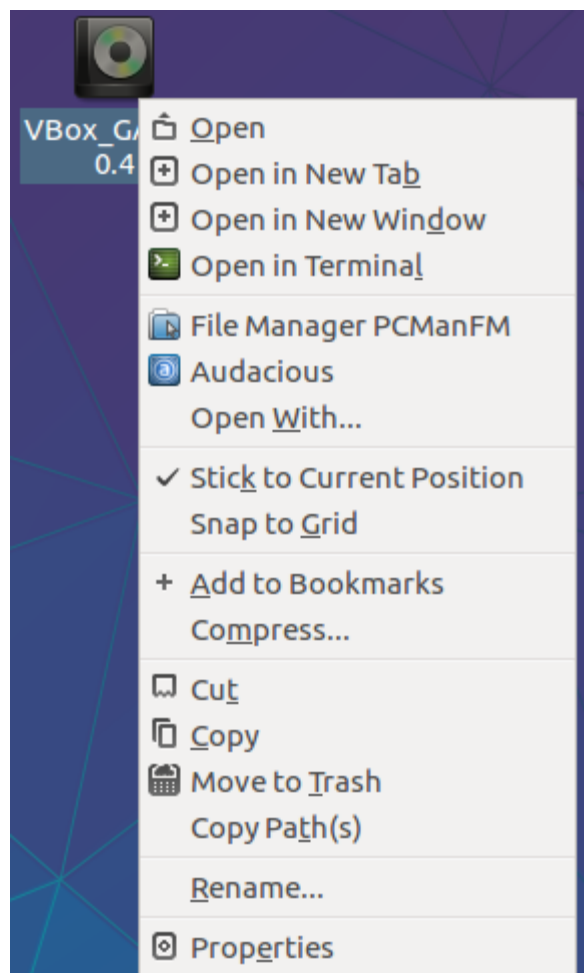
- a. Iniciar la máquina virtual y loguearnos en el sistema operativo.
- b. Ir al siguiente menú y seleccionar **Insert Guest Additions CD image...**



- c. En el escritorio del sistema operativo dentro de la máquina virtual deberá aparecer el siguiente icono.



- d. Seleccione el icono y presione botón derecho. En el menú contextual seleccione Open In terminal (Esto abrirá una terminal nueva)



- e. En la terminal ejecute los siguientes comandos

```
jerome@linuxVm:/media/jerome/VBox_Gas_6.0.4$ sudo apt install  
build-essential  
  
jerome@linuxVm:/media/jerome/VBox_Gas_6.0.4$  
sudo ./VBoxLinuxAdditions.run
```

- f. Reinicie la máquina virtual.

■ ■ ■

## 5. Mi primer programa en C

1. Una vez que inicio Linux abra una terminal (presione CTRL+ALT+T para abrirla)
2. Desde la terminal se crea un directorio llamado **primerPrograma** utilizando el comando mkdir.

```
jerome@linuxVm:~$ mkdir primerPrograma
```

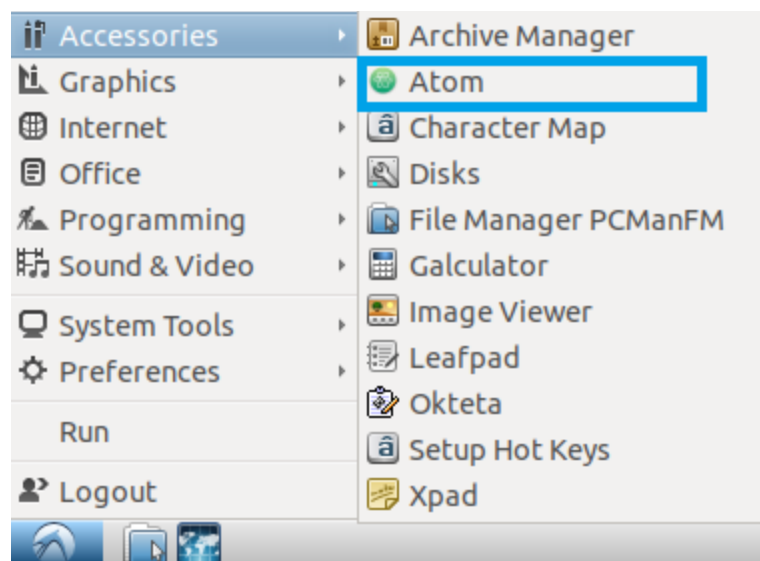
3. Muévase al directorio creado usando el comando cd.

```
jerome@linuxVm:~$ cd primerPrograma
```

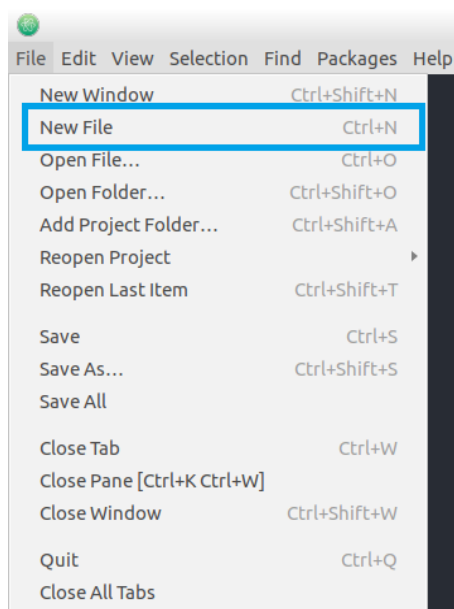
4. Abra el editor de texto atom escribiendo el siguiente comando en la consola atom & (el ampersand es para que el comando se ejecute en background)

```
jerome@linuxVm:~/primerPrograma$ atom &
```

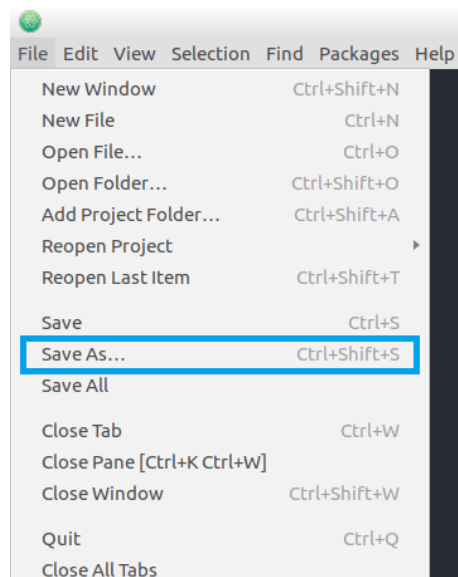
Otra opcion para abrir el atom es usando el menu



5. Cree un nuevo archivo, seleccione el menú File->New File (Archivo->nuevo).



6. Vaya al menú File->Save as.. (Archivo->Guardar como) y guarde el archivo con el siguiente nombre **hola.c** (respete las mayúsculas y minúsculas del nombre) en el directorio **~/primerPrograma/**



7. En el archivo creado copie el siguiente programa y guárdelo.

```
#include <stdio.h>

int main (void)
{
    printf ("Hola!!!\r\n");
    return (0);
}
```

**Advertencia:**  
Verifique que salvo el código.

8. Vuelva a la consola, compile el programa con el siguiente comando.

```
jerome@linuxVm:~/primerPrograma$ gcc hola.c -Wall -ohola.out
```

Eso generará un archivo de salida con el nombre holaMundo.out Verifique su creación listando los archivos del directorio con el comando ls -las

```
jerome@linuxVm:~/primerPrograma$ ls
hola.c    hola.out
```

9. Ejecute el programa compilado, escribiendo en la consola **./hola.out**

```
jerome@linuxVm:~/primerPrograma$ ./hola.out
Hola!!!
```



## 6. Accediendo al repositorio

Se mostrará en sencillos pasos cómo usar git para realizar tareas simples, verificando primero los repositorios desde un navegador y luego usando algunas funciones de git desde la consola.

### Verificando los repositorios desde el navegador.

1. Abra la ventana del navegador (Firefox, Chrome, chromium, etc) y acceda a la dirección <https://gitlab.frba.utn.edu.ar> y loguese con su usuario sinap

LDAP

LDAP Username

Password

☐ Remember me

Sign in

2. Accedera a una página similar a esta donde están al menos dos repositorios
  - a. material: Es el repositorio para descargar el material de la materia
  - b. Un repositorio con su usuario SinAp: Este repositorio lo usarán para subir los trabajos prácticos.

Seleccione el repositorio de material

GitLab

Projects

Groups

More

Search or jump to...

Projects

New project

Your projects 39

Starred projects 2

Explore projects

Filter by name...

Name

J

info1-R1053/2021/jatencio

Owner

★ 1

🔗 0

📄 0

📁 0

Updated 6 minutes ago

M

info1-R1053/Material

Owner

★ 1

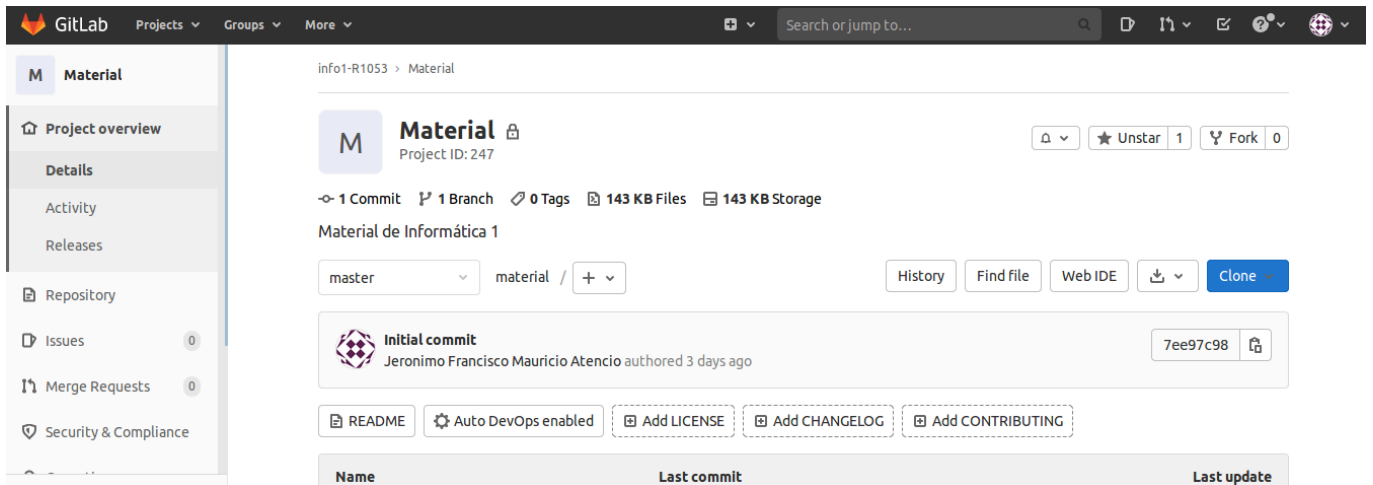
🔗 0

📄 0

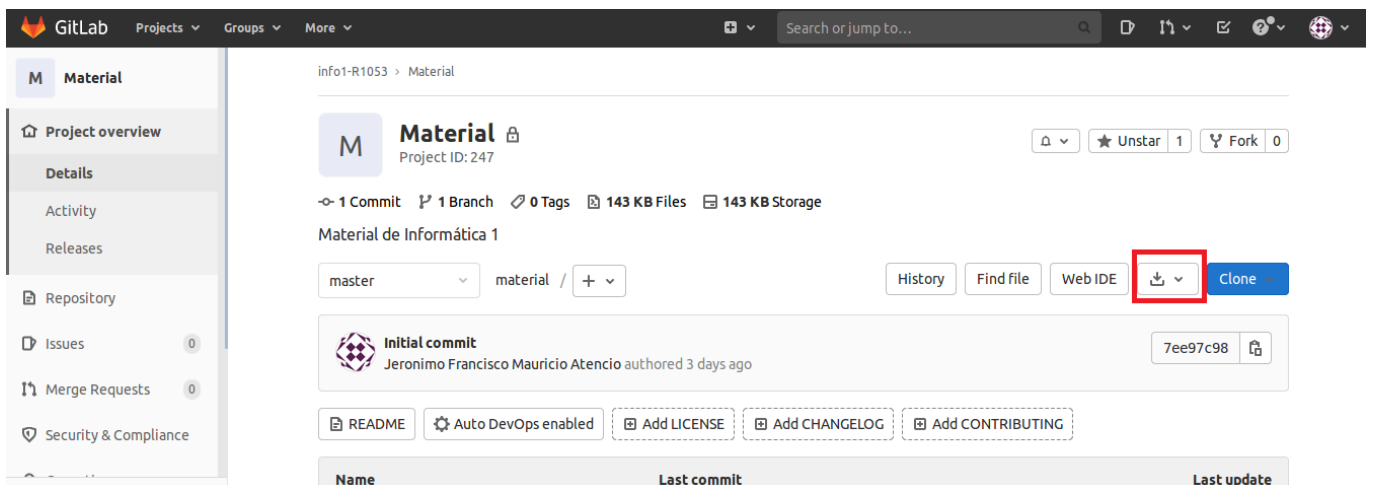
📁 0

Updated 6 minutes ago

### 3. Accedera a una página como la siguiente



### 4. Seleccionando el icono indicado en rojo podrá descargar el contenido del repositorio.



## Configurando Git por primera vez

Antes de comenzar a usar el git deberá ejecutar los siguientes comandos para configurar su usuario e email. Reemplace lo indicado en azul por sus datos

```
jerome@linuxVm:~$ git config --global user.name "usuario"  
jerome@linuxVm:~$ git config --global user.email "email"
```

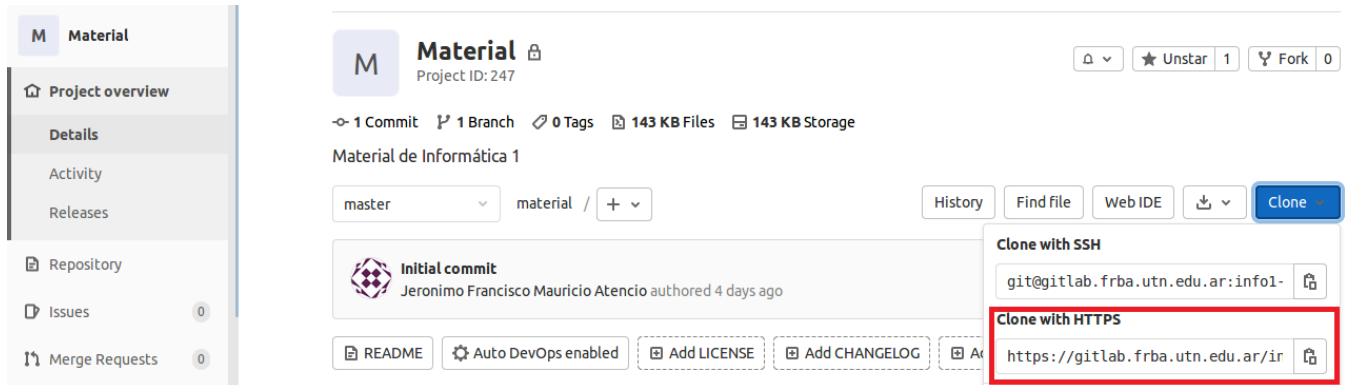
Utilizando el siguiente comando confirme que los datos se configuraron correctamente.

```
jerome@linuxVm:~$ git config --global --list
```



## Clonando el repositorio de material.

1. Obtener la dirección del repositorio de material. Para ello entre en el repositorio a clonar, presione el botón **Clone** y copie la dirección marcada con el recuadro rojo.



2. La dirección obtenida en el punto anterior se coloca luego del comando git clone. Reemplazan lo indicado en azul por la dirección obtenida en el punto anterior.

```
jerome@linuxVm:~$ git clone direccionRepositorio
```

Luego de ejecutar el comando para realizar la clonación, se le solicitará el usuario y el password para acceder al repositorio. Si los datos son correctos se creará un directorio con el nombre del repositorio al cual puede acceder con el comando **cd**.

Por ejemplo para el repositorio de material quedaría

```
jerome@linuxVm:~/jatencio$ git clone
https://gitlab.frba.utn.edu.ar/info1-r1053/material.git
Cloning into 'material'...
Username for 'https://gitlab.frba.utn.edu.ar':
Password for 'https://jatencio@gitlab.frba.utn.edu.ar':
remote: Enumerating objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

## Actualizando el la versión local del repositorio de material desde uno remoto (pull)

1. Ingresar a la carpeta del repositorio de material utilizando el comando **cd**.

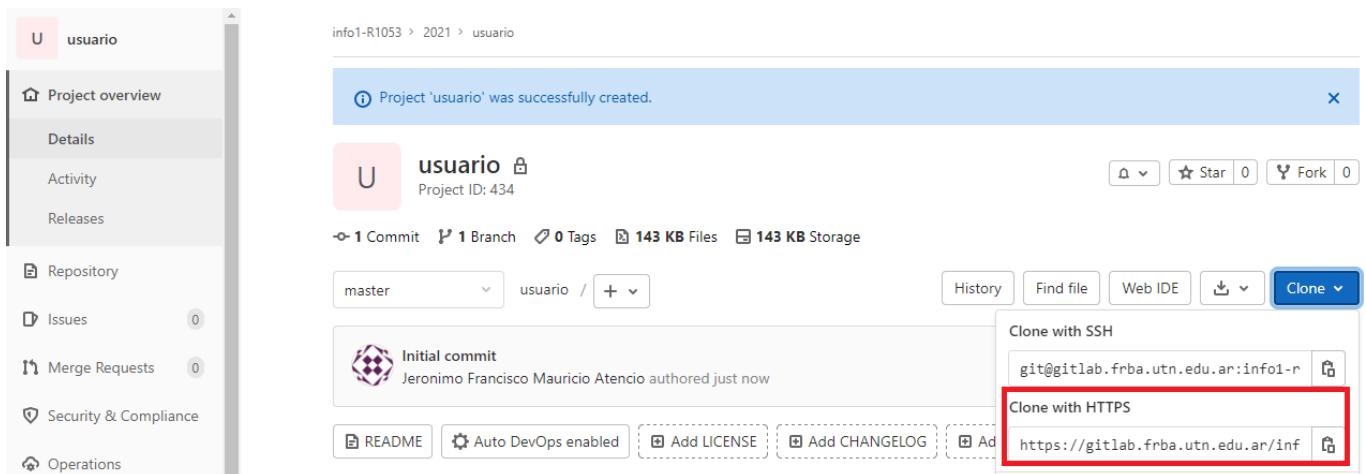
```
jerome@linuxVm:~$ cd material
jerome@linuxVm:~/material$
```

2. Realizamos un pull del repositorio remoto. (Lo que está en azul debe reemplazarlo por su nombre de usuario)

```
jerome@linuxVm:~/material$ git pull
Username for 'https://gitlab.frba.utn.edu.ar': usuario
Password for 'https://usuario@gitlab.frba.utn.edu.ar':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://gitlab.frba.utn.edu.ar/info1-r1053/material
   c6d8a8a..97a0a6a  master       -> origin/master
Updating c6d8a8a..97a0a6a
Fast-forward
 README.md | 5 ++--
 1 file changed, 2 insertions(+), 3 deletions(-)
```

## Clonando su repositorio personal.

3. Obtener la dirección del repositorio de su repositorio personal. Para ello entre en el repositorio a clonar, presione el botón **Clone** y copie la dirección marcada con el recuadro rojo.



4. La dirección obtenida en el punto anterior se coloca luego del comando **git clone**. Reemplazan lo indicado en azul por la dirección obtenida en el punto anterior.

```
jerome@linuxVm:~$ git clone direccionRepositorio
```

Luego de ejecutar el comando para realizar la clonación, se le solicitará el usuario y el password para acceder al repositorio. Si los datos son correctos se creará un directorio con el nombre del repositorio al cual puede acceder con el comando **cd**.

Por ejemplo para el repositorio de material quedaría

```
jerome@linuxVm:~/jatencio$ git clone
https://gitlab.frba.utn.edu.ar/info1-r1053/2021/usuario.git
Cloning into 'usuario'...
Username for 'https://gitlab.frba.utn.edu.ar':
```

```
Password for 'https://usuario@gitlab.frba.utn.edu.ar':
remote: Enumerating objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

## Agregando o modificando un archivo a su repositorio. (commit, push)

1. Vamos a agregar a nuestro repositorio remoto el primer programa en C que hicimos. Para ello copiamos el archivo **hola.c** desde el directorio **primerPrograma** hasta el directorio del repositorio. Reemplazan lo indicado en azul por su nombre de usuario, que coincide en este caso con el nombre del repositorio.

```
jerome@linuxVm:~$ cp ./primerPrograma/hola.c ./usuario
```

2. Ingresar a la carpeta del repositorio de material utilizando el comando **cd**. (Lo que está en azul debe reemplazarlo por su nombre de usuario)

```
jerome@linuxVm:~$ cd ./usuario
```

3. Para iniciar el seguimiento de un archivo tenemos que ejecutar la siguiente línea de comandos, esto lo deberá solo una vez por cada archivo que desee agregar al repositorio.

```
jerome@linuxVm:~/usuario$ git add ./hola.c
```

4. Para hacer el commit ejecutamos el siguiente comando. Luego del -m podemos colocar una leyenda que identifique el commit. El commit almacena nuestro cambio en el repositorio locals

```
jerome@linuxVm:~/usuario$ git commit -m "Primer Commit"
```

5. Finalmente subimos el cambio al repositorio remoto

```
jerome@linuxVm:~/usuario$ git push
```

6. Cada vez que usted modifique un archivo que ya esta agregado para su seguimiento deberá efectuar solamente las operaciones de commit y push para subir los cambios al repositorio remoto

## Resumen

- git clone
- git add
- git commit
- git push
- git pull
- git config
- git status



## 7. Sistemas de numeración

Un sistema de numeración es un conjunto de reglas que permite representar números utilizando símbolos. Los números que construimos en cada sistema de numeración son entidades abstractas que nos permiten representar cantidades enteras o fracciones de cantidades enteras. El sistema de numeración que utilizamos en la vida cotidiana es el Decimal (o base 10) El cual consiste de diez símbolos (los números del cero al nueve)

### Clasificación de sistemas de numeración

Los sistemas de numeración los podemos clasificar en dos:

- **No posicionales**

Son los más primitivos, representando cantidades haciendo marcas en un palo o nudos en una cuerda por ejemplo. Su característica principal es que las marcas tienen el mismo valor sin importar la posición que ocupen. Esto resulta en que las operaciones de suma y resta consistan en contar, pero sea más dificultoso realizar otras operaciones.

- **Posicionales**

En estos sistemas de numeración cualquier número puede ser representado con un conjunto limitado de símbolos, siendo la cantidad de símbolos lo que define la base del sistema de numeración. Su característica principal es que un determinado símbolo toma un valor distinto dependiendo de la posición que ocupa y cada posición corresponde a sucesivas potencias de la base del sistema de numeración. Si bien es posible definir cualquier número entero como base de un sistema de numeración, sólo analizaremos cuatro las siguientes:

- Binario, base dos
- Octal, base ocho
- Decimal, base diez
- Hexadecimal, base dieciséis

Genéricamente podemos construir un número de cualquier base como

$$A = (a_n; a_{n-1}; \dots; a_0, a_{-1}; a_{-2}; \dots; a_{-k})$$

$$A = a_n b^n + a_{n-1} b^{n-1} + \dots + a_0 b^0 + a_{-1} b^{-1} + \dots + a_{-k} b^{-k}$$

$$A = \sum_{j=n}^k a_j b^j$$

Donde:

- b: Base del sistema de numeración
- A: Número válido del sistema de numeración
- $a_j$ : Símbolos válidos (b-1 símbolos)

- $n + 1$ : Cantidad de símbolos de la parte entera
- $k$ : Cantidad de dígitos de la parte fraccionaria

## Sistema de numeración decimal

Su base es diez y los símbolos válidos son: 0; 1; 2 ;3; 4; 5; 6; 7; 8; 9

$$A = \sum_{j=-k}^n a_j 10^j$$

Ejemplo:

$$1022,74_{10} = 1 * 10^3 + 0 * 10^2 + 2 * 10^1 + 2 * 10^0 + 7 * 10^{-1} + 4 * 10^{-2}$$

## Sistema de numeración binario

Su base es dos y los símbolos válidos son: 0; 1

$$A = \sum_{j=-k}^n a_j 2^j$$

Ejemplo:

$$00100100_2 = 0 * 2^7 + 0 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0$$

## Sistema de numeración octal

Su base es ocho y los símbolos válidos son: 0; 1; 2 ;3; 4; 5; 6; 7

$$A = \sum_{j=-k}^n a_j 8^j$$

Ejemplo:

$$117_8 = 1 * 8^2 + 1 * 8^1 + 7 * 8^0 = 79_{10}$$

## Sistema de numeración hexadecimal

Su base es dieciséis y los símbolos válidos son: 0; 1; 2 ;3; 4; 5; 6; 7; 8; 9; A; B; C; D; E; F

$$A = \sum_{j=-k}^n a_j 16^j$$

Ejemplo:

$$EA80_{16} = E * 16^3 + A * 16^2 + 8 * 16^1 + 0 * 16^0 = 60032_{10}$$

## Símbolos para los sistemas de numeración base 2, 8, 10 y 16

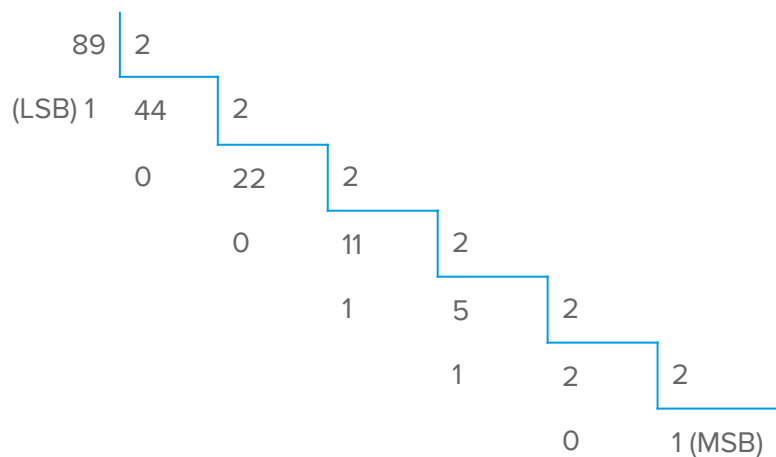
Decimal	Binario	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

## Cambio de base decimal a binario, octal o hexadecimal

El procedimiento para convertir un número decimal en binario, octal o hexadecimal consiste

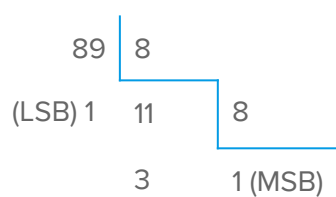
- En tomar el número decimal A y dividirlo por la base que se quiere convertir.
- Luego se repite la operación dividiendo el cociente obtenido por la base destino, hasta que el cociente sea menor que la base destino.
- El resultado se obtiene al concatenar los restos obtenidos de las sucesivas divisiones tomando el primer resto como el dígito menos significativo (LSB: Least Significant Bit) del número en la base destino y el último cociente corresponde al dígito más significativo (MSB: Most Significant Bit) del número en la base destino.

Ejemplo: Convertir el número 89 en base decimal a binario



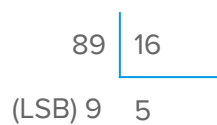
$$89_{10} = 1011001_2$$

Ejemplo: Convertir el número 89 en base decimal a octal



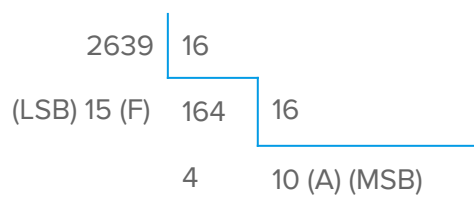
$$89_{10} = 131_8$$

Ejemplo: Convertir el número 89 en base decimal a hexadecimal



$$89_{10} = 59_{16}$$

Ejemplo: Convertir el número 2639 en base decimal a hexadecimal



$$2639_{10} = A4F_{16}$$

## Cambio de base de octal a binario y binario a octal

- Octal a binario

El método para convertir un número octal a binario, consiste en tomar cada dígito octal y convertirlo en binario de forma independiente. Por ejemplo para convertir el número  $4126_{10}$  a binario

Octal	4	1	2	6
Binario	100	001	011	110

$$4126_8 = 100001011110_2$$

- Binario a octal

El método es similar al anterior, pero en este caso se agrupan de a tres dígitos binarios comenzando del bit menos significativo para luego convertir cada grupo de forma individual. Por ejemplo para convertir el número  $11101111000_2$

Binario	11	101	111	000
Octal	3	5	7	0

$$11101111000_2 = 3570_8$$

## Cambio de base hexadecimal a binario o de binario a hexadecimal

El método para realizar estas conversiones es igual al explicado en el apartado anterior con la salvedad que de los dígitos binarios se agrupan de a 4 dígitos. Ejemplos

- Hexadecimal a binario

Hexadecimal	F	1	C	5
Binario	1111	0001	1100	0101

$$F1C5_{16} = 1111000111000101_2$$

- Binario a hexadecimal

Binario	0011	1010	1011	0000
Hexadecimal	3	A	B	0

$$0011101010110000_2 = 3AB0_{16}$$

## Cambio de base octal a hexadecimal o hexadecimal a octal

Esta conversión de hexadecimal a octal se logra transformando de forma intermedia el número hexadecimal a binario, para luego transformar ese número binario a octal. Para el caso octal a hexadecimal se realiza el mismo proceso de pasar de forma intermedia a binario.



## Unidad de información

Un bit (binary-digit) es la mínima unidad de información disponible en sistemas digitales, puede tomar solo dos valores. Los bits pueden ser agrupados en distintas cantidades como por ejemplo:

- Nibble: Es un conjunto de 4 bits, en el puede representarse un dígito hexadecimal.
- Bytes: Es un conjunto de 8 bits.
- Word: Genéricamente es la agrupación de varios bytes por ejemplo:
  - Word16: Agrupa 2 bytes, que son 16 bits
  - Word32: Agrupa 4 bytes, que son 32 bits
  - Word64: Agrupa 8 bytes, que son 64 bits

## Ejercicios

Completa la siguiente tabla con los números en las bases correspondientes

Binario	Octal	Decimal	Hexadecimal
10100101			
	765		
		2019	
			A5A5
			FD07ACD367E
		291338389	
	205037		
11101001			

Verifique los resultados utilizando la calculadora de Lubuntu



## 8. Ingreso de datos e impresión en pantalla

### Tipos de datos

Tipo de dato	Tamaño en bytes	Rango de representación	
char	1 (8 bits)	$[(-1) 2^7 \sim (2^7 - 1)]$	$[-128 \sim 127]$
int	4 (32 bits)	$[(-1) 2^{31} \sim (2^{31} - 1)]$	$[-2.147.483.648 \sim 2.147.483.647]$
float	4 (32 bits)	Ver IEEE 754	
double	8 (64 bits)	Ver IEEE 754	

### Secuencias de escape

Secuencia de escape	Descripción
\n	Salto de línea. Avanza el cursor a la línea siguiente.
\r	Retorno de carro. Coloca el cursor al inicio de la línea.
\t	Tabulador horizontal. Mueve el cursor un tabulador.
\\	Imprime la contrabarra.
\"	Imprime las comillas dobles

### Especificadores de formato

Especificador de formato	Tipo de dato	Descripción
%d	int	Imprime/convierte el dato en un entero decimal
%3d	int	Imprime/convierte el dato en un entero decimal con 3 dígitos, si hay menos deja los espacios adelante al imprimir.
%03d	int	Imprime/convierte el dato en un entero decimal con 3 dígitos, si hay menos dígitos coloca ceros adelante al imprimir.
%c	char	Imprime/convierte el dato en un carácter ASCII
%o	int	Imprime/convierte el dato en un entero octal
%x	int	Imprime/convierte el dato en un entero hexadecimal
%f	float	Imprime/convierte el dato al siguiente formato (-) dd.dddddd
%lf	double	Imprime/convierte el dato al siguiente formato (-) dd.dddddd
%2.3f	float	Imprime/convierte el dato al siguiente formato (-) dd.ddd 2 dígitos antes del punto decimal y 3 después del punto decimal
%%	----	Imprime %

## Tabla ASCII

Puede obtener la tabla ascii desde la terminal

```
jerome@linuxVm:~$ man ascii
```

## Funciones utilizadas

Función	Archivo de cabecera
printf	stdio.h
scanf	stdio.h

## Cómo obtener el manual de una función de biblioteca

Si pido el man de printf como se muestra a continuación, me dará el manual del comando printf y no de la función printf. Podrá notar que en la parte superior al costado de printf aparece un número entre paréntesis indicando el número de manual, en este caso el 1.

```
jerome@linuxVm:~$ man printf
PRINTF(1)                                User Commands                                PRINTF(1)

NAME
    printf - format and print data

SYNOPSIS
    printf FORMAT [ARGUMENT]...
    printf OPTION
... .
```

Para poder ver el manual de la función de librería podemos utilizar la opción -a y recorrerlos hasta encontrar lo que buscamos o buscar en el manual 3 como se muestra a continuación.

```
jerome@linuxVm:~$ man 3 printf
```

## Ejemplos

1. Programa que imprime en pantalla una leyenda solicitando un número entero y después lo imprime

```
#include <stdio.h>

int main (void)
{
    int var;    //-- Declaracion de variable entera --

    /* Imprimo en pantalla una leyenda */
    printf ("Ingrese un numero:\r\n");
    /* Espero al que el usuario ingrese un numero entero */
    scanf ("%d", &var);
```

```
//-- Imprimo una leyenda y el valor almacenado en la variable --
printf ("El numero ingresado es: %d\r\n", var);

//-- Imprimo el número en hexa --
printf ("El numero ingresado es(en hexa): %x\r\n", var);

return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo08\_00.c

```
jerome@linuxVm:~$ gcc ejemplo08_00.c -Wall -oejemplo08_00.out
jerome@linuxVm:~$ ./ejemplo08_00.out
```

## Ejercicios

1. Implemente un programa que utilizando la función printf imprima en pantalla la leyenda Hola Mundo.  
La leyenda en pantalla debe verse de la siguiente forma:

```
jerome@linuxVm:~$ ./ejercicio08_01.out
Hola Mundo
```

Recuerde colocar al final de la leyenda \r\n para que baje una línea

2. Implemente un programa en el cual se define una variable de tipo int llamada varInt inicializada con el valor 376 y luego la imprime en pantalla utilizando la función printf. La salida del programa se muestra a continuación.

```
jerome@linuxVm:~$ ./ejercicio08_02.out
La variable varInt contiene el valor 376
```

3. Repita el programa anterior con los siguientes tipos de datos y valores de inicialización
  - a. char: varChar = 'c';
  - b. int: varInt = 0x55AA;
  - c. int: varInt = 017;
  - d. float: varFloat = 1.27;
  - e. double: varDouble = 2.7172;

### Advertencia:

Se utiliza el punto y no la coma para separar la parte entera de la parte decimal de un número decimal.

### Advertencia:

Los números enteros que comienzan con 0x representan números en hexadecimal.  
Los números enteros que comienzan con 0 representan números en octal.

4. Implemente un programa que utilizando la función `scanf` le solicite al usuario que ingrese un número entero. Luego imprima este número como se muestra a continuación

```
jerome@linuxVm:~$ ./ejercicio08_04.out
Ingrese número: 33
El número ingresado es: 33
```

5. Repita el programa anterior utilizando los siguientes tipos de datos.
- char (para este tipo de dato el usuario ingresará una letra)
  - float
6. Realice un programa que convierta un número entero ingresado por teclado a hexadecimal y octal.

```
jerome@linuxVm:~$ ./ejercicio08_04.out
Ingrese número: 16
El número ingresado fue: 16 (decimal); 0x10 (hexadecimal); 020 (octal)
```

7. Modifique el programa anterior para que le permita al usuario ingresar un número en hexadecimal y el programa lo convierta a octal y decimal. Utilice para ello `scanf` con los especificadores de formato `%o` y `%x`
8. Realice un programa que le pida al usuario que ingrese una letra e imprima su correspondiente código ASCII en decimal y hexadecimal

```
jerome@linuxVm:~$ ./ejercicio08_08.out
Ingrese letra: A
El código ASCII para la letra A es: 65; 0x41
```

9. Realice un programa que le pida al usuario un número tipo float y lo imprima en pantalla con solo dos dígitos luego del punto decimal.

```
jerome@linuxVm:~$ ./ejercicio08_09.out
Ingrese numero: 1.2785
El ingresados es: 1.27
```

■ ■ ■

## 9. Operaciones aritmética - casteo.

### Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de la división

### Funciones utilizadas

Función	Archivo de cabecera
sin	math.h
cos	math.h
tan	math.h
sqrt	math.h
log	math.h
exp	math.h
pow	math.h

### Ejemplos

1. Programa que le solicita al usuario 2 números, los almacena en las variables a y b. Luego calcula  $a^b$  imprime el resultado.

```
#include <stdio.h>
#include <math.h>
int main (void)
{
    double a, b, r;
    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%lf", &a);
    printf ("Ingrese un numero:\r\n");
    scanf ("%lf", &b);
    //-- Imprimo una leyenda y el valor almacenado en la variable --
    r = pow (a, b);
    printf ("El resultado es: %lf\r\n", r);
    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo09\_01.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo09_01.c -Wall -lm -o ejemplo09_01.out
jerome@linuxVm:~$ ./ejemplo09_01.out
```

Se agrega la directiva de linker -l junto con el nombre de la librería a incluir en este caso m. Por ello se observa -lm que debe ser agregado para linkear las funciones de la librería matemática.

#### Advertencia:

No olvide agregar las library necesarias para linkear su código. Use la directiva -l

2. Programa que calcula el promedio de dos números enteros ingresados por teclado.

```
#include <stdio.h>
#include <math.h>
int main (void)
{
    int a, b;
    float promedio;

    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &b);
    //-- Imprimo una leyenda y el valor almacenado en la variable --
    p = (float)(a + b) / (float)2.0;
    printf ("El resultado es: %f\r\n", p);
    return (0);
}
```

## Ejercicios

1. Realice un programa que sume dos números "reales"(use como tipo de dato float) y lo muestre el resultado en pantalla.

```
jerome@linuxVm:~$ ./ejercicio09_01.out
Ingrese número: 1.27
Ingrese número: 1
La suma de 1.27 + 1 es igual a 2.27
```

2. Implemente un programa que permita el ingreso de un número real (float) e imprima por separado la parte entera y la decimal

```
jerome@linuxVm:~$ ./ejercicio09_02.out
Ingrese número: 1.27
La parte entera es: 1
La parte decimal es: 0.27
```

3. Realice un programa que le pida al usuario el ingreso de dos números enteros, calcule la división e informe el cociente (entero) y el resto (use el operador %)

```
jerome@linuxVm:~$ ./ejercicio09_03.out
Ingrese número: 101
Ingrese número: 2
El cociente es: 50
El resto es: 1
```

4. Realice un programa que calcule e imprima (con 4 decimales) la raíz cuadrada de un número ingresado por teclado. ¿Qué ocurre si el número ingresado es negativo? Indique su respuesta en un comentario en el código.

```
jerome@linuxVm:~$ ./ejercicio09_04.out
Ingrese número: 2
La raíz cuadrada de 2 es 1.4142
```

5. Realice un programa que calcule e imprima la coseno de un ángulo expresado en grados ingresado por teclado.

```
jerome@linuxVm:~$ ./ejercicio09_05.out
Ingrese número: 45
El coseno de 45 es 0.707
```

6. Realice un programa que calcule la hipotenusa de un triángulo rectángulo cuyos valores de sus catetos son ingresados por teclado.

```
jerome@linuxVm:~$ ./ejercicio09_06.out
Ingrese número: 1
Ingrese número: 1
La hipotenusa es 1.4142
```

7. Implemente un programa que solicite un número y calcule el logaritmo en base 2 del mismo.

```
jerome@linuxVm:~$ ./ejercicio09_07.out
Ingrese número: 64
El logaritmo en base 2 de 64 es 6
```

■ ■ ■



## 10. Sentencias condicionales if y switch-case

### Operadores relacionales

Operador	Descripción
>	Mayor que
<	Menor que
==	Igual
!=	Distinto
>=	Mayor e igual que
<=	Menor e igual que

### Operadores lógicos

Operador	Descripción
&&	and lógica. (Y lógica)
	or lógica (O lógica)
!	not lógico (Negación lógica)

### Sentencias utilizadas

Sentencias
if
switch - case

### Ejemplos

1. Programa que le solicita al usuario un número y nos indique si vale cero

```
#include <stdio.h>
int main (void)
{
    int a;
    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);
    //-- Verifico si vale cero --
    if (a == 0) {
        printf ("El usuario ingreso cero\r\n");
    }
    return (0);
}
```

2. Programa que le solicita al usuario un número y nos indica si es mayor o igual a 6 o es menor.

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

    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);

    //-- Verifico si es mayor o igual a seis --
    if (a >= 6) {
        printf ("El numero ingresado es mayor o igual a 6\r\n");
    } else {
        printf ("El numero ingresado es menor a 6\r\n");
    }
    return (0);
}
```

3. Programa que le solicita al usuario un número y nos indica si es positivo, negativo o cero.

```
#include <stdio.h>

int main (void)
{
    int a;

    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);

    //-- Verifica si es cero, positivo o negativo --
    if (a == 0) {
        printf ("El usuario ingreso cero\r\n");
    } else {
        if (a > 0) {
            printf ("El usuario ingreso un numero positivo\r\n");
        } else {
            printf ("El usuario ingreso un numero negativo\r\n");
        }
    }
    return (0);
}
```

4. Programa que le solicita al usuario dos números y nos indica cual es mayor o si son iguales

```
#include <stdio.h>

int main (void)
{
    int a, b;

    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &b);

    //-- Compara dos numeros ingresados --
    if (a == b) {
        printf ("Son iguales\r\n");
    } else {
        if (a > b) {
            printf ("El primero es mayor que el segundo\r\n");
        } else {
            printf ("El segundo es mayor que el primero\r\n");
        }
    }
    return (0);
}
```

5. Programa que le solicita al usuario un número e indica si el mismo está entre 1 y 10 (incluyendo ambos)

```
#include <stdio.h>

int main (void)
{
    int a;

    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);

    //-- Verifico si el numero esta entre 1 y 10 --
    if ((a >= 1) && (a <= 10)) {
        printf ("El numero ingresado esta entre 1 y 10\r\n");
    }
    return (0);
}
```

6. Programa que le solicita al usuario un número e indica si el mismo es mayor a 10 o menor que 1

```
#include <stdio.h>

int main (void)
{
    int a;

    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);

    //-- Verifico si el numero es mayor a 10 o menor a 1 --
    if ((a < 1) || (a > 10)) {
        printf ("El numero ingresado es mayor que 10 o menor a 1 \r\n");
    }
    return (0);
}
```

7. Programa que le solicita al usuario un carácter e indica si el mismo es una letra mayúscula

```
#include <stdio.h>

int main (void)
{
    char a;

    /* Ingreso de datos */
    printf ("Ingrese un caracter:\r\n");
    scanf ("%c", &a);

    //-- Verifico si es una letra mayúscula --
    if ((a >= 'A') && (a <= 'Z')) {
        printf ("El caracter ingresado es una letra mayuscula\r\n");
    }
    return (0);
}
```

Consulte el el man del ascii para ver el orden de los caracteres en la tabla.

## 8. Programa que le solicita al usuario un carácter e indica si el mismo es una vocal minúscula

```
#include <stdio.h>

int main (void)
{
    char a;

    /* Ingreso de datos */
    printf ("Ingrese un caracter:\r\n");
    scanf ("%c", &a);

    //-- Verifico si es vocal --
    switch (a) {
        case 'a':
            printf ("Es vocal\r\n");
            break;

        case 'e':
            printf ("Es vocal\r\n");
            break;

        case 'i':
            printf ("Es vocal\r\n");
            break;

        case 'o':
            printf ("Es vocal\r\n");
            break;

        case 'u':
            printf ("Es vocal\r\n");
            break;

        default:
            printf ("No es vocal\r\n");
            break;
    }
    return (0);
}
```

9. Programa que pide el ingreso de un número entero de 1 dígito e imprime con letras el valor. En caso de que el dígito ingresado sea negativo o tenga más de un dígito escribe una leyenda indicándolo

```
#include <stdio.h>

int main (void)
{
    int a;

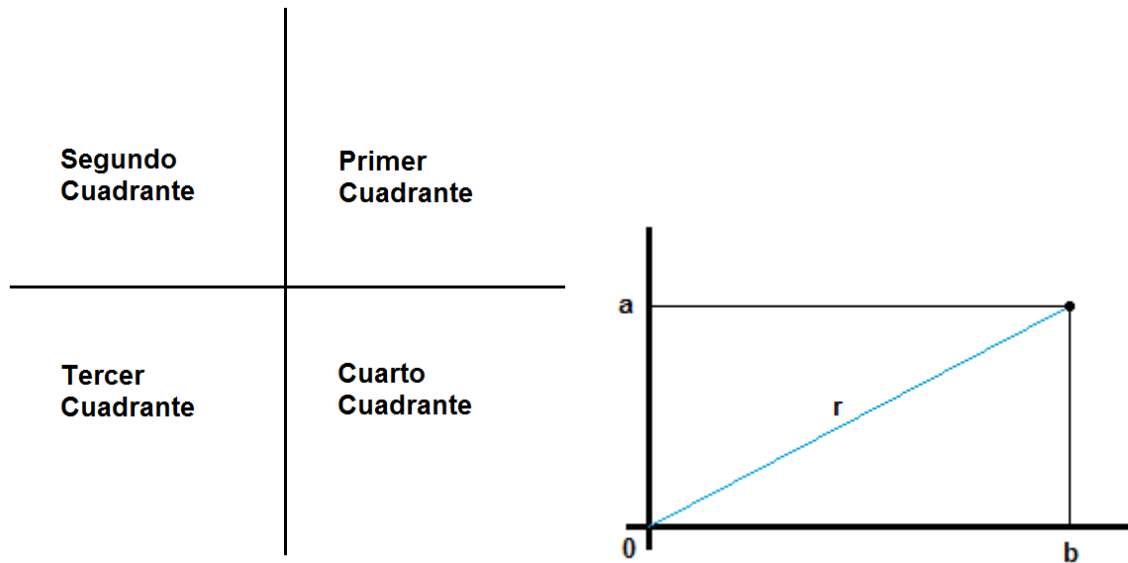
    /* Ingreso de datos */
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);

    //-- Verifico si es vocal --
    switch (a) {
        case 0: printf ("cero\r\n");    break;
        case 1: printf ("uno\r\n");    break;
        case 2: printf ("dos\r\n");    break;
        case 3: printf ("tres\r\n");    break;
        case 4: printf ("cuatro\r\n"); break;
        case 5: printf ("cinco\r\n");  break;
        case 6: printf ("seis\r\n");   break;
        case 7: printf ("siete\r\n");  break;
        case 8: printf ("ocho\r\n");   break;
        case 9: printf ("nueve\r\n");  break;

        default:
            printf ("Ingreso invalido\r\n");
            break;
    }
    return (0);
}
```

## Ejercicios

1. Escriba un programa que permita el ingreso de un número entero e indique si el mismo es par o impar. (use `if` y el operador `%`)
2. Escriba un programa donde ingresa por teclado un par de valores enteros, que representan las coordenadas rectangulares de distintos puntos en el plano. Se pide determinar e informar por pantalla:
  - Si ambos valores son cero.
  - A cuál cuadrante pertenece el punto.
  - La distancia al origen de coordenadas. que se calcula como:  $r = \sqrt{a^2 + b^2}$



3. Realice un programa que indique si la letra ingresada es mayúscula, minúscula, un número u otro carácter. Verifique el funcionamiento con los siguientes casos

Valores de entrada	stdout (pantalla)
desde 'a' hasta 'z'	Es una letra minúscula.
desde 'A' hasta 'Z'	Es una letra mayúscula.
desde '0' hasta '9'	Es un número
Otro carácter	Es otro carácter

4. Elabore un programa donde se ingresan dos valores reales y el símbolo de la operación ('+', '-', '\*', '/'). Se deberá presentar por pantalla, los datos ingresados, la operación y el resultado. Si el símbolo utilizado no correspondiera a ninguna de las cuatro operaciones deberá presentar un mensaje de "Operación no válida". Para leer el símbolo de la operación desde el teclado use `scanf ("%*c%c", &op)`. (El programa deberá resolverse mediante el uso de la estructura switch)
5. Implemente un programa que pida el ingreso de las notas de dos parciales para determinar si el estudiante de informática I:
  - Firmó la materia.
  - Promociono
  - Debe recuperar algún parcial.

La nota válida estará en el rango de 1 a 10, en caso de error sale del programa.

## 11. Sentencias de repetición for; while; do-while

### Operadores asignación, incremento y decremento

Operador	Descripción
=	Asignación (igual)
+=	Incremento. Ejemplo <code>x += 2;</code> es equivalente a <code>x = x + 2;</code>
-=	Decremento. Ejemplo <code>x -= 2;</code> es equivalente a <code>x = x - 2;</code>
++	Pre o post incremento.
--	Pre o post decremento.

### Sentencias utilizadas

Sentencias
for
while
do - while

### Ejemplos

1. Programa de ejemplo de operadores de pre y post incremento.

```
#include <stdio.h>

int main (void)
{
    int w, x, y, z;

    w = 0;      x = 0;      y = 0; z = 0;
    //-- Imprimo los valores originales --
    printf ("w = %d\tx = %d\ty = %d\tz = %d\r\n", w, x, y, z);

    //-- Incremento en uno e imprimo --
    w = w + 1;  x++; ++y; z+=1;
    printf ("w = %d\tx = %d\ty = %d\tz = %d\r\n", w, x, y, z);

    //-- Incremento en uno e imprimo --
    printf ("w = %d\tx = %d\ty = %d\tz = %d\r\n", w = w + 1, x++, ++y, z+=1);

    //-- Imprimo los valores --
    printf ("w = %d\tx = %d\ty = %d\tz = %d\r\n", w, x, y, z);

    return (0);
}
```



2. Programa que imprime la leyenda Hola Mundo 10 veces.

```
#include <stdio.h>

int main (void) {
    int i;

    for (i = 0; i < 10; i++) {
        printf ("Hola Mundo\r\n");
    }

    return (0);
}
```

3. Programa que imprime los números enteros del cero al nueve

```
#include <stdio.h>
int main (void)
{int i;
    for (i = 0; i < 10; i++) {
        printf ("%d\r\n", i);
    }

    return (0);
}
```

4. Programa que imprime le pide números al usuario sucesivamente hasta que este ingrese uno mayor que 10

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

    do {
        printf ("Ingrese un numero:\r\n");
        scanf ("%d", &a);
    } while (a < 10);

    printf ("El numero ingresado fue mayor que 10\r\n");

    return (0);
}
```

- Programa que acumula los números ingresados por teclado mientras esta acumulacion no supere el valor 100. Al superar el valor de 100 informa la suma total en pantalla.

```
#include <stdio.h>
int main (void)
{
    int a;
    int suma = 0;


    while (suma <= 100) {
        printf ("Ingrese un numero:\r\n");
        scanf ("%d", &a);
        suma += a;
    }

    printf ("La suma es %d\r\n", suma);

    return (0);
}
```

## Ejercicios

- Implemente un programa que imprima los números enteros pares entre el cero y el cien.
- Implemente un programa que le pida al usuario dos números enteros e imprima todos los números enteros entre ellos, incluyendo los límites. Si los dos números son iguales deberá imprimir ese número solamente. Ejemplos:
  - El usuario ingresa -1 y 2. El programa debe imprimir: -1; 0; 1; 2
  - El usuario ingresa 2 y -1. El programa debe imprimir: 2; 1; 0; -1
  - El usuario ingresa 2 y 2. El programa debe imprimir: 2
- Implemente un programa utilizando la sentencia **for** que calcule el promedio de 10 números enteros ingresados por teclado.
- Realice un programa utilizando la sentencia **do-while** que imprima los números del 0 al 9
- Realice un programa utilizando la sentencia **while** que imprima los números del 0 al 9
- Implemente un programa que acumule los números ingresados por teclado mientras esta acumulacion no supere el valor 100. Informe este número en pantalla. Si el usuario ingresa: 10, 80, 20 el programa debe imprimir: 90
- Realice un programa que calcule el promedio de todas las notas ingresadas por teclado. Las notas válidas están en el intervalo [1; 10] y el ingreso de datos terminará cuando el usuario coloque como nota el valor -1 el cual no deberá tenerse en cuenta para el promedio. Si el usuario ingresa un número fuera del rango deberá informar con una leyenda en pantalla y continuar el ingreso de datos. Finalmente muestre el promedio con 2 decimales.
- Realice un programa que calcule el promedio de todas las notas ingresadas por teclado. Las notas válidas están en el intervalo [1; 10] y el ingreso de datos terminará cuando el usuario coloque como



nota el valor -1 el cual no deberá tenerse en cuenta para el promedio. Si el usuario ingresa un número fuera del rango deberá informar con una leyenda en pantalla y continuar el ingreso de datos. Además deberá controlar que la cantidad de notas válidas ingresadas sea mayor que tres en caso contrario deberá indicarle al usuario que continúe con el ingreso de datos. Finalmente muestre el promedio con 2 decimales. Utilice `do-while`.

9. Escriba un programa que le permita al usuario jugar a adivinar un número secreto entre 0 y 9. Para ello le pedirá que ingrese un número (el secreto) y luego le pedirá que ingrese números hasta adivinarlo. Si el usuario ingresa un número fuera del rango válido [0; 9] deberá indicarlo y en el caso del número secreto deberá continuar pidiendo hasta que ingrese el valor en el rango válido. El programa continuará pidiendo números hasta que el usuario lo adivine indicando "GANASTE" con lo cual debe de comenzar nuevamente el juego o se equivoque 3 veces indicando "NO GANASTE, INTENTALO OTRA VEZ" . Utilice `while`.
10. Implemente un programa que le pida al usuario que ingrese un número entero y luego informe la cantidad de dígitos del mismo. Utilice `while`.



## 12. Directiva de precompilador define

### Directivas de precompilación

Directiva	Descripción
define	Define una constante simbólica

### Ejemplos

1. Programa que imprime la leyenda Hola Mundo 10 veces. Coloque este umbral en un define.

```
#include <stdio.h>
#define CANT ((int)10)

int main (void)
{
    int i;

    for (i = 0; i < CANT; i++) {
        printf ("Hola Mundo\r\n");
    }

    return (0);
}
```

2. Programa que calcula el promedio de CANT datos ingresados.

```
#include <stdio.h>
#define CANT ((int)10)

int main (void)
{
    int i;
    int acc = 0, n;
    float promedio;

    for (i = 0; i < CANT; i++) {
        printf ("Ingrese numero\r\n");
        scanf ("%d", &n);
        acc += n;
    }

    promedio = acc / (float)CANT;

    printf ("Promedio %f\r\n", promedio);

    return (0);
}
```

### Advertencia:

No se puede modificar el valor de un `define` en ejecución, por ejemplo NO puede hacer `CANT = 0`

## Ejercicios

1. Implemente un programa que le pida un número al usuario e indique si el número es mayor a 100. Coloque este umbral en un `define`.
2. Implemente un programa que calcule el factorial de un número entero positivo ingresado por teclado

El factorial de  $n$  es el producto de todos los enteros positivos hasta  $n$  inclusive.

- Para  $n > 0 \Rightarrow n! = 1 \times 2 \times 3 \times \dots \times n$
- Para  $n = 0 \Rightarrow n! = 1$

Verifique todas las condiciones que considere pertinentes para que el resultado obtenido sea correcto. Determine el número máximo al cual le puede calcular el factorial. Verifique el funcionamiento del programa con los siguientes valores.

x	x! (salida de Stdout)
-1	No puedo calcular el factorial de un número negativo
0	1
1	1
2	2
3	6
12	479001600

3. Ingresar un número entero por teclado y determinar si es primo.
4. Implemente un programa que le pida al usuario que ingrese 10 (use `define`) números enteros e informe el mayor y el menor de todos los ingresados.
5. Implemente un programa que le pida al usuario que ingrese 10 (use `define`) números enteros positivo y cuente la cantidad de números pares e impares ingresados.



## 13. Funciones

### Constantes simbólicas

Constante	Descripción
M_PI	Número Pi, definido en math.h

### Ejemplos

1. Función que imprime la leyenda "Hola Mundo".

```
#include <stdio.h>

void imprimir (void)
{
    printf ("Hola Mundo\r\n");
}

int main (void)
{
    imprimir ();
    return (0);
}
```

2. Función que imprime la leyenda "Hola Mundo", la cantidad de veces que se paso como parametro.

```
#include <stdio.h>

void imprimir (int cant)
{
    int i;

    for (i = 0; i < cant; i++) {
        printf ("Hola Mundo\r\n");
    }
}

int main (void)
{
    int c;

    printf ("Ingrese cantidad\r\n");
    scanf ("%d", &c);

    imprimir (c);

    return (0);
}
```

3. Función que devuelve el número pasado como parámetro sumándole 1.

```
#include <stdio.h>
#define CANT ((int)10)

int suma1 (int a)
{
    int r;

    r = a + 1;

    return (r);
}

int main (void)
{
    int c, r;
    printf ("Ingrese cantidad\r\n");
    scanf ("%d", &c);

    r = suma1 (c);
    printf ("Resultado = %d\r\n", r);

    return (0);
}
```

4. Función que convierta de mayúsculas a minúsculas la letra pasada como parámetro

char pasaAminusculas(char a)

```
#include <stdio.h>
#define CANT ((int)10)

char pasaAminusculas (char a)
{
    char r;

    if ((a >= 'A') && (a <= 'Z')) {
        r = (a - 'A') + 'a';
    } else {
        r = a;
    }

    return (r);
}
```

```

int main (void)
{
    char c, r;

    printf ("Ingrese letra\r\n");
    scanf ("%c", &c);

    r = pasaAminusculas (c);
    printf ("Resultado = %c\r\n", r);

    return (0);
}

```

#### 5. Función que devuelve la suma de dos números enteros pasados como parámetros

```

#include <stdio.h>

int sumaDosNumeros (int a, int b)
{
    int r;

    r = a + b;

    return (r);
}

int main (void)
{
    int x, y, r;
    printf ("Ingrese numero\r\n");
    scanf ("%d", &x);
    printf ("Ingrese numero\r\n");
    scanf ("%d", &y);

    r = sumaDosNumeros (x, y);
    printf ("Resultado = %d\r\n", r);

    return (0);
}

```

**Advertencia:**  
 Recuerde definir la función o colocar el prototipo antes de la llamada a la función.



## Ejercicios

1. Implemente una función que calcule el área de un círculo. Utilice la constante de `pi` de `math.h`. El prototipo es

```
float areaCirculo (float radio);
```

2. Implemente una función que calcule el perímetro de un círculo. Utilice la constante de `pi` de `math.h`. El prototipo es

```
float perimetroCirculo (float radio);
```

3. Implemente una función a la cual le pase un carácter como parámetro y me devuelva

- 0: si el carácter es una letra mayúscula.
- 1: si el carácter es una letra minúscula.
- 2: si el carácter es un número.
- 3: en caso que no sea ninguno de los anteriores.

Utilice `define` para las constantes. El prototipo es `int filtroASCII (char character);`

4. Implemente una función que realice las cuatro operaciones básicas entre dos números de tipo `float` y retorne el resultado. El prototipo de la función es el siguiente

```
float calculo (float opA, float opB, char op)
```

Donde:

- `opA` y `OpB` son los números con los cuales se debe realizar la operación
- `op`: La operación a realizar
  - '+': Realiza la suma.
  - '-': Realiza la resta.
  - '\*': Realiza el producto
  - '/': Realiza la división
- La función devuelve cero si la operación es inválida.

5. Implemente una función que le pase como parámetro dos números que representan los catetos de un triángulo rectángulo y me devuelva la hipotenusa. El prototipo es

```
float calcHipo (float catetoA, float catetoB)
```

6. Implemente una función que calcule el factorial de un número pasado como parámetro. Si el factorial no puede ser calculado la función debe devolver cero.

```
int factorial (int n);
```



## 14. Vectores y strings

### Ejemplos

1. Programa en el cual se define un vector de diez números enteros y se inicializa en tiempo de ejecución con los números del 0 al 9. Luego lo imprime en orden ascendente y descendente.

```
#include <stdio.h>

#define CANT ((int)10)

int main (void)
{
    int v[CANT];

    //-- Inicializo el vector --
    for (i = 0; i < CANT; i++) {
        v[i] = i;
    }

    //-- Imprimo en orden ascendente --
    for (i = 0; i < CANT; i++) {
        printf ("%d\r\n", v[i]);
    }

    //-- Imprimo en orden descendente --
    for (i = CANT - 1; i >= 0; i--) {
        printf ("%d\r\n", v[i]);
    }

    return (0);
}
```

2. Programa en el cual se define un vector de diez números enteros y se inicializa en tiempo de compilación con la tabla de multiplicar del 5. Pida al usuario que ingrese un número entre cero y nueve, para luego calcular la multiplicación por cinco de dicho número indexando el vector.

```
#include <stdio.h>
#define CANT ((int)10)
int main (void)
{
    int m5[CANT] = {0, 5, 10, 15, 20, 25, 30, 35, 40, 45};
    int num;

    //-- Ingreso numero --
    printf ("Ingrese numero\r\n");
    scanf ("%d", &num);
}
```

```

    if ((num >= 0) && (num <= 9)) {
        printf ("El resultado es: %d\r\n", m5[num]);
    } else {
        printf ("Imposible calcular\r\n");
    }

    return (0);
}

```

3. Programa en el cual se le pide al usuario que ingrese 10 números, para luego imprimir de forma separada los números pares e impares ingresados. El vector se inicializa en cero en tiempo de compilación.

```

#include <stdio.h>
#define CANT ((int)10)

int main (void)
{
    int v[CANT] = {0};
    int i;

    //-- Ingreso numero --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese numero\r\n");
        scanf ("%d", &v[i]);
    }

    printf ("Los pares son: \r\n");
    for (i = 0; i < CANT; i++) {
        if ((v[i] % 2) == 0) {
            printf ("%d.%d\r\n", i, v[i]);
        }
    }

    printf ("Los impares son: \r\n");
    for (i = 0; i < CANT; i++) {
        if ((v[i] % 2) != 0) {
            printf ("%d.%d\r\n", i, v[i]);
        }
    }

    return (0);
}

```

**Advertencia:**

Verifique siempre que el índice del vector esté dentro del rango definido.

4. Programa que muestra cómo generar 10 números pseudoaleatorios.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define CANT ((int)10)

int main (void)
{
    int i;

    srand(time(NULL));

    for (i = 0; i < CANT; i++) {
        printf ("%d\r\n", rand());
    }

    return (0);
}
```

5. Programa que le pide al usuario que ingrese una palabra para luego indicar la cantidad de vocales que tiene la misma.

```
#include <stdio.h>
#define CANT ((int)32)
#define CANT_VOCALES ((int)10)
int main (void)
{
    char v[CANT];
    char vocales[CANT_VOCALES] = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'};
    int vocalesCont = 0;
    int i, j;

    //-- Ingreso palabra --
    printf ("Ingrese palabra\r\n");
    scanf ("%s", &v[0]);

    //-- Cuento las vocales --
    i = 0;
    while (v[i] != '\0') {
        for (j = 0; j < sizeof (vocales); j++) {
            if (v[i] == vocales[j]) {
                vocalesCont++;
            }
        }
        i++;
    }
    printf ("La cantidad de vocales es: %d\r\n", vocalesCont);
    return (0);
}
```

6. Programa que inicializa un string en tiempo de compilación y lo muestra en pantalla.

```
#include <stdio.h>
#define CANT ((int)32)
#define CANT_VOCALES ((int)10)
int main (void)
{
char str0[] = "Hola";
char str1[] = {'h', 'o', 'l', 'a', '\0'};

    //-- Ingreso palabra --
    printf ("Palabra %s\r\n", str0);
    printf ("Palabra %s\r\n", str1);

    return (0);
}
```

7. Programa que indica la cantidad de caracteres de un string ingresado por teclado.


```
#include <stdio.h>
#define CANT ((int)32)
#define CANT_VOCALES ((int)10)
int main (void)
{
char str0[] = "Hola";
char str1[] = {'h', 'o', 'l', 'a', '\0'};

    //-- Ingreso palabra --
    printf ("Palabra %s\r\n", str0);
    printf ("Palabra %s\r\n", str1);

    return (0);
}
```

## Ejercicios

1. Escriba un programa en el cual defina un vector que almacene todas las letras del alfabeto, salvo la ñe (use como tipo de dato char). Inicialicelo en tiempo de ejecución. Imprímalo en orden alfabético
2. Implemente un código que genere un número pseudo aleatorio entre [0; 99] , luego el programa debe pedirle al usuario números e indicarle si acertó o no el número generado pseudo aleatoriamente. Si el usuario repite un número el programa deberá indicarlo, lo mismo si el usuario ingresa uno fuera de rango.
3. Realice un programa que permita al usuario ingresar las alturas de un grupo de como máximo cien personas. El fin del ingreso de datos ocurre cuando la altura ingresada sea menor que cero. Luego se le pedirá al usuario que ingrese dos valores de altura y el programa debe indicar la cantidad de personas con alturas en ese rango. Si el intervalo ingresado por el usuario es inválido indíquelo por pantalla.



4. Implemente un programa que le pida al usuario números enteros y los almacene en 4 vectores diferentes según su tipos.

- Positivos y cero.
- Negativos.
- Pares.
- Impares.

El usuario ingresara diez valores y luego el programa deberá imprimir por pantalla la cantidad de números almacenados en cada vector y posteriormente los datos almacenados en cada uno de ellos.

5. Implemente un programa que le pida al usuario que ingrese una palabra por teclado y la imprima en mayúscula por pantalla.

6. Implemente un programa que le pida al usuario que ingrese una palabra por teclado e informe la cantidad de caracteres que esta posee sin contar el '\0'.

7. Implemente un programa que le pida al usuario que ingrese dos palabras por teclado e indique si son iguales o cual aparece primero en el diccionario.

8. Implemente un programa que le pida al usuario que ingrese una palabra y un carácter por teclado. A continuación reemplace este carácter en la palabra por asterisco. Finalmente debe indicar la cantidad de veces que reemplazó el carácter.



## 15. Punteros

### Forma básica de usar punteros

Con el fin de mantener controlados a los punteros evitando de esta forma violaciones de segmento se recomienda seguir las siguientes recomendación en su uso

- Declarar puntero.
- Inicializar el puntero siempre antes de usarlo. Puede inicializarlo a:
  - A la variable que corresponda
  - A NULL
- Hacer que el tipo de dato coincida con el del tipo de dato del puntero.
- Si la variable a donde inicializa el puntero es un vector inicialicelo al elemento cero.
- Cuando el puntero se inicializó con el elemento cero de un vector utilice el puntero de esta forma  $*(p + i)$  donde  $i$  es el índice entero

Estas recomendaciones evitan el uso de punteros modificando el valor al que apuntan, por ejemplo haciendo `p++`;

Ejemplos de uso de punteros

	Variable int	Vector de int
<b>Declaración de variable</b>	<code>int a;</code>	<code>int v[3];</code>
<b>Declaración de puntero</b>	<code>int *p;</code>	<code>int *p;</code>
<b>Inicialización de puntero</b>	<code>p = &amp;a;</code>	<code>p = &amp;v[0];</code>
<b>Asigno un valor la variable a</b>	<code>a = 10;</code>	<code>v[0] = 1;</code> <code>v[1] = 2;</code> <code>v[2] = 3;</code>
<b>Asigno un valor a la variable usando un puntero</b>	<code>*p = 10; // a = 10;</code>	<code>*(p + 0) = 1; // v[0] =1;</code> <code>*(p + 1) = 2; // v[1] =2;</code> <code>*(p + 2) = 3; // v[2] =3;</code>
<b>Imprimo el valor de la variable</b>	<code>printf ("%d\r\n", a);</code>	<code>printf ("%d\r\n", v[0]);</code> <code>printf ("%d\r\n", v[1]);</code> <code>printf ("%d\r\n", v[2]);</code>
<b>Imprimo el valor de la variable usando el puntero</b>	<code>printf ("%d\r\n", a);</code>	<code>printf ("%d\r\n", *(p + 0));</code> <code>printf ("%d\r\n", *(p + 1));</code> <code>printf ("%d\r\n", *(p + 2));</code>

## Ejemplos

1. Programa en el que se instancia una variable tipo `int` y un puntero del mismo tipo. Se apunta el puntero a esta variable y la inicializa usando el puntero con el valor `0x55`. Luego imprime el valor almacenado en la variable utilizando el puntero.

```
#include <stdio.h>

int main (void)
{
    int a;
    int *p;          //-- Declaro el puntero --

    p = &a;          //-- Inicializo el puntero --

    *p = 0x55;       //-- Inicializo la variable a usando el puntero --

    printf ("%d\r\n", *p); //-- Imprimo usando el puntero --

    return (0);
}
```

2. Programa que define un vector de 10 elementos de tipo `char`, luego lo inicializa con los números del '0' al '9' utilizando un puntero. Finalmente imprime dicho vector usando un puntero.

```
#include <stdio.h>

#define CANT ((int)CANT)

int main (void)
{
    char c[CANT];
    char *p;

    p = &c[0];        //-- Inicializo el puntero --

    //-- Inicializo el vector usando el puntero --
    for (i = 0; i < CANT; i++) {
        *(p + i) = '0' + i;
    }

    //-- Imprimo el vector usando el puntero --
    for (i = 0; i < CANT; i++) {
        printf ("%d\r\n", *(p + i));
    }

    return (0);
}
```



3. Se realiza una función que suma dos números enteros que son pasado como parámetro y retorna el resultado usando el return de la función.

```
#include <stdio.h>

int suma (int *a, int *b)
{
    int r;

    r = *a + *b;

    return (r);
}

int main (void)
{
    int x, y, r;

    //-- Ingreso de los dos numeros a sumar --
    printf ("Ingrese numero\r\n");
    scanf ("%d", &x);
    printf ("Ingrese numero\r\n");
    scanf ("%d", &y);

    r = suma (&x, &y);

    printf ("El resultado %d\r\n", r);

    return (0);
}
```

4. Se realiza una función que suma dos números enteros que son pasado como parámetro y retorna el resultado usando un puntero.

```
#include <stdio.h>

void suma (int *a, int *b, int *r)
{
    *r = *a + *b;

    return;
}

int main (void)
{
    int x, y, r;

    //-- Ingreso de los dos numeros a sumar --
    printf ("Ingrese numero\r\n");
    scanf ("%d", &x);
    printf ("Ingrese numero\r\n");
    scanf ("%d", &y);
}
```

```

    suma (&x, &y, &r);

    printf ("El resultado %d\r\n", *r);

    return (0);
}

```

5. Se realiza una función que devuelve un puntero al elemento central de un vector de enteros. Finalmente se imprime el elemento central del vector usando el puntero obtenido con el

```

#include <stdio.h>

#define CANT ((int)3)

int* medio (int *dataPtr, int dataCant)
{
    int *p;

    p = dataPtr + (dataCant / 2);

    return (p);
}

int main (void)
{
    int v[CANT] = {1, 2, 3};
    int *q;

    q = medio (&v[0], CANT);

    //-- Imprimo el elemento central --
    printf ("El elemento central vale %d\r\n", *q);

    return (0);
}

```

**Advertencia:**  
No olvide inicializar el puntero antes de utilizarlo.

**Advertencia:**  
Evite realizar operaciones de incremento o decremento sobre punteros, por ejemplo: p++ ó p--;

## Ejercicios

1. Implemente una función que calcule el promedio de un vector de tipo float. El prototipo de la función es

```
float promedio (float *dataPtr, int dataCant);
```

Donde:

dataPtr: Es el puntero a los datos.

dataCant: Es la cantidad de elementos del vector apuntado.

2. Implemente una función que invierta el contenido de dos variables cuyo prototipo es:

```
void swap (int *a, int *b);
```

Donde: a y b son los punteros a las variables que se les debe invertir el contenido.

3. Implemente una función que verifique si un vector de int está ordenado de manera creciente o decreciente. El prototipo es

```
int orden (int *dataPtr, int dataCant);
```

Donde:

dataPtr: Es el puntero a los datos

dataCant: Es la cantidad de elementos del vector apuntado

Devuelve: 1 si el orden es creciente; 0 si no está ordenado; -1 si el orden es decreciente.

4. Implemente una función que devuelva un puntero al elemento que contiene el valor máximo de un vector.

```
int * myMax (int *dataPtr, int dataCant);
```

Donde:

dataPtr: Es el puntero a los datos

dataCant: Es la cantidad de elementos del vector apuntado

Devuelve: El puntero al elemento que contiene el máximo

5. Implemente una función que devuelva un puntero al elemento que contiene el valor mínimo de un vector.

```
int * myMin (int *dataPtr, int dataCant);
```

Donde:

dataPtr: Es el puntero a los datos

dataCant: Es la cantidad de elementos del vector apuntado

Devuelve: El puntero al elemento que contiene el mínimo

6. Implemente una función que imprima en hexadecimal todos los caracteres de un string. El prototipo de la función es el siguiente.

```
int myHexa (char *dataPtr)
```

Donde:

dataPtr: Es el puntero al string a pasar a hexadecimal.

Devuelve la cantidad de caracteres sin contar el '\0'

7. Implemente una función que se le pase como parámetro un puntero a un string e indique si este contiene solo los dígitos del '0' al '9'. El prototipo de la función es el siguiente.

```
int esNumero (char *dataPtr);
```

Donde:

dataPtr: Es el puntero al string

Devuelve: Cero si algún carácter del string no corresponde a un dígito del '0' al '9', devuelve uno

8. Implemente una función que recibe un puntero a un string que contiene un número y devuelve ese número en un `int`. El prototipo de la función es el siguiente.

```
int convertirA_Int(char *dataPtr)
```

Donde:

dataPtr: Es el puntero al string a pasar a convertir.

Devuelve: El número entero si pudo convertirlo, si el string contiene un carácter distinto a los dígitos del '0' al '9' y menos uno en caso de error.



## 16. Asignación dinámica de memoria

Funciones utilizadas de `stdlib.h`

Función	Descripción
<code>malloc</code>	Asigna dinámicamente memoria.
<code>free</code>	Libera memoria asignada dinámicamente.
<code>realloc</code>	Asigna dinámicamente memoria.

### Ejemplos

1. Programa en el que se instancia dinámicamente un vector de 10 elementos de tipo `int`. Estos se inicializan con los números del 0 al 9. Finalmente se imprime este vector en pantalla y se libera la memoria asignada dinámicamente.

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

#define CANT ((int)10)

int main (void)
{
    int *p;
    int i;

    //-- Pido memoria dinamicamente. --
    p = (int *)malloc (sizeof (*p) * CANT);
    if (p == NULL) {
        return (-1);
    }
    //-- Inicializo el vector --
    for (i = 0; i < CANT ; i++) {
        *(p + i) = i;
    }

    //-- Imprimo el vector --
    for (i = 0; i < CANT ; i++) {
        printf ("%d\r\n", *(p + i));
    }

    //-- Libero memoria --
    free (p);

    return (0);
}
```

2. Programa en el que se instancia dinámicamente un vector de 10 elementos de tipo int. Estos se inicializan con números del 0 al 9. Luego utilizando realloc se piden 10 elementos más y se los inicializa con los números del 10 al 19. Finalmente se imprime este vector en pantalla y se libera la memoria asignada dinámicamente.

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

#define CANT ((int) 10)

int main (void)
{
    int *p, *pBack;
    int cantTotal, i;
    int cantUsada;

    //-- Pido memoria para CANT elementos --
    cantTotal = CANT;
    cantUsada = 0;
    p = (int*)malloc (sizeof (*p) * cantTotal);
    if (p == NULL) {
        return (-1);
    }

    //-- Inicializo la zona de memoria asignada con cero--
    for (i = cantUsada; i < cantTotal; i++) {
        *(p + i) = i;
        cantUsada++;
    }

    //-- Pido memoria para CANT elementos mas--
    pBack = (int*)realloc (p, sizeof (*p) * (cantTotal + CANT));
    if (pBack != NULL) {
        //-- Realloc ok --
        //-- Hay cantTotal + CANT elementos --
        cantTotal += CANT;
        p = pBack;
    }

    //-- Inicializo la zona de memoria asignada con uno--
    for (i = cantUsada; i < cantTotal; i++) {
        *(p + i) = i;
        cantUsada++;
    }

    //-- Imprimo el vector utilizado, en este caso es todo --
    for (i = 0; i < cantUsada; i++) {
        printf ("%d\r\n", *(p + i));
    }
}
```

```
//-- Libero memoria --
free (p);
return (0);
}
```

## Ejercicios

1. Implemente un programa que utilizando `malloc` reserve memoria para almacenar las letras mayúsculas del alfabeto (salvo la Ñ). Luego imprima la zona reservada y la libere antes de finalizar el programa.
2. Implemente un programa que le pregunte al usuario cuantas letras desea ingresar, reserve utilizando `malloc` la memoria necesaria y luego le pida al usuario caracteres hasta llenar el vector reservado dinámicamente. Finalmente debe imprimir el vector en orden inverso al ingresado y liberar la memoria reservada.
3. Implemente un programa que le pida al usuario que ingrese letras y las almacene en memoria, el fin del ingreso de datos ocurre cuando el usuario ingresa el signo de admiración. Luego se deberán imprimir todas las letras ingresadas por el usuario. (Use `realloc`)



## 17. Algoritmos integradores

### Funciones utilizadas de string.h

Función	Descripción
strcat	Concatenar un string con otro
strchr	Busca un caracter en un string
strcmp	Compara dos strings alfabéticamente
strcpy	Copia un string en otro.
strlen	Obtiene el tamaño de un string
strstr	Busca un string en otro string
memcpy	Copia una cantidad de bytes de una zona de memoria en otra
memset	Escribe una zona de memoria con un carácter determinado.
memcmp	Compara dos zonas de memoria.

### Funciones utilizadas de stdlib.h

Función	Descripción
qsort	Ordena un vector.

### Ejemplos

1. Implemente una función que cuente la cantidad de caracteres de un string sin contar el '\0'. El prototipo de la función es

```
int myStrLen (char *s);
```

Además implemente un main que verifique automáticamente el funcionamiento básico de la función implementada

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

int myStrLen (char *s)
{
    int c = 0;

    if (s != NULL) {
        while (*(s + c) != '\0') {
            c++;
        }
    }
    return (c);
}
```



```

int main (void)
{
    //-- Vectores de prueba --
    char v0[]="Hola";
    char v1[]="";
    char v2[]="Hola como te va?";
    int r;

    //-- Prueba de las funciones --
    r = myStrLen (v0);      printf ("strlen > %s: %d\r\n", v0, r);
    r = myStrLen (v1);      printf ("strlen > %s: %d\r\n", v1, r);
    r = myStrLen (v2);      printf ("strlen > %s: %d\r\n", v2, r);
    r = myStrLen (NULL);    printf ("strlen > NULL: %d\r\n", r);

    return (0);
}

```

Escriba y guarde el código anterior en un archivo llamado ejemplo01.c

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo01.c -Wall -oejemplo01.out
jerome@linuxVm:~$ ./ejemplo01.out
strlen > Hola: 4
strlen > : 0
strlen > Hola como te va? : 16
strlen > NULL : 0

```

2. Programa en el que se instancia dinámicamente un vector de 10 elementos de tipo int. Estos se inicializan con los números del 0 al 9. Se utiliza la función `qsort` para ordenar el vector de mayor a menor. Se imprime el vector antes y después de llamar a la función `qsort`. Finalmente se libera memoria.

```

#include <stdio.h>
#include <stdlib.h>

#define CANT ((int) 10)

int comparaInt (const void *a, const void *b)
{
    int *a_, *b_;

    a_ = (int *)a;
    b_ = (int *)b;
    return (*b_ - *a_);
}

int main (void)
{
    int *p;
    int i;

```

```

//-- Pido memoria para CANT elementos --
p = (int*)malloc (sizeof (*p) * CANT);
if (p == NULL) {
    printf ("Error\r\n");
    return (-1);
}

//-- Inicializo la zona de memoria asignada con cero--
for (i = 0; i < CANT; i++) {
    *(p + i) = i;
}

//-- Imprimo el vector --
printf ("Antes de qsort\r\n");
for (i = 0; i < CANT; i++) {
    printf ("%d\r\n", *(p + i));
}

//-- Ordeno --
qsort (p, CANT, sizeof (*p), comparaInt);

//-- Imprimo el vector --
printf ("\r\nDespues de qsort\r\n");
for (i = 0; i < CANT; i++) {
    printf ("%d\r\n", *(p + i));
}

//-- Libero memoria --
free (p);

return (0);
}

```

3. Programa en el que se instancia dinámicamente un vector de 10 elementos de tipo `int`. Estos se inicializan con los números del 0 al 9. Se utiliza el algoritmo del burbujeo para ordenar el vector de mayor a menor. Se imprime el vector antes y después de ordenar. Finalmente se libera memoria.

```

#include <stdio.h>
#include <stdlib.h>

#define CANT ((int) 10)

int main (void)
{
    int *p;
    int i, j;
    int aux;

```

```

    //-- Pido memoria para CANT elementos --
    p = (int*)malloc (sizeof (*p) * CANT);
    if (p == NULL) {
        printf ("Error\r\n");
        return (-1);
    }

    //-- Inicializo la zona de memoria asignada con cero --
    for (i = 0; i < CANT; i++) {
        *(p + i) = i;
    }

    //-- Imprimo el vector --
    printf ("Antes de ordenar\r\n");
    for (i = 0; i < CANT; i++) {
        printf ("%d\r\n", *(p + i));
    }

    //-- Ordeno --
    for (i = 0; i < CANT - 1; i++) {
        for (j = i + 1; j < CANT; j++) {
            if (*(p + i) < *(p + j)) {
                aux = *(p + i);
                *(p + i) = *(p + j);
                *(p + j) = aux;
            }
        }
    }

    //-- Imprimo el vector --
    printf ("\r\nDespues de ordenar\r\n");
    for (i = 0; i < CANT; i++) {
        printf ("%d\r\n", *(p + i));
    }

    //-- Libero memoria --
    free (p);
    return (0);
}

```

## Ejercicios

En los ejercicios que solicite implementar una función genera además un `main` que demuestre de forma automática el funcionamiento de la función.

1. Implemente un programa que le pida al usuario que ingrese letras y las almacene en memoria, el fin del ingreso de datos ocurre cuando el usuario ingresa el signo de admiración. Luego se deberán imprimir todas las letras ingresadas por el usuario en orden alfabético. Utilice la función `qsort`.
2. Implemente un programa que le pida al usuario que ingrese letras y las almacene en memoria, el fin del ingreso de datos ocurre cuando el usuario ingresa el signo de admiración. Luego se deberán imprimir todas las letras ingresadas por el usuario en orden alfabético. Para ordenar coloque el algoritmo burbujeo en una función y utilicela para ordenar.
3. Implemente una función que me indique si el string ingresado contiene solo letras o solo números. El prototipo de la función es el siguiente:

```
int validaString (char *dataPtr);
```

Devuelve

- 1 si el string contiene solo letras .
- 2 si el string contiene sólo números.
- 0 Si no es ninguna de las anteriores.

4. Implemente una función que cuente la ocurrencia de cada carácter (histograma) de un string pasado como parámetro.

```
void contarCaracteres (char *dataPtr, int *dataCntPtr);
```

Donde:

`dataPtr`: Es el puntero al string en el que hay que contar la ocurrencia de cada carácter.

`dataCntPtr`: Es el puntero un vector de 256 enteros en el que se lleva la cuenta de los caracteres del string

Ejemplo:

El string apuntado por `dataPtr` es "11AB1B1ZZZZ1"

- La posición 65 (65 es el ASCII de la 'A') del vector apuntado por `dataCntPtr` debe tener el número 1 (Cantidad de 'A' en el string)
  - La posición 66 (66 es el ASCII de la 'B') del vector apuntado por `dataCntPtr` debe tener el número 2 (Cantidad de 'B' en el string)
  - La posición 90 (90 es el ASCII de la 'Z') del vector apuntado por `dataCntPtr` debe tener el número 4 (Cantidad de 'Z' en el string)
  - La posición 49 (49 es el ASCII de la '1') del vector apuntado por `dataCntPtr` debe tener el número 5 (Cantidad de '1' en el string)
  - El resto de los elementos del vector deberán estar en cero.
5. Implemente una función que cuente la ocurrencia de cada carácter (histograma) de un string pasado como parámetro.

```
int* contarCaracteres (char *dataPtr);
```

Donde:

`dataPtr`: Es el puntero al string en el que hay que contar la ocurrencia de cada carácter.

Devuelve un puntero vector de 256 enteros en el que se lleva la cuenta de los caracteres del string, el cual es generado dinámicamente dentro de la función.

Ejemplo:

El string apuntado por dataPtr es "11AB1B1ZZZZ1"

- La posición 65 (65 es el ASCII de la 'A') del vector apuntado por dataCntPtr debe tener el número 1 (Cantidad de 'A' en el string)
- La posición 66 (66 es el ASCII de la 'B') del vector apuntado por dataCntPtr debe tener el número 2 (Cantidad de 'B' en el string)
- La posición 90 (90 es el ASCII de la 'Z') del vector apuntado por dataCntPtr debe tener el número 4 (Cantidad de 'Z' en el string)
- La posición 49 (49 es el ASCII de la '1') del vector apuntado por dataCntPtr debe tener el número 5 (Cantidad de '1' en el string)
- El resto de los elementos del vector deberán estar en cero.

6. Implemente una función que analice un párrafo de texto y devuelva la cantidad de caracteres y palabras que contiene. El prototipo de la función es el siguiente:

```
int analizaString (char *dataPtr, int *palabrasCant,  
                  int *caracteresCant);
```

Donde:

- dataPtr es el puntero al string a analizar
- palabrasCant es un puntero a una variable donde se almacenará la cantidad de palabras del texto
- caracteresCant es un puntero a una variable donde se almacenará la cantidad de caracteres del texto

Devuelve

- 0 si el string contiene solo letras y signos de puntuación.
- -1 en caso contrario.

Notas de implementación:

- Los caracteres a contar son todas las letras del alfabeto, números, signos de puntuación y espacios.
- Una palabra se separa de otra por un espacio o signo de puntuación.
- No cuente espacios para obtener la cantidad de palabras, ya que el texto puede contener dos espacios seguidos y esto dará un conteo incorrecto.

7. Implemente una función que determine si una palabra pasada como parámetro es palindromo

Prototipo de la función

```
int detectorPalindromo (char *palabraPtr)
```

Donde:

- palabraPtr: es el puntero a la palabra a analizar

Devuelve:

- 0 si la palabra es palindromo
- -1 si la palabra no es palindromo
- -2 si la palabra tiene menos de 2 caracteres

8. Implemente una función que elimine de un string el carácter pasado como parámetro. Tenga en cuenta que al eliminar caracteres deberá ajustar la posición del '\0'. El prototipo de la función es el siguiente.

```
int eliminarCaracter (char *dataPtr, char c);
```

Donde:

dataPtr: Es el puntero al string a modificar.

c: Carácter a eliminar.

Devuelve:

- La cantidad de caracteres eliminados.

9. Implemente una función que valide números de tarjeta de crédito utilizando el algoritmo de Luhn. El prototipo de la función es el siguiente:

```
int luhnAlg (char *tarjeta);
```

Parámetros

- tarjeta: puntero al vector que contiene el número de la tarjeta de crédito terminado en '\0'

Devuelve:

- Cero o un número positivo indicando que la tarjeta es válida.
- -1: Cuando la cantidad de dígitos de la tarjeta es distinto que 16.
- -2: Indica que el número de tarjeta es inválido.

Algoritmo de Luhn

- Multiplique los dígitos que se encuentran en la posición par del vector por dos, si el resultado es mayor o igual que diez se suman cada uno de los dígitos.
- Multiplique los dígitos que se encuentran en la posición impar del vector por uno.
- Suma todos los resultados obtenidos en los puntos a y b, obteniendo la suma llamada S. Si el módulo 10 de la suma obtenida S es igual a cero, el número de tarjeta es válido.

Ejemplo:

Dígitos Tarjeta	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	7	
Valor a multiplicar Por dígito	X2	X1	X2	X1	X2	X1	X2	X1	X2	X1	X2	X1	X2	X1	X2	X1	
Resultado de la multiplicación	0	1	4	3	8	5	12	7	16	9	0	1	4	3	8	7	
Dígitos a sumar	0	1	4	3	8	5	1+2	7	1+6	9	0	1	4	3	8	7	= 70

$70 \% 10 = 0 \Rightarrow$  La tarjeta es válida

10. Implemente una función que convierta un número positivo hexadecimal de 4 dígitos almacenado en un string y devuelva el correspondiente número decimal. Las letras del número hexadecimal están en mayúscula.

El prototipo de la función es:

```
int hexaToDec (char *dataPtr);
```

Devuelve:

- El valor hexadecimal en decimal
- -1 Si hay un símbolo que no corresponda a un número hexadecimal.
- -2 Si la cantidad de dígitos es distinta a 4

Ejemplos

```
hexaToDec ("0001"); // Devuelve 1
hexaToDec ("CAFE"); // Devuelve 51966
hexaToDec ("JJJJ"); // Devuelve -1
hexaToDec ("1");    // Devuelve -2
```

11. Implemente una función que convierta en binario un número entero positivo pasado como parámetro. El prototipo de la función es

```
char* imprimirBinario (int n);
```

Donde:

n número a imprimir en binario.

Devuelve NULL si el número es menor que cero, en caso contrario devuelve el puntero a un string con el número binario.

12. Implemente una función que calcule el promedio de las notas pasadas por un string separadas por espacios. La última nota tiene un espacio también. El prototipo de la función es el siguiente

```
float calcularPromedio (char *dataPtr)
```

Si el string pasado como parámetro está mal formado devuelva NAN

Ejemplos:

```
calcularPromedio("Hola");           // Devuelve NAN
calcularPromedio("");               // Devuelve NAN
calcularPromedio("1 2 3 12");       // Devuelve NAN
calcularPromedio("0 1 2 3 4 5 6 7 8 9 "); // Devuelve 4.5
```

Notas de implementación:

- Esta función solamente calcula el promedio de números de un dígito separados por espacio.

13. Implemente una función que busque dentro de un string todas las ocurrencias de una palabra pasada como parámetro y las reemplace por otra palabra. El prototipo de la función es el siguiente:

```
int reemplazaPalabra (char *dataPtr, char *palabraBuscar,
                     char *palabraNueva);
```

Donde:

dataPtr: Es un puntero al string donde se realizan los reemplazos

palabraBuscar: Es un puntero a la palabra a buscar.

palabraNueva: Es un puntero a la palabra a reemplazar

Devuelve

- un número positivo indicando la cantidad de palabras reemplazadas..
- -1 en caso de que palabraBuscar y palabraNueva tengan distinto tamaño.
- -2 en caso de que palabraBuscar y palabraNueva sean '\0'.

14. Implemente una función que determine si un password cumple con todas las recomendaciones de seguridad que se describen a continuación:

- Debe tener al menos 8 caracteres.
- No debe tener espacios.
- Debe contener letras mayúsculas y minúsculas.
- Debe contener alguno de estos símbolos ~!@#\$%^&\* \_-+ ='|\\()\[\];: '"<>.,? /
- Debe contener un dígito base 10. (0 - 9)

El prototipo de la función es el siguiente

```
int validarPassword(char *dataPtr)
```

Donde:

- dataPtr: Es el puntero al password a validar.
- Devuelve:
  - 1: El password cumple las cuatro recomendaciones de seguridad.
  - 1: Si no cumple la recomendación a
  - 2: Si no cumple la recomendación b
  - 3: Si no cumple la recomendación c
  - 4: Si no cumple la recomendación d
  - 5: Si no cumple la recomendación e

15. Implemente una función que obtenga el dígito verificador de un número de CUIT pasado como parámetro, el cálculo se realiza utilizando el algoritmo módulo11. El prototipo de la función es el siguiente:

```
int cuitValida (char *cuit);
```

Parámetros

- cuit: puntero al vector que contiene el número de CUIT terminado en '\0'

Devuelve:

- Un número positivo indicando el dígito verificador.
- 1: Cuando la cantidad de dígitos es distinto de 10
- 2: Indica que el número de CUIT es inválido (contiene algo distinto a números)

Algoritmo módulo 11

- Multiplique los dígitos desde el menos significativo por la serie 2,3,4,5,6,7.
- Sume el resultado de las multiplicaciones anteriores.
- Calcule el módulo 11 de la suma anterior.
- Calcule 11 menos el resultado anterior, si el resultado es menor que 10 lo obtenido es el dígito verificador. En cambio si vale 10 el dígito verificador es 9. Si vale 11 el dígito verificador es 0

Ejemplo:

CUIT	2	0	1	2	3	4	5	6	7	8	Suma	%11	Dígito
Valor a multiplicar Por dígito	X5	X4	X3	X2	X7	X6	X5	X4	X3	X2		148%11	11-5
Resultado de la multiplicación	10	0	3	4	21	24	25	24	21	16	=148	=5	6



## 18. Modelo de memoria

### Tipos de datos (para 64 bits)

Tipo de dato	Descripción	Cantidad de bytes (sizeof)	Rango
char	Carácter	1	-128 a 127
unsigned char	Carácter	1	0 a 255
short	Número entero	2	-32768 a 32767
unsigned short	Número entero	2	0 a 65535
int	Número entero	4	-2147483648 a 2147483647
unsigned int	Número entero	4	0 a 4294967295
float	Número real	4	3.4E-38 a 3.4E+38
double	Número real	8	1.7E-308 a 1.7E+308
void*; int*; float*; char*	Punteros	8	---

### Ámbito de uso de una variable. (scope de una variable)

El ámbito de uso (scope) de una variable es la zona o parte del programa donde es posible accederla.

1. Por ejemplo en el siguiente fragmento de código la variable
  - a. La variable suma y el parámetro param son solo accesibles dentro de la función funcSuma1
  - b. Las variables aux y res son solo accesibles dentro de la función main.

```
#include <stdio.h>

int funcSuma1 (int param)
{
    int suma;

    suma = param + 1;

    return (suma);
}

int main (void)
{
    int num, res;

    printf ("Ingrese numero\r\n");
    scanf ("%d", &num);
```

```
    res = funcSuma1(num);
    printf ("Ingrese numero\r\n");

    return (0);
}
```

**Advertencia:**

Nunca devuelva la dirección de una variable local o parametro de una función

```
int * funcMal (void) {
    int MAL = 10;
    return (&MAL);
}
```

2. Por ejemplo en el siguiente fragmento de código la variable

- El parámetro param son solo accesibles dentro de la función funcSuma1
- Las variables aux y res son solo accesibles dentro de la función main.
- La variable suma es global y es accesible en todas las funciones del archivo .c en este caso es accesible por funcSuma1 y por la función main

```
#include <stdio.h>

int suma;

int funcSuma1 (int param)
{
    suma = param + 1;

    return (suma);
}

int main (void)
{
    int num, res;

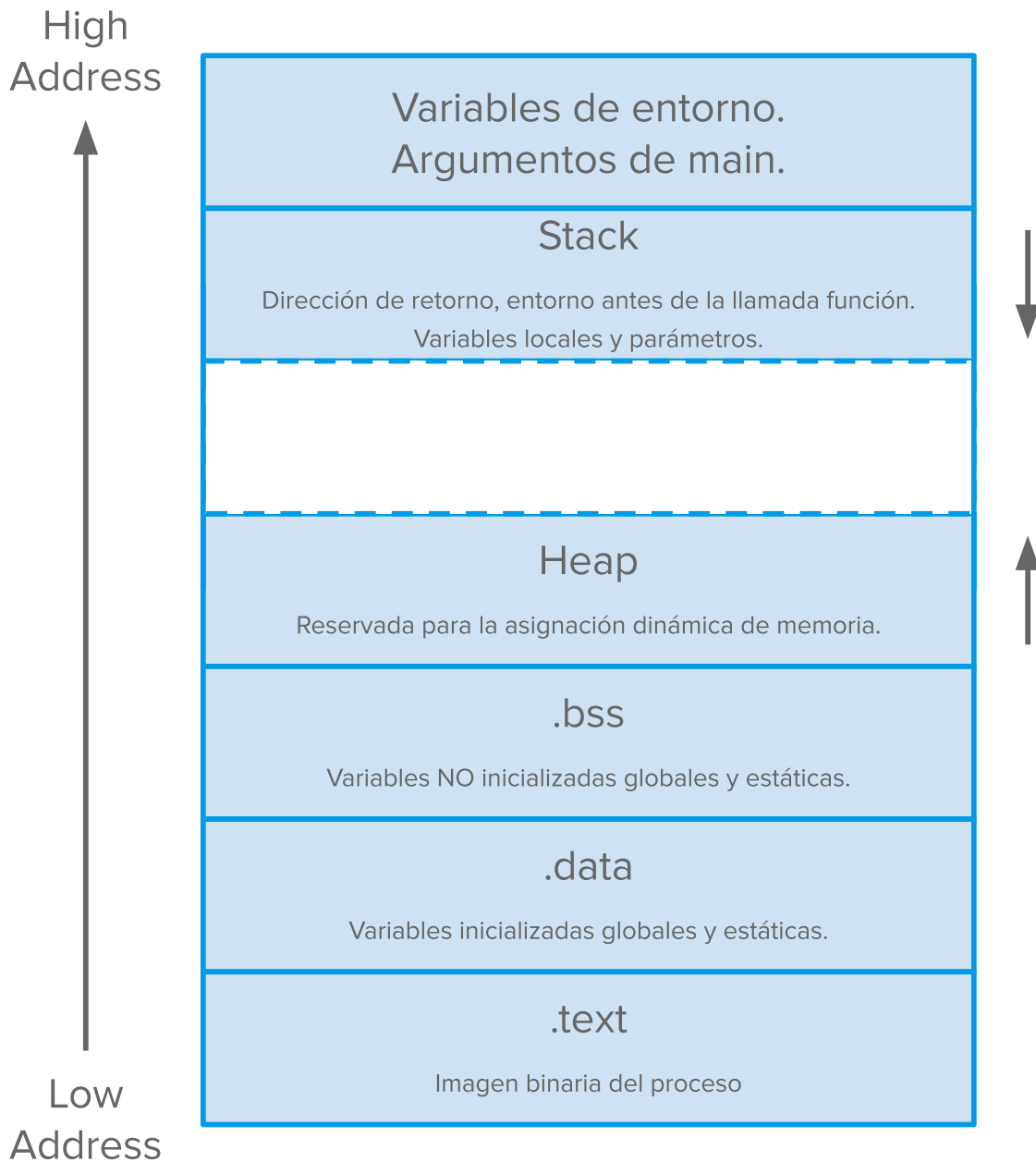
    printf ("Ingrese numero\r\n");
    scanf ("%d", &num);
    res = funcSuma1 (num);
    printf ("res = %d\r\n", res);
    printf ("suma = %d\r\n", suma);

    return (0);
}
```

**Advertencia:**

Evite en lo posible el uso de variables globales.

## Mapa de memoria



## Modificadores de variables

- **auto:** Todas las variables son auto salvo que se indique lo contrario. Estas variables son alojadas en el stack. Las variables de este tipo dejan de existir una vez que salimos de la función donde fueron definidas.
- **extern:** Se usa en variables globales, para que las mismas puedan ser accedidas fuera del .c en el cual están definidas.
- **static:** Su funcionamiento depende de si se usa en una variable local o global.
  - Si se usa en una variable local a una función el valor de la misma persiste por más que se haya salido de la función. Es como una variable local, pero solo puede ser usada dentro de

la función que fue definida. Al definir una variable local como static la misma debe ser inicializada en el momento que se define.

- Si se usa en una variable global, esta no puede ser accedida desde otro .c Incluso si se hace un extern la variable no podrá ser accedida desde otro .c
- **volatile**: Una variable declarada así le indica al compilador que no optimice la variable por que no encuentra que esté siendo usada en ese contexto.
- **const**: Al declararla de esta forma se evita que la variable sea modificada luego de inicializarla.
- **register**: Es una sugerencia para colocar la variable en un registro del procesador.

El siguiente muestra ejemplos de

```
#include <stdio.h>

int funcSuma1 (int param)
{
    static int suma = 0;    //-- Declaro una variable static --

    suma = param + 1;

    return (suma);
}

int main (void)
{
    int num, res;    //-- Variables auto --
    const int CONSTANTE = 10;    //-- Declaro una variable constante --

    printf ("Ingrese numero\r\n");
    scanf ("%d", &num);
    res = funcSuma1 (num);
    printf ("res = %d\r\n", res);

    return (0);
}
```

## Direcciones de memoria

La memoria es el lugar donde se almacenan variables y el programa, suele organizarse en bytes y a cada uno de estos se le asigna un número que lo identifica llamado dirección. Estas direcciones se escriben en hexadecimal ya que facilita su lectura. En el siguiente diagrama se muestra 4 posiciones de memoria de 1 byte que almacenan los números 0, 1, 2, 3 y a su izquierda las direcciones de cada uno de ellos

	1 Byte
0x7ffebdeb2f60	0
0x7ffebdeb2f61	1

0x7ffebdeb2f62

2

0x7ffebdeb2f63

3

En el código nunca colocamos posiciones de memoria fijas (salvo por el NULL) ya que cada vez que ejecutamos el código nuevamente el sistema operativo nos asignará una zona de memoria distinta. Las variables son la abstracción que usamos para referirnos a las zonas de memoria.

Para que sea más fácil ver los ejemplo vamos a simplificar los números de las direcciones y arriba vamos a indicar el ancho en bytes de la memoria, los siguientes ejemplo colocamos 4 bytes para que coincida con el tamaño de una variable int y sea más fácil de leer. Además no vamos a tener en cuenta el byte order de los datos almacenados en memoria.

A continuación se muestra un pequeño código y se indica en verde la línea donde se detuvo la ejecución (la línea en verde no fue ejecutada al momento de imprimir el mapa de memoria) a la derecha se observa un mapa de memoria con las dos variables que intervienen en el código.

```
#include <stdio.h>

int main (void)
{
    int num = 0, res = 0;

    printf ("Ingrese numero\r\n");
    scanf ("%d", &num);
    res = num + 1
    printf ("res = %d\r\n", num);

    return (0);
}
```

	4 Bytes	
0x7F00	0x00000000	num
0x7F04	0x00000000	res

Continuamos la ejecución del código del código ingresando el número decimal 4660 que en hexadecimal es 0x1234, este número se almacena en la variable num que esta en la posición de memoria 0x7F00

```
#include <stdio.h>

int main (void)
{
    int num = 0, res = 0;

    printf ("Ingrese numero\r\n");
    scanf ("%d", &num);
    res = num + 1
    printf ("res = %d\r\n", num);

    return (0);
}
```

	4 Bytes	
0x7F00	0x00001234	num
0x7F04	0x00001235	res

El siguiente ejemplo muestra el mapa de memoria con un puntero en el código, debe recordar que un puntero es una variable que almacena una dirección de memoria.

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

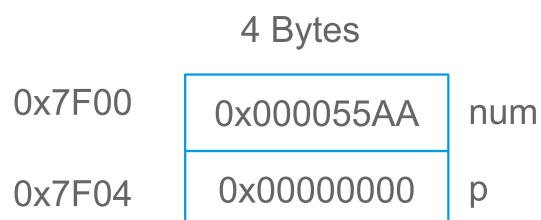
int main (void)
{
    int num = 0x55AA;
    int *p = NULL;

    p = &num;

    *p = 0x1234;

    num = 0x2233;

    return (0);
}
```



Luego de asignar el puntero. El puntero p apunta a la variable num.

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

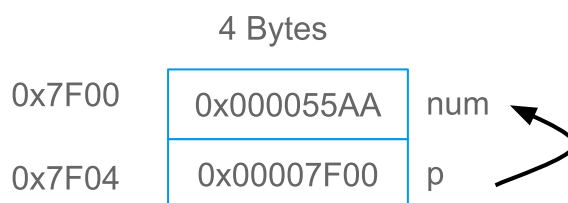
int main (void)
{
    int num = 0x55AA;
    int *p = NULL;

    p = &num;

    *p = 0x1234;

    num = 0x2233;

    return (0);
}
```



Cambiamos el valor almacenado en la variable num usando el puntero.

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

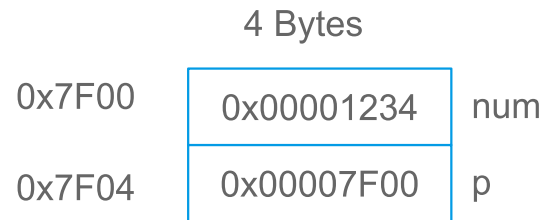
int main (void)
{
    int num = 0x55AA;
    int *p = NULL;

    p = &num;

    *p = 0x1234;

    num = 0x2233;

    return (0);
}
```



Finalmente cambio el valor de la variable

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

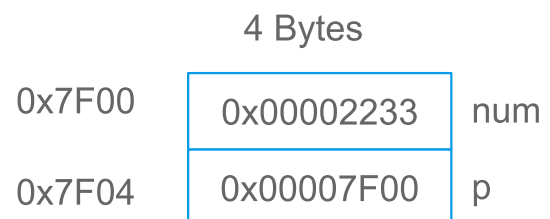
int main (void)
{
    int num = 0x55AA;
    int *p = NULL;

    p = &num;

    *p = 0x1234;

    num = 0x2233;

    return (0);
}
```

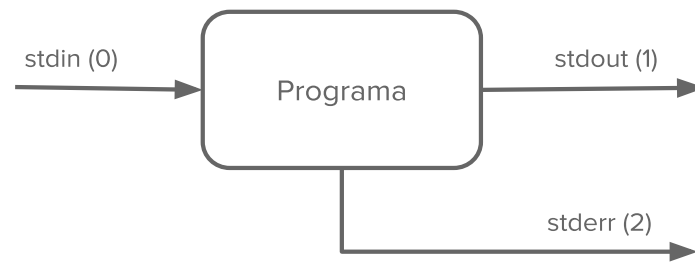


## Standard streams

Al ejecutarse la función main tiene 3 streams predefinidos abiertos:

- **stdin:** Stream de entrada estándar del programa, habitualmente redireccionado al teclado
- **stdout:** Stream de salida estándar del programa, habitualmente redireccionado a la terminal de texto.
- **stderr:** Stream de salida de errores del programa, habitualmente redireccionado a la terminal de texto.

Los tres están definidos en `stdio.h` cómo `FILE *stdin`; `FILE *stdout`; `FILE *stderr`;





## 19. Estructuras

### Operador

Operador	Descripción
sizeof()	Devuelve el tamaño del operando

### Ejemplos

1. Programa que crea una estructura que contiene los datos nombre, edad y sexo. Luego es inicializada en tiempo de ejecución y la imprime por pantalla.

```
#include <stdio.h>
#include <string.h>
#define NOMBRE_CANT ((int)16) //!< Cantidad de elementos del vector
nombre

struct persona {
    char nombre[NOMBRE_CANT];
    int edad;
    char sexo;
};

int main (void)
{
    struct persona estudiante;

    //-- Inicializo la estructura --
    strcpy (&estudiante.nombre[0], "Juan");
    estudiante.edad = 18;
    estudiante.sexo = 'M';

    //-- Imprimo la estructura --
    printf ("Nombre: %s\r\n", estudiante.nombre);
    printf ("Edad: %d\r\n", estudiante.edad);
    printf ("Sexo: %c\r\n", estudiante.sexo);

    return (0);
}
```

2. Programa que crea una estructura que contiene los datos nombre, edad y sexo. Luego es inicializada por el usuario y la imprime por pantalla.

```
#include <stdio.h>

#define NOMBRE_CANT ((int)16) //!< Cantidad de elementos del vector
nombre

struct persona {
    char nombre[NOMBRE_CANT];
    int edad;
    char sexo;
};

int main (void)
{
    struct persona estudiante;

    //-- Inicializo la estructura --
    printf ("Ingrese nombre:\r\n"); scanf ("%s", &estudiante.nombre[0]);
    printf ("Ingrese edad:\r\n");    scanf ("%d", &estudiante.edad);
    printf ("Ingrese sexo:\r\n"); scanf ("%c", &estudiante.sexo);

    //-- Imprimo la estructura --
    printf ("Nombre: %s\r\n", estudiante.nombre);
    printf ("Edad: %d\r\n", estudiante.edad);
    printf ("Sexo: %c\r\n", estudiante.sexo);

    return (0);
}
```

3. Programa que crea un vector de estructuras del tipo struct persona. Luego el vector es inicializado por el usuario y lo imprime por pantalla.

```
#include <stdio.h>

#define NOMBRE_CANT ((int)16) //!< Cantidad de elementos del vector nombre
#define CANT         ((int)3)  //!< Cantidad de elemento del vector estudiante

struct persona {
    char nombre[NOMBRE_CANT];
    int edad;
    char sexo;
};

int main (void)
{
    struct persona estudiante[CANT];
    int i;

    //-- Inicializo la estructura --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese nombre:\r\n");
        scanf ("%s", &(estudiante[i].nombre[0]));
        printf ("Ingrese edad:\r\n");
        scanf ("%d", &(estudiante[i].edad));
        printf ("Ingrese sexo:\r\n");
        scanf ("%*c%c", &(estudiante[i].sexo));
    }

    //-- Imprimo el vector de estructuras --
    for (i = 0; i < CANT; i++) {
        printf ("Nombre: %s\r\n", estudiante[i].nombre);
        printf ("Edad: %d\r\n", estudiante[i].edad);
        printf ("Sexo: %c\r\n", estudiante[i].sexo);
    }

    return (0);
}
```

4. Programa que crea un vector de estructuras del tipo struct persona. Luego el vector es inicializado utilizando un puntero y lo imprime por pantalla utilizando el mismo puntero.

```
#include <stdio.h>

#define NOMBRE_CANT ((int)16) //!< Cant de elementos del vector nombre
#define CANT        ((int)3)  //!< Cant de elementos del vector estudiante

struct persona {
    char nombre[NOMBRE_CANT];
    int edad;
    char sexo;
};

int main (void)
{
    struct persona estudiante[CANT];
    struct persona *p;
    int i;

    p = &estudiante[0];    //-- Inicializo el puntero --

    //-- Inicializo la estructura --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese nombre:\r\n");
        scanf ("%s", &((p + i)->nombre[0]));
        printf ("Ingrese edad:\r\n");
        scanf ("%d", &((p + i)->edad));
        printf ("Ingrese sexo:\r\n");
        scanf ("%*c%c", &((p + i)->sexo));
    }

    //-- Imprimo el vector de estructuras --
    for (i = 0; i < CANT; i++) {
        printf ("Nombre: %s\r\n", (p + i)->nombre);
        printf ("Edad: %d\r\n", (p + i)->edad);
        printf ("Sexo: %c\r\n", (p + i)->sexo);
    }

    return (0);
}
```

5. Programa que crea una estructura del tipo struct persona y un elemento de esta es otra estructura del tipo struct datos. Luego ambas son inicializadas por el usuario y lo imprime por pantalla.

```
#include <stdio.h>

#define NOMBRE_CANT ((int)16) //!< Cant de elementos del vector nombre
#define CANT        ((int)3)  //!< Cant de elementos del vector estudiante

struct datos {
    int edad;
    char sexo;
};

struct persona {
    char nombre[NOMBRE_CANT];
    struct datos eg;
};

int main (void)
{
    struct persona estudiante[CANT];
    int i;

    //-- Inicializo la estructura --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese nombre:\r\n");
        scanf ("%s", &(estudiante[i].nombre[0]));
        printf ("Ingrese edad:\r\n");
        scanf ("%d", &(estudiante[i].eg.edad));
        printf ("Ingrese sexo:\r\n");
        scanf ("%*c%c", &(estudiante[i].eg.sexo));
    }

    //-- Imprimo el vector de estructuras --
    for (i = 0; i < CANT; i++) {
        printf ("Nombre: %s\r\n", estudiante[i].nombre);
        printf ("Edad: %d\r\n", estudiante[i].eg.edad);
        printf ("Sexo: %c\r\n", estudiante[i].eg.sexo);
    }

    return (0);
}
```

6. Programa que crea un vector de estructuras del tipo struct persona y un elemento de esta es otra estructura del tipo struct datos. Luego el vector es inicializado utilizando un puntero y lo imprime por pantalla utilizando el mismo puntero.

```
#include <stdio.h>

#define NOMBRE_CANT ((int)16) //!< Cant de elementos del vector nombre
#define CANT        ((int)3)  //!< Cant de elementos del vector estudiante

struct datos {
    int edad;
    char sexo;
};

struct persona {
    char nombre[NOMBRE_CANT];
    struct datos eg;
};

int main (void)
{
    struct persona estudiante[CANT];
    struct persona *p;
    int i;

    p = &estudiante[0];    //-- Inicializo el puntero --

    //-- Inicializo la estructura --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese nombre:\r\n");
        scanf ("%s", &((p + i)->nombre[0]));
        printf ("Ingrese edad:\r\n");
        scanf ("%d", &((p + i)->eg.edad));
        printf ("Ingrese sexo:\r\n");
        scanf ("%*c%c", &((p + i)->eg.sexo));
    }

    //-- Imprimo el vector de estructuras --
    for (i = 0; i < CANT; i++) {
        printf ("Nombre: %s\r\n", (p + i)->nombre);
        printf ("Edad: %d\r\n", (p + i)->eg.edad);
        printf ("Sexo: %c\r\n", (p + i)->eg.sexo);
    }
    return (0);
}
```

7. Programa que crea un vector de estructuras del tipo struct persona cuyos elementos son un vector para almacenar el nombre y un vector de estructuras de tipo struct finales.

```
#include <stdio.h>

#define NOMBRE_CANT ((int)16) //!< Cant de elementos del vector nombre
#define CANT ((int)3) //!< Cant de elementos del vector estudiante
#define FINALES_CANT ((int)16) //!< Cant de elementos del vector materia

struct finales {
    char materia[MATERIA_CANT];
    int nota;
};

struct persona {
    char nombre[NOMBRE_CANT];
    struct finales final[FINALES_CANT];
};

int main (void)
{
    struct persona estudiante[CANT];
    int i, j;
    p = &estudiante[0];    //-- Inicializo el puntero --
    //-- Inicializo la estructura --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese nombre:\r\n");
        scanf ("%s", &(estudiante[i].nombre[0]));
        for (j = 0; j < CANT; j++) {
            printf ("Ingrese materias:\r\n");
            scanf ("%s", &((p + i)->final[j].materia[0]));
            printf ("Ingrese nota:\r\n");
            scanf ("%d", &((p + i)->final[j].nota));
        }
    }

    //-- Imprimo el vector de estructuras --
    for (i = 0; i < CANT; i++) {
        printf ("Nombre: %s\r\n", (p + i)->nombre);
        for (j = 0; j < FINALES_CANT; j++) {
            printf ("Materia: %s\r\n", (p + i)->final[j].materia);
            printf ("Nota: %d\r\n", (p + i)->final[j].nota);
        }
    }
    return (0);
}
```

8. Programa que crea un vector de estructuras del tipo struct persona cuyos elementos son un vector para almacenar el nombre y un vector de estructuras de tipo struct finales. Utilizando un puntero.

```
#include <stdio.h>

#define NOMBRE_CANT ((int)16) //!< Cant de elementos del vector nombre
#define CANT ((int)3) //!< Cant de elementos del vector estudiante
#define FINALES_CANT ((int)16) //!< Cant de elementos del vector materia

struct finales {
    char materia[MATERIA_CANT];
    int nota;
};

struct persona {
    char nombre[NOMBRE_CANT];
    struct finales final[FINALES_CANT];
};

int main (void)
{
    struct persona estudiante[CANT], *p;
    int i, j;

    //-- Inicializo la estructura --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese nombre:\r\n");
        scanf ("%s", &(estudiante[i].nombre[0]));
        for (j = 0; j < CANT; j++) {
            printf ("Ingrese materias:\r\n");
            scanf ("%s", &(estudiante[i].final[j].materia[0]));
            printf ("Ingrese nota:\r\n");
            scanf ("%d", &(estudiante[i].final[j].nota));
        }
    }

    //-- Imprimo el vector de estructuras --
    for (i = 0; i < CANT; i++) {
        printf ("Nombre: %s\r\n", estudiante[i].nombre);
        for (j = 0; j < FINALES_CANT; j++) {
            printf ("Materia: %s\r\n", estudiante[i].final[j].materia);
            printf ("Nota: %d\r\n", estudiante[i].final[j].nota);
        }
    }
    return (0);
}
```



9. Programa que crea una estructura que almacena edad, sexo y un puntero a un vector creado dinámicamente para almacenar el nombre.

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

#define NOMBRE_CANT ((int)16) //!< Cantidad maxima de elementos del nombre.

struct persona {
    char *nombre;
    int edad;
    char sexo;
};

int main (void)
{
    struct persona estudiante;
    char nombre[NOMBRE_CANT];

    //-- Inicializo la estructura --
    printf ("Ingrese nombre:\r\n"); scanf ("%s", &nombre[0]);
    estudiante.nombre = (char *)malloc (strlen (nombre) + 1);
    if (estudiante.nombre == NULL) {
        return (-1);
    } else {
        strcpy (estudiante.nombre, nombre);
    }
    printf ("Ingrese edad:\r\n");    scanf ("%d", &estudiante.edad);
    printf ("Ingrese sexo:\r\n"); scanf ("%c", &estudiante.sexo);

    //-- Imprimo la estructura --
    printf ("Nombre: %s\r\n", estudiante.nombre);
    printf ("Edad: %d\r\n", estudiante.edad);
    printf ("Sexo: %c\r\n", estudiante.sexo);

    return (0);
}
```

## 10. Programa que demuestra el uso de sizeof

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

#define CANT ((int)10) //!< Cantidad.

struct data{
    char *ptr;
    int varInt;
    int vectInt[CANT];
};

int main (void)
{
    int varInt, vectInt[CANT], *ptrInt;
    char varChar, vectChar[CANT], *ptrChar;
    struct data varStruct, vectStruct[CANT], *ptrStruct;

    printf ("sizeof(varInt): %d\r\n", sizeof(varInt));
    printf ("sizeof(vectInt): %d\r\n", sizeof(vectInt));
    printf ("sizeof(vectInt[0]): %d\r\n", sizeof(vectInt[0]));
    printf ("sizeof(ptrInt): %d\r\n", sizeof(ptrInt));
    printf ("sizeof(*ptrInt): %d\r\n", sizeof(*ptrInt));

    printf ("sizeof(varChar): %d\r\n", sizeof(varChar));
    printf ("sizeof(vectChar): %d\r\n", sizeof(vectChar));
    printf ("sizeof(vectChar[0]): %d\r\n", sizeof(vectChar[0]));
    printf ("sizeof(ptrChar): %d\r\n", sizeof(ptrChar));
    printf ("sizeof(*ptrChar): %d\r\n", sizeof(*ptrChar));

    printf ("sizeof(varStruct): %d\r\n", sizeof(varStruct));
    printf ("sizeof(vectStruct): %d\r\n", sizeof(vectStruct));
    printf ("sizeof(vectStruct[0]): %d\r\n", sizeof(vectStruct[0]));
    printf ("sizeof(ptrStruct): %d\r\n", sizeof(ptrStruct));
    printf ("sizeof(*ptrStruct): %d\r\n", sizeof(*ptrStruct));

    printf ("sizeof(varStruct.ptr): %d\r\n", sizeof(varStruct.ptr));
    printf ("sizeof(varStruct.varInt): %d\r\n", sizeof(varStruct.varInt));
    printf ("sizeof(varStruct.vectInt): %d\r\n", sizeof(varStruct.vectInt));

    return (0);
}
```

Al ejecutar el código se observa

```
jerome@linuxVm:~$ ./ejemplo18_10.out
sizeof(varInt): 4
sizeof(vectInt): 40
sizeof(vectInt[0]): 4
sizeof(ptrInt): 8
sizeof(*ptrInt): 4
sizeof(varChar): 1
sizeof(vectChar): 10
sizeof(vectChar[0]): 1
sizeof(ptrChar): 8
sizeof(*ptrChar): 1
sizeof(varStruct): 56
sizeof(vectStruct): 560
sizeof(vectStruct[0]): 56
sizeof(ptrStruct): 8
sizeof(*ptrStruct): 56
sizeof(varStruct.ptr): 8
sizeof(varStruct.varInt): 4
sizeof(varStruct.vectInt): 40
```

## 11. Programa que demuestra

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

struct data{
    int varInt0;
    char varChar0;
    int varInt1;
    char varChar1;
};

struct dataAlign {
    int varInt0;
    int varInt1;
    char varChar0;
    char varChar1;
};

int main (void)
{
    struct data varStruct;
    struct dataAlign varStructAlign;

    printf ("sizeof(varStruct): %ld\r\n", sizeof(varStruct));
    printf ("sizeof(varStructAlign): %ld\r\n", sizeof(varStructAlign));

    return (0);
}
```

Al ejecutar el código se observa

```
jerome@linuxVm:~/jatencio$ ./ejemplo18_11.out
sizeof(varStruct): 16
sizeof(varStructAlign): 12
```

**Advertencia:**

El sizeof de una estructura no siempre coincide con la sumatoria de los sizeof de cada campo, esto se debe a la alineación de los campo de la estructura.

## Ejercicios

1. Implemente un programa le pida al usuario datos para rellenar un vector de 10 estructuras del tipo struct persona. Este vector de estructuras debe ser creado dinamicamente usando malloc. Se sabe que el nombre ingresado por el usuario no tiene más de 16 caracteres incluyendo el '\0'. Para almacenar el nombre ingresado por el usuario deberá utilizar malloc reservando exactamente la cantidad de caracteres del nombre ingresado más el espacio para el '\0'. Al finalizar el ingreso imprima todo el vector y libere la memoria reservada antes de finalizar el programa.

```
struct persona {
    char *nombre;
    int edad;
    char sexo;
};
```

2. Implemente una función con el siguiente prototipo que realice el ingreso de datos como el ejercicio anterior.

```
struct persona* ingreso (int dataCant);
```

Donde:

dataCant: Es la cantidad de estructuras a reservar.

La función devuelve un puntero al vector de estructuras creado dinamicamente.

3. Implementar una función que imprima los datos almacenados en el vector de estructuras pasado como parámetro.

```
void imprimir (struct persona *p, int dataCant);
```

Donde:

p: Puntero al vector de estructuras.

dataCant: Es la cantidad de estructuras apuntadas por p.

4. Implementar una función que libere la memoria reservada.

```
void liberar (struct persona *p, int dataCant);
```

Donde:

p: Puntero al vector de estructuras a liberar.

dataCant: Es la cantidad de estructuras apuntadas por p.

5. Implemente una función que realice el ordenamiento de un vector de estructuras de tipo struct persona por nombre. El prototipo de la función es el siguiente.

```
void ordenar(struct persona *p, int dataCant);
```

Donde:

p: Puntero al vector de estructuras.

dataCant: Es la cantidad de estructuras apuntadas por p.

6. Implemente una función con el siguiente prototipo que realice el ingreso de datos para las estructuras del los siguientes tipos.

```
struct finales {  
    char materia[16];  
    int nota;  
};  
struct estudiante {  
    char nombre[16]  
    int cantFinales;  
    struct finales *final;  
};
```

```
struct estudiante* ingreso (int dataCant);
```

Donde:

dataCant: Es la cantidad de estructuras a reservar.

El usuario ingresara el nombre del estudiante y luego para cada uno ingresara el nombre de la materia y la nota del final. La carga de los finales termina cuando se ingresa una materia con nota igual a -1.

El elemento cantFinales indica la cantidad de elementos del vector apuntado por el puntero final.

Utilice realloc para la realización de este ejercicio.

7. Implementar una función que imprima los datos almacenados en el vector de estructuras pasado como parámetro.

```
void imprimir (struct estudiante *p, int dataCant);
```

Donde:

p: Puntero al vector de estructuras.

dataCant: Es la cantidad de estructuras apuntadas por p.

8. Implementar una función que libere la memoria reservada.

```
void liberar (struct estudiante *p, int dataCant);
```

Donde:

p: Puntero al vector de estructuras a liberar.

dataCant: Es la cantidad de estructuras apuntadas por p.

9. Implemente una función que realice el ordenamiento de un vector de estructuras de tipo struct estudiante por cantidad de finales. El prototipo de la función es el siguiente.

```
void ordenar(struct estudiante *p, int dataCant);
```

Donde:

p: Puntero al vector de estructuras.

dataCant: Es la cantidad de estructuras apuntadas por p

10. Implemente una función que realice una búsqueda por nombre y materia en un vector de estructuras de tipo struct estudiante. El prototipo de la función es el siguiente.

```
int buscar(struct estudiante *p, int dataCant, char *nombre,  
          char *materia);
```

Donde:

p: Puntero al vector de estructuras.

dataCant: Es la cantidad de estructuras apuntadas por p

nombre: Puntero del nombre a buscar

materia: Puntero de la materia a buscar.

Devuelve:

-1: Si no encontró el nombre

-2: Si no encontró la materia

La nota si encontró el nombre y la materia.



## 20. Archivos I

### Funciones utilizadas de stdio.h

Función	Descripción
fopen	Abre un archivo.
fprintf	Escribe un texto en un archivo.
fgets	Lee un string de un archivo.
fscanf	Lee de un archivo según el formato indicado.
feof	Indica si llegamos al final del archivo
fclose	Cierra un archivo.

### Modos de apertura de un archivo

Modo	Descripción	Posición inicial
r	Abre el archivo para lectura	Al principio del archivo
r+	Abre el archivo para lectura y escritura	Al principio del archivo
w	Abre el archivo para escritura Trunca el archivo a tamaño cero o Crea el archivo si no existe	Al principio del archivo
w+	Abre el archivo para escritura Trunca el archivo a tamaño cero o Crea el archivo si no existe	Al principio del archivo
a	Abre el archivo para agregar. Crea el archivo si no existe.	Al final del archivo
a+	Abre el archivo para leer y agregar. Crea el archivo si no existe.	Al final del archivo

### Programas necesarios

Use apt para instalarlos

Paquete	Descripción
okteta	Editor de archivos raw. Permite verlos en hexa, octal, ascii, etc
hexdump	Muestra el contenido de un archivo en hexa, octal, ascii, etc

## Comandos

Comando	Descripción
cat	Imprime un archivo por stdout (pantalla)
head	Imprime las primeras líneas de un archivo por stdout (pantalla)
tail	Imprime las últimas líneas de un archivo por stdout (pantalla)

### Ejemplo de uso de comandos.

Realizamos un ls de / y lo almacenamos en el archivo list.txt.

```
jerome@linuxVm:~$ ls / -las > list.txt
```

Podemos verificar el contenido del archivo list.txt utilizando un editor de texto (por ejemplo el atom)

Ejecutamos el comando cat sobre list.txt, para ver todo el contenido en la pantalla.

```
jerome@linuxVm:~$ cat list.txt
total 104
4 drwxr-xr-x 23 root root 4096 abr 2 21:22 .
4 drwxr-xr-x 23 root root 4096 abr 2 21:22 ..
4 drwxr-xr-x 2 root root 4096 abr 2 21:23 bin
4 drwxr-xr-x 3 root root 4096 abr 5 11:01 boot
4 drwxrwxr-x 2 root root 4096 abr 2 21:15 cdrom
0 drwxr-xr-x 18 root root 3920 may 10 15:33 dev
12 drwxr-xr-x 122 root root 12288 may 10 15:33 etc
4 drwxr-xr-x 3 root root 4096 abr 2 21:21 home
0 lrwxrwxrwx 1 root root 33 abr 2 21:22 initrd.img ->
boot/initrd.img-4.10.0-28-generic
4 drwxr-xr-x 20 root root 4096 abr 2 21:23 lib
4 drwxr-xr-x 2 root root 4096 abr 3 04:07 lib64
16 drwx----- 2 root root 16384 abr 2 21:07 lost+found
4 drwxr-xr-x 3 root root 4096 abr 2 22:22 media
4 drwxr-xr-x 2 root root 4096 ago 1 2017 mnt
4 drwxr-xr-x 3 root root 4096 abr 5 10:59 opt
0 dr-xr-xr-x 137 root root 0 may 10 15:32 proc
4 drwx----- 3 root root 4096 may 10 15:33 root
0 drwxr-xr-x 22 root root 680 may 10 15:38 run
12 drwxr-xr-x 2 root root 12288 abr 5 11:01 sbin
4 drwxr-xr-x 2 root root 4096 ago 1 2017 srv
0 dr-xr-xr-x 13 root root 0 may 10 15:38 sys
4 drwxrwxrwt 9 root root 4096 may 10 15:38 tmp
4 drwxr-xr-x 10 root root 4096 ago 1 2017 usr
4 drwxr-xr-x 13 root root 4096 ago 1 2017 var
0 lrwxrwxrwx 1 root root 30 abr 2 21:22 vmlinuz ->
boot/vmlinuz-4.10.0-28-generic
```

Muestra las últimas 5 líneas del archivo list.txt



```
jerome@linuxVm:~$ tail -n5 list.txt
0 dr-xr-xr-x 13 root root 0 may 10 15:38 sys
4 drwxrwxrwt 9 root root 4096 may 10 15:38 tmp
4 drwxr-xr-x 10 root root 4096 ago 1 2017 usr
4 drwxr-xr-x 13 root root 4096 ago 1 2017 var
0 lrwxrwxrwx 1 root root 30 abr 2 21:22 vmlinuz ->
boot/vmlinuz-4.10.0-28-generic
```

Muestra las primeras 5 líneas del archivo list.txt

```
jerome@linuxVm:~$ head -n5 list.txt
total 104
4 drwxr-xr-x 23 root root 4096 abr 2 21:22 .
4 drwxr-xr-x 23 root root 4096 abr 2 21:22 ..
4 drwxr-xr-x 2 root root 4096 abr 2 21:23 bin
4 drwxr-xr-x 3 root root 4096 abr 5 11:01 boot
```

Muestra el archivo list.txt en hexa y ASCII

```
jerome@linuxVm:~$ hexdump ./list.txt -C
00000000  74 6f 74 61 6c 20 31 30  34 0a 20 34 20 64 72 77 |total 104. 4 drw|
00000010  78 72 2d 78 72 2d 78 20  20 32 33 20 72 6f 6f 74 |xr-xr-x 23 root|
00000020  20 72 6f 6f 74 20 20 34  30 39 36 20 61 62 72 20 | root 4096 abr |
00000030  20 32 20 32 31 3a 32 32  20 2e 0a 20 34 20 64 72 | 2 21:22 .. 4 dr|
00000040  77 78 72 2d 78 72 2d 78  20 20 32 33 20 72 6f 6f |wxr-xr-x 23 roo|
00000050  74 20 72 6f 6f 74 20 20  34 30 39 36 20 61 62 72 |t root 4096 abr|
00000060  20 20 32 20 32 31 3a 32  32 20 2e 2e 0a 20 34 20 | 2 21:22 ... 4 |
00000070  64 72 77 78 72 2d 78 72  2d 78 20 20 20 32 20 72 |drwxr-xr-x 2 r|
00000080  6f 6f 74 20 72 6f 6f 74  20 20 34 30 39 36 20 61 |oot root 4096 a|
00000090  62 72 20 20 32 20 32 31  3a 32 33 20 62 69 6e 0a |br 2 21:23 bin.|
000000a0  20 34 20 64 72 77 78 72  2d 78 72 2d 78 20 20 20 | 4 drwxr-xr-x |
000000b0  33 20 72 6f 6f 74 20 72  6f 6f 74 20 20 34 30 39 |3 root root 409|
000000c0  36 20 61 62 72 20 20 35  20 31 31 3a 30 31 20 62 |6 abr 5 11:01 b|
000000d0  6f 6f 74 0a 20 34 20 64  72 77 78 72 77 78 72 2d |oot. 4 drwxrwxr-|
000000e0  78 20 20 20 32 20 72 6f  6f 74 20 72 6f 6f 74 20 |x 2 root root |
000000f0  20 34 30 39 36 20 61 62  72 20 20 32 20 32 31 3a | 4096 abr 2 21:|
00000100  31 35 20 63 64 72 6f 6d  0a 20 30 20 64 72 77 78 |15 cdrom. 0 drwx|
00000110  72 2d 78 72 2d 78 20 20  31 38 20 72 6f 6f 74 20 |r-xr-x 18 root |
00000120  72 6f 6f 74 20 20 33 39  32 30 20 6d 61 79 20 31 |root 3920 may 1|
00000130  30 20 31 35 3a 33 33 20  64 65 76 0a 31 32 20 64 |0 15:33 dev.12 d|
00000140  72 77 78 72 2d 78 72 2d  78 20 31 32 32 20 72 6f |rwxr-xr-x 122 ro|
00000150  6f 74 20 72 6f 6f 74 20  31 32 32 38 38 20 6d 61 |ot root 12288 ma|
00000160  79 20 31 30 20 31 35 3a  33 33 20 65 74 63 0a 20 |y 10 15:33 etc. |
00000170  34 20 64 72 77 78 72 2d  78 72 2d 78 20 20 20 33 |4 drwxr-xr-x 3|
00000180  20 72 6f 6f 74 20 72 6f  6f 74 20 20 34 30 39 36 | root root 4096|
00000190  20 61 62 72 20 20 32 20  32 31 3a 32 31 20 68 6f | abr 2 21:21 ho|
000001a0  6d 65 0a 20 30 20 6c 72  77 78 72 77 78 72 77 78 |me. 0 lrwxrwxrwx|
000001b0  20 20 20 31 20 72 6f 6f  74 20 72 6f 6f 74 20 20 | 1 root root |
000001c0  20 20 33 33 20 61 62 72  20 20 32 20 32 31 3a 32 | 33 abr 2 21:2|
000001d0  32 20 69 6e 69 74 72 64  2e 69 6d 67 20 2d 3e 20 |2 initrd.img -> |
000001e0  62 6f 6f 74 2f 69 6e 69  74 72 64 2e 69 6d 67 2d |boot/initrd.img-|
000001f0  34 2e 31 30 2e 30 2d 32  38 2d 67 65 6e 65 72 69 |4.10.0-28-generi|
```

00000200	63	0a	20	34	20	64	72	77	78	72	2d	78	72	2d	78	20	c. 4 drwxr-xr-x
00000210	20	32	30	20	72	6f	6f	74	20	72	6f	6f	74	20	20	34	20 root root 4
00000220	30	39	36	20	61	62	72	20	20	32	20	32	31	3a	32	33	096 abr 2 21:23
00000230	20	6c	69	62	0a	20	34	20	64	72	77	78	72	2d	78	72	lib. 4 drwxr-xr
00000240	2d	78	20	20	20	32	20	72	6f	6f	74	20	72	6f	6f	74	-x 2 root root
00000250	20	20	34	30	39	36	20	61	62	72	20	20	33	20	30	34	4096 abr 3 04
00000260	3a	30	37	20	6c	69	62	36	34	0a	31	36	20	64	72	77	:07 lib64.16 drw
00000270	78	2d	2d	2d	2d	2d	2d	20	20	20	32	20	72	6f	6f	74	x----- 2 root
00000280	20	72	6f	6f	74	20	31	36	33	38	34	20	61	62	72	20	root 16384 abr
00000290	20	32	20	32	31	3a	30	37	20	6c	6f	73	74	2b	66	6f	2 21:07 lost+fo
000002a0	75	6e	64	0a	20	34	20	64	72	77	78	72	2d	78	72	2d	und. 4 drwxr-xr-
000002b0	78	20	20	20	33	20	72	6f	6f	74	20	72	6f	6f	74	20	x 3 root root
000002c0	20	34	30	39	36	20	61	62	72	20	20	32	20	32	32	3a	4096 abr 2 22:
000002d0	32	32	20	6d	65	64	69	61	0a	20	34	20	64	72	77	78	22 media. 4 drwx
000002e0	72	2d	78	72	2d	78	20	20	20	32	20	72	6f	6f	74	20	r-xr-x 2 root
000002f0	72	6f	6f	74	20	20	34	30	39	36	20	61	67	6f	20	20	root 4096 ago
00000300	31	20	20	32	30	31	37	20	6d	6e	74	0a	20	34	20	64	1 2017 mnt. 4 d
00000310	72	77	78	72	2d	78	72	2d	78	20	20	20	33	20	72	6f	rwxr-xr-x 3 ro
00000320	6f	74	20	72	6f	6f	74	20	20	34	30	39	36	20	61	62	ot root 4096 ab
00000330	72	20	20	35	20	31	30	3a	35	39	20	6f	70	74	0a	20	r 5 10:59 opt.
00000340	30	20	64	72	2d	78	72	2d	78	72	2d	78	20	31	33	37	0 dr-xr-xr-x 137
00000350	20	72	6f	6f	74	20	72	6f	6f	74	20	20	20	20	20	30	root root 0
00000360	20	6d	61	79	20	31	30	20	31	35	3a	33	32	20	70	72	may 10 15:32 pr
00000370	6f	63	0a	20	34	20	64	72	77	78	2d	2d	2d	2d	2d	2d	oc. 4 drwx-----
00000380	20	20	20	33	20	72	6f	6f	74	20	72	6f	6f	74	20	20	3 root root
00000390	34	30	39	36	20	6d	61	79	20	31	30	20	31	35	3a	33	4096 may 10 15:3
000003a0	33	20	72	6f	6f	74	0a	20	30	20	64	72	77	78	72	2d	3 root. 0 drwxr-
000003b0	78	72	2d	78	20	20	32	32	20	72	6f	6f	74	20	72	6f	xr-x 22 root ro
000003c0	6f	74	20	20	20	36	38	30	20	6d	61	79	20	31	30	20	ot 680 may 10
000003d0	31	35	3a	33	38	20	72	75	6e	0a	31	32	20	64	72	77	15:38 run.12 drw
000003e0	78	72	2d	78	72	2d	78	20	20	20	32	20	72	6f	6f	74	xr-xr-x 2 root
000003f0	20	72	6f	6f	74	20	31	32	32	38	38	20	61	62	72	20	root 12288 abr
00000400	20	35	20	31	31	3a	30	31	20	73	62	69	6e	0a	20	34	5 11:01/sbin. 4
00000410	20	64	72	77	78	72	2d	78	72	2d	78	20	20	20	32	20	drwxr-xr-x 2
00000420	72	6f	6f	74	20	72	6f	6f	74	20	20	34	30	39	36	20	root root 4096
00000430	61	67	6f	20	20	31	20	20	32	30	31	37	20	73	72	76	ago 1 2017 srv
00000440	0a	20	30	20	64	72	2d	78	72	2d	78	72	2d	78	20	20	. 0 dr-xr-xr-x
00000450	31	33	20	72	6f	6f	74	20	72	6f	6f	74	20	20	20	20	13 root root
00000460	20	30	20	6d	61	79	20	31	30	20	31	35	3a	33	38	20	0 may 10 15:38
00000470	73	79	73	0a	20	34	20	64	72	77	78	72	77	78	72	77	sys. 4 drwxrwxrw
00000480	74	20	20	20	39	20	72	6f	6f	74	20	72	6f	6f	74	20	t 9 root root
00000490	20	34	30	39	36	20	6d	61	79	20	31	30	20	31	35	3a	4096 may 10 15:
000004a0	33	38	20	74	6d	70	0a	20	34	20	64	72	77	78	72	2d	38 tmp. 4 drwxr-
000004b0	78	72	2d	78	20	20	31	30	20	72	6f	6f	74	20	72	6f	xr-x 10 root ro
000004c0	6f	74	20	20	34	30	39	36	20	61	67	6f	20	20	31	20	ot 4096 ago 1
000004d0	20	32	30	31	37	20	75	73	72	0a	20	34	20	64	72	77	2017 usr. 4 drw
000004e0	78	72	2d	78	72	2d	78	20	20	31	33	20	72	6f	6f	74	xr-xr-x 13 root
000004f0	20	72	6f	6f	74	20	20	34	30	39	36	20	61	67	6f	20	root 4096 ago
00000500	20	31	20	20	32	30	31	37	20	76	61	72	0a	20	30	20	1 2017 var. 0
00000510	6c	72	77	78	72	77	78	72	77	78	20	20	20	31	20	72	lrwxrwxrwx 1 r
00000520	6f	6f	74	20	72	6f	6f	74	20	20	20	20	33	30	20	61	oot root 30 a
00000530	62	72	20	20	32	20	32	31	3a	32	32	20	76	6d	6c	69	br 2 21:22 vml
00000540	6e	75	7a	20	2d	3e	20	62	6f	6f	74	2f	76	6d	6c	69	nuz -> boot/vml

```
00000550  6e 75 7a 2d 34 2e 31 30  2e 30 2d 32 38 2d 67 65  |nuz-4.10.0-28-ge|
00000560  6e 65 72 69 63 0a                                |neric.|
00000566
```

## Ejemplos

1. Programa que crea un archivo con el nombre prueba.txt, escribe en él la leyenda "Hola mundo!!!" y cierra el archivo.

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

int main (void)
{
    FILE *f;
    int aux;

    //-- Abro el archivo --
    f = fopen (".prueba.txt", "w");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Escribo --
    aux = fprintf (f, "Hola mundo!!!");
    if (aux < 0) {
        printf ("Error al escribir en el archivo\r\n");
        return (-1);
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo20\_01.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo20_01.c -Wall -oejemplo20_01.out
jerome@linuxVm:~$ ./ejemplo20_01.out
jerome@linuxVm:~$ cat ./prueba.txt
jerome@linuxVm:~$ hexdump ./prueba.txt -C
```

Finalmente abra el archivo prueba.txt con un editor de texto o con el comando cat y confirme su contenido. También verifique el contenido con okteta o el comando hexdump

## 2. Programa lee un archivo que contiene una línea de texto y la imprime en pantalla.

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

int main (void)
{
    FILE *f;
    int aux;
    char buffer[32], *p;

    //-- Abro el archivo --
    f = fopen("./prueba.txt", "r");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Leo --
    p = fgets (buffer, sizeof (buffer), f);
    if (p == NULL) {
        printf ("Error al leer el archivo\r\n");
        return (-2);
    }

    printf ("El string leído es: %s\r\n", buffer);

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
        return (-3);
    }

    return (0);
}
```

### 3. Programa que agrega una línea de texto al final del archivo.

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

int main (void)
{
    FILE *f;
    int aux;

    //-- Abro el archivo --
    f = fopen (".prueba.txt", "a");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Escribo --
    aux = fprintf (f, "Nueva linea!!\r\n");
    if (aux < 0) {
        printf ("Error al escribir en el archivo\r\n");
        return (-1);
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo20\_03.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo20_03.c -Wall -oejemplo20_03.out
jerome@linuxVm:~$ ./ejemplo20_03.out
jerome@linuxVm:~$ cat ./prueba.txt
jerome@linuxVm:~$ hexdump ./prueba.txt -C
```

Finalmente abra el archivo prueba.txt con un editor de texto o con el comando cat y confirme su contenido. También verifique el contenido con okteta o el comando hexdump

4. Programa que escribe un número entero ingresado por el usuario en un archivo llamado pruebaNum.txt

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

int main (void)
{
    FILE *f;
    int aux;
    int a;

    //-- Ingreso de un numero entero --
    printf ("Ingrese numero\r\n");
    scanf ("%d", &a);

    //-- Abro el archivo --
    f = fopen (".pruebaNum.txt", "w");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Escribo --
    aux = fprintf (f, "%d\r\n", a);
    if (aux < 0) {
        printf ("Error al escribir en el archivo\r\n");
        return (-1);
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo20\_04.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo20_04.c -Wall -oejemplo20_04.out
jerome@linuxVm:~$ ./ejemplo20_04.out
jerome@linuxVm:~$ cat ./pruebaNum.txt
jerome@linuxVm:~$ hexdump ./pruebaNum.txt -C
```

Finalmente abra el archivo pruebaNum.txt con un editor de texto o con el comando cat y confirme su contenido. También verifique el contenido con okteta o el comando hexdump

5. Programa que lee un número entero que se encuentra almacenado en un archivo.

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

int main (void)
{
    FILE *f;
    int aux;
    int a;

    //-- Abro el archivo --
    f = fopen (".pruebaNum.txt", "r");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Leo un numero del archivo --
    aux = fscanf (f, "%d\r\n", &a);
    if (aux <= 0) {
        printf ("Error al leer el archivo\r\n");
        return (-1);
    } else {
        printf ("El numero leido es: %d\r\n", a);
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo20\_05.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo20_05.c -Wall -oejemplo20_05.out
jerome@linuxVm:~$ ./ejemplo20_05.out
```

Finalmente verifique que el número que se leyó del archivo coincida con el almacenado por el ejemplo anterior.

## 6. Programa que escribe diez números enteros en un archivo llamado pruebaNum.txt

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

int main (void)
{
    FILE *f;
    int aux;
    int i;

    //-- Abro el archivo --
    f = fopen (".pruebaNum.txt", "w");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Escribo --
    for (i = 0; i < CANT; i++) {
        aux = fprintf (f, "%d\r\n", i);
        if (aux < 0) {
            printf ("Error al escribir en el archivo\r\n");
            return (-1);
        }
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo20\_06.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo20_06.c -Wall -oejemplo20_06.out
jerome@linuxVm:~$ ./ejemplo20_06.out
jerome@linuxVm:~$ cat ./pruebaNum.txt
jerome@linuxVm:~$ hexdump ./pruebaNum.txt -C
```

Finalmente abra el archivo pruebaNum.txt con un editor de texto o con el comando cat y confirme su contenido. También verifique el contenido con okteta o el comando hexdump



7. Programa que lee todos los números enteros en un archivo llamado pruebaNum.txt y los imprime en pantalla.

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

int main (void)
{
    FILE *f;
    int aux;
    int a;

    //-- Abro el archivo --
    f = fopen (".pruebaNum.txt", "r");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Leo --
    while (!feof(f)) {
        aux = fscanf (f, "%d\r\n", &a);
        if (aux < 0) {
            printf ("Error al leer en el archivo\r\n");
            return (-1);
        } else {
            printf ("El numero leido es: %d\r\n", a);
        }
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo20\_07.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo20_07.c -Wall -oejemplo20_07.out
jerome@linuxVm:~$ ./ejemplo20_07.out
```

## Ejercicios

1. Implemente un programa que imprima todo el contenido del archivo ejercicio20\_01.c por pantalla.
2. Implemente un programa que le pida al usuario que ingrese la ruta completa (path) y el nombre de un archivo (se ingresa todo junto) y luego imprima todo el contenido del archivo. Tenga en cuenta que si usa la función `fgets` la misma agrega el carácter `\x0A` al final de lo ingresado y deberá ser eliminado, para ello puede usar la función `strchr` que busca un caracter en un string y reemplázelo por `'\0'`
3. Desarrolle un programa que cree un archivo llamado prueba.txt y utilizando la función `fprintf` (verifique el valor devuelto por ella) escriba todos los números impares entre cero y 100. Si el archivo existiera lo deberá sobrescribir.
4. Implemente un programa que le pida al usuario diez nombres junto con diez edades y los almacene en un archivo con el siguiente formato. Use la función `fprintf`.

```
Nombre,edad\r\n
Jose,33\r\n
```

Observe el contenido del archivo con un visualizador hexadecimal y con un editor de texto. Obtenga conclusiones. El nombre del archivo es "bDatos.txt" colóquelo en un `define`.

5. Desarrolle un programa que lea el contenido de un archivo llamado "bDatos.txt" que contiene datos con el formato descrito en el punto anterior y los coloque en un vector de las siguientes estructuras. (use `fscanf(f, "%[^,]%d\r\n", nombre, &edad)`; donde nombre es un string y edad es un int.)

```
#define NOMBRE_SIZE      (32)
struct data_S {
    char nombre[NOMBRE_SIZE];
    unsigned int edad;
};
```

La cantidad máxima de registros es diez. Finalmente imprima el vector de estructuras en pantalla.



## 21. Archivos II

### Funciones utilizadas de stdio.h

Función	Descripción
ftell	Devuelve la posición actual dentro del archivo.
fseek	Se mueve a la posición indicada del archivo.
rewind	Vuelve al inicio del archivo.
fstat	Retorna información acerca del archivo.
fwrite	Escribe en el archivo
fread	Lee un archivo

### Funciones utilizadas de sys/stat.h

Función	Descripción
fstat	Retorna información acerca del archivo. Necesita los siguientes includes: #include <sys/types.h> #include <sys/stat.h> #include <unistd.h>

### Comandos

Comando	Descripción
chmod	Cambia permisos de a un archivo o directorio.
chown	Cambia al propietario a un archivo o directorio.
chgrp	Cambia al propietario a un archivo o directorio

### Ejemplo de uso de comandos.

Creamos un archivo de texto.

```
jerome@linuxVm:~$ ls / -las > list.txt
```

Listamos al archivo usando ls -l

```
jerome@linuxVm:~$ ls ./list.txt -l
-rw-rw-r-- 1 jerome jerome 1382 may 10 15:39 ./list.txt
```

Cada columna de lo devuelto por ls -l es:

Permisos	Hard links	Propietario	Grupo propietario	Tamaño	Fecha	Nombre
-rw-rw-r--	1	jerome	jerome	1382	may 10 15:39	list.txt

- Permisos: Este campo se divide en cuatro

Tipo de archivo	Permisos propietario	Permisos del grupo propietario	Permisos del resto de usuarios del sistema
-	rw-	rw-	r--

- Tipo de archivo:
  - - (guión): Archivo
  - d: Directorio
  - l: Link
- Permisos:
  - r : Read (Lectura)
  - w: Write (Escritura)
  - x: Execute (Ejecución)

Número	Representación	Permiso
0	---	Sin permisos
1	--x	Ejecución
2	-w-	Escritura
3	-wx	Escritura y ejecución
4	r--	Lectura
5	r-x	Lectura y ejecución
6	rw-	Lectura y escritura
7	rwX	Lectura, escritura y ejecución.

- Hard link: Cantidad de hard links
- Propietario: Es el propietario del archivo.
- Grupo propietario: Es el grupo al que pertenece el archivo.
- Tamaño: Tamaño en bytes del archivo.
- Fecha: Fecha de la última modificación del archivo.
- Nombre: Nombre del archivo

Ejemplo de uso de uso de chmod, le damos al archivo permisos de lectura, escritura y ejecución para el propietario, el grupo y otros.

```
jerome@linuxVm:~$ chmod 777 list.txt
jerome@linuxVm:~$ ls ./list.txt -l
-rwxrwxrwx 1 jerome jerome 1382 may 10 15:39 ./list.txt
```

## Ejemplos

1. Programa que le pide al usuario el nombre de un archivo e imprima en pantalla el tamaño en bytes del mismo.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int main (void)
{
    int aux;
    struct stat st;

    //-- leo la estructura stat del archivo --
    aux = stat("./ejemplo01.c", &st);
    if (aux < 0) {
        printf ("Error en fstat\r\n");
        return (-1);
    }

    printf ("El archivo tiene %ld bytes\r\n", st.st_size);

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo20\_01.c

Compilando y ejecutando el programa. Verifique el tamaño del archivo con el comando ls -l

```
jerome@linuxVm:~$ gcc ejemplo20_01.c -Wall -oejemplo20_01.out
jerome@linuxVm:~$ ./ejemplo20_01.out
jerome@linuxVm:~$ ls -l
```

2. Programa que escribe en un archivo llamado prueba.bin un número entero ingresado por teclado

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

int main (void)
{
    FILE *f;
    int aux, a;

    //-- Abro el archivo --
    f = fopen ("./prueba.bin", "w");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Ingreso un entero --
    printf ("Ingrese un numero\r\n");
```

```

scanf ("%d", &a);

//-- Escribo --
aux = fwrite (&a, sizeof (a), 1, f);
if (aux < 0) {
    printf ("Error al escribir en el archivo\r\n");
    return (-1);
}

//-- Cierro --
aux = fclose (f);
if (aux != 0) {
    printf ("Error al cerrar el archivo\r\n");
}

return (0);
}

```

Escriba y guarde el código anterior en un archivo llamado ejemplo21\_02.c

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo21_02.c -Wall -oejemplo21_02.out
jerome@linuxVm:~$ ./ejemplo21_02.out
jerome@linuxVm:~$ hexdump ./prueba.bin -C

```

Finalmente abra el archivo prueba.txt con un editor de texto o con el comando cat y confirme su contenido. También verifique el contenido con okteta o el comando hexdump

### 3. Programa que escribe en un archivo llamado prueba.bin los números enteros del 0 al 9

```

#include <stdio.h>
#include <stdlib.h>
#define CANT ((int)10)

int main (void)
{
    FILE *f;
    int aux, i;
    int v[CANT];

    //-- Abro el archivo --
    f = fopen ("./prueba.bin", "w");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Creo un vector con los numeros del 0 al 9 --
    for (i = 0 ;i < CANT; i++) {
        v[i] = i;
    }

    //-- Escribo --
    aux = fwrite (&v[0], sizeof (v[0]), CANT, f);
}

```

```

    if (aux < 0) {
        printf ("Error al escribir en el archivo\r\n");
        return (-1);
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}

```

Escriba y guarde el código anterior en un archivo llamado ejemplo21\_03.c

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo21_03.c -Wall -oejemplo21_03.out
jerome@linuxVm:~$ ./ejemplo21_03.out
jerome@linuxVm:~$ hexdump ./prueba.bin -C

```

Finalmente abra el archivo prueba.bin con un editor hexa o el comando hexdump y confirme su contenido.

4. Programa que lee un archivo llamado prueba.bin e imprime el contenido del mismo en pantalla.

```

#include <stdio.h>
#include <stdlib.h>

#define CANT ((int)20)

int main (void)
{
    FILE *f;
    int aux, i;
    int v[CANT], cantRead;

    //-- Abro el archivo --
    f = fopen (".prueba.bin", "r");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Leo --
    cantRead = fread (&v[0], sizeof (v[0]), CANT, f);
    if (cantRead < 0) {
        printf ("Error al escribir en el archivo\r\n");
        return (-1);
    } else {
        printf ("Cantidad de enteros leidos: %d\r\n", cantRead);
        for (i = 0; i < cantRead; i++) {
            printf ("%d\r\n", v[i]);
        }
    }
}

```

```

    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}

```

5. Programa que escribe en un archivo llamado prueba.bin un vector de estructuras de tipo struct persona

```

#include <stdio.h>
#include <stdlib.h>

#define CANT ((int)3)

struct persona {
    char nombre [16];
    int edad;
};

int main (void)
{
    FILE *f;
    int aux, i, cantWrite;
    struct persona v[CANT];

    //-- Abro el archivo --
    f = fopen (".prueba.bin", "w");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Ingreso de datos --
    for (i = 0; i < CANT; i++) {
        printf ("Ingrese nombre\r\n");
        scanf ("%s", v[i].nombre);
        printf ("Ingrese edad\r\n");
        scanf ("%d", &v[i].edad);
    }

    //-- Escribo --
    cantWrite = fwrite (&v[0], sizeof (v[0]), CANT, f);
    if (cantWrite < 0) {
        printf ("Error al escribir en el archivo\r\n");
        return (-1);
    } else {

```



```

        printf ("Cantidad de estructuras escritas: %d\r\n", cantWrite);
    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }
    return (0);
}

```

Escriba y guarde el código anterior en un archivo llamado ejemplo21\_05.c

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo21_05.c -Wall -oejemplo21_05.out
jerome@linuxVm:~$ ./ejemplo21_05.out
jerome@linuxVm:~$ cat ./prueba.bin
jerome@linuxVm:~$ hexdump ./prueba.bin -C

```

Finalmente abra el archivo prueba.bin con un editor hexa o el comando hexdump y confirme su contenido.

6. Programa que lee de un archivo llamado prueba.bin el vector de estructuras struct persona almacenado.

```

#include <stdio.h>
#include <stdlib.h>

#define CANT ((int)3)

struct persona {
    char nombre [16];
    int edad;
};

int main (void)
{
    FILE *f;
    int aux, i, cantRead;
    struct persona v[CANT];

    //-- Abro el archivo --
    f = fopen ("./prueba.bin", "r");
    if (f == NULL) {
        printf ("Error al abrir el archivo\r\n");
        return (-1);
    }

    //-- Leo el archivo --
    cantRead = fread (&v[0], sizeof (v[0]), CANT, f);
    if (cantRead < 0) {
        printf ("Error al escribir en el archivo\r\n");
        return (-1);
    } else {
        printf ("Cantidad de estructuras leidas: %d\r\n", cantRead);
    }
}

```

```

        for (i = 0; i < CANT; i++) {
            printf ("%d.%s,%d\r\n", i, v[i].nombre, v[i].edad);
        }

    }

    //-- Cierro --
    aux = fclose (f);
    if (aux != 0) {
        printf ("Error al cerrar el archivo\r\n");
    }

    return (0);
}

```


## Ejercicios

1. Implemente una función que imprima en hexadecimal por pantalla los datos apuntados por un puntero. El prototipo de la función es el siguiente  

```
void printHex (void *dataPtr, int dataSize)
```

Donde:

  - o dataPtr: Puntero a los datos a imprimir en hexadecimal.
  - o dataSize: Cantidad de datos a escribir.
2. Utilizando un editor de texto, cree un archivo con el nombre prueba.txt. Escriba en él la leyenda: Hola mundo!!! guarde y cierre el archivo. Implemente un programa que lea el contenido del archivo prueba.txt usando la función fread y que imprima el contenido por pantalla utilizando printf (Tenga en cuenta que el archivo no está almacenado el '\0') además imprímalo usando la función printHex (No copie la función en su código, use el archivo .c generado anteriormente) . Intente leer el archivo byte a byte o de a 32 bytes. Verifique el valor devuelto por las funciones fread y feof en cada caso.
3. Desarrolle un programa que cree un archivo llamado prueba.txt y utilizando la función fwrite escriba la leyenda Hola info1!!! Si el archivo existiera lo deberá sobrescribir. Tenga en cuenta que el carácter '\0' no debe ser escrito en el archivo. Verifique el contenido del archivo utilizando un editor hexa y un editor de texto. Obtenga conclusiones.
4. Implemente un programa que le agregue la leyenda Adiós!! al archivo prueba.txt creado en el punto anterior, sin destruir el contenido previo del archivo.
5. Desarrolle un programa que le pida al usuario que ingrese 10 números enteros y los vaya almacenando en un archivo con el nombre entero.bin utilizando la función fwrite. Verifique el contenido del archivo utilizando un editor hexa y un editor de texto. Obtenga conclusiones.
6. Desarrolle un programa que lea utilizando la función fread el archivo creado en el punto anterior y coloque los datos en un vector de enteros generado dinámicamente. Finalmente imprima en decimal y hexadecimal por pantalla el vector de enteros leído. Compare lo impreso en pantalla con lo observado en el archivo utilizando un editor hexadecimal.

- 
7. Desarrolle un programa que realice la copia de un archivo indicado por el usuario a través de la línea de comandos Ejemplo copiar origen destino

Donde:

- copiar: es el nombre de nuestro programa.
- origen: ruta y nombre del archivo origen.
- destino: ruta y nombre del archivo destino.

Observe y compare el contenido usando el okteta.

8. Desarrolle un programa que imprima por pantalla en hexadecimal la segunda mitad de un archivo indicado por el usuario. Para ello deberá usar la función `fseek` para posicionarse en la mitad del archivo y luego con la función `fread` leerlo.
9. Implemente un programa que le pida al usuario que ingrese nombres junto con edades y los almacene en un archivo (Use la estructura que se encuentra debajo para almacenar los datos). El final del ingreso de datos ocurre cuando el usuario ingresa como edad -1.

```
#define NOMBRE_SIZE      (32)
struct data_S {
    char nombre[NOMBRE_SIZE];
    int edad;
};
```

El nombre del archivo es "bDatos.bin" colóquelo en un define.

10. Desarrolle un programa que imprima por pantalla el contenido del archivo creado en el punto anterior. Tenga en cuenta que desconoce la cantidad de estructuras almacenadas.



## 22. Punteros, el regreso!

En este capítulo se verán ejemplos de los siguientes usos de punteros

- Array de punteros.
- Array de punteros a string.
- Argumentos de main.
- Puntero a función.
- Array de punteros a función

Ejemplos de uso de punteros

	Variable int	Vector de int
<b>Declaración de variable</b>	<code>int a;</code>	<code>int v[3];</code>
<b>Declaración de puntero</b>	<code>int *p;</code>	<code>int *p;</code>
<b>Inicialización de puntero</b>	<code>p = &amp;a;</code>	<code>p = &amp;v[0];</code>
<b>Asigno un valor a la variable a</b>	<code>a = 10;</code>	<code>v[0] = 1;</code> <code>v[1] = 2;</code> <code>v[2] = 3;</code>
<b>Asigno un valor a la variable usando un puntero</b>	<code>*p = 10; // a = 10;</code>	<code>*(p + 0) = 1; // v[0] =1;</code> <code>*(p + 1) = 2; // v[1] =2;</code> <code>*(p + 2) = 3; // v[2] =3;</code>
<b>Imprimo el valor de la variable</b>	<code>printf ("%d\r\n", a);</code>	<code>printf ("%d\r\n", v[0]);</code> <code>printf ("%d\r\n", v[1]);</code> <code>printf ("%d\r\n", v[2]);</code>
<b>Imprimo el valor de la variable usando el puntero</b>	<code>printf ("%d\r\n", *p);</code>	<code>printf ("%d\r\n", *(p + 0));</code> <code>printf ("%d\r\n", *(p + 1));</code> <code>printf ("%d\r\n", *(p + 2));</code>

## Ejemplos

1. Programa que crea un vector de 3 punteros a enteros, los apunta a tres enteros para luego operar sobre las variables usando los punteros.

```
#include <stdio.h>

int main (void)
{
    int *p[3];
    int a = 11, b = 22, c = 33;

    p[0] = &a;
    p[1] = &b;
    p[2] = &c;

    //-- Accedo a las variables a,b y c utilizando el vector de punteros --
    printf ("a = %d\tb = %d\tc = %d\r\n", a, b, c);
    printf ("*p[0] = %d\t*p[1] = %d\t*p[2] = %d\r\n", *p[0], *p[1], *p[2]);

    //-- Muestro la direccion de las variables a, b y c y los punteros --
    printf ("%a = %p\t&b = %p\t&c = %p\r\n", &a, &b, &c);
    printf ("p[0] = %p\tp[1] = %p\tp[2] = %p\r\n", p[0], p[1], p[2]);

    //-- Modifico el valor de las variables a, b, c usando los punteros --
    *p[0] = 100; *p[1] = 200; *p[2] = 300;
    printf ("a = %d\tb = %d\tc = %d\r\n", a, b, c);
    printf ("*p[0] = %d\t*p[1] = %d\t*p[2] = %d\r\n", *p[0], *p[1], *p[2]);

    return (0);
}
```

```
jerome@linuxVm:~$ gcc ejemplo22_01.c -Wall -oejemplo22_01.out
jerome@linuxVm:~$ ./ejemplo22_01.out
    a = 11          b = 22          c = 33
*p[0] = 11         *p[1] = 22         *p[2] = 33

    &a = 0x7ffeb47cd5a4    &b = 0x7ffeb47cd5a8    &c = 0x7ffeb47cd5ac
p[0] = 0x7ffeb47cd5a4    p[1] = 0x7ffeb47cd5a8    p[2] = 0x7ffeb47cd5ac

    a = 100          b = 200          c = 300
*p[0] = 100         *p[1] = 200         *p[2] = 300
```

## 2. Programa que crea un vector de punteros y se lo pasa a una función como parámetro.

```
#include <stdio.h>

void func (int **p)
{
    *p[0] = 100; *p[1] = 200; *p[2] = 300;
}

int main (void)
{
    int *p[3];
    int a = 11, b = 22, c = 33;

    p[0] = &a;
    p[1] = &b;
    p[2] = &c;

    printf ("Valores de las variables\r\n");
    printf ("a = %d\tb = %d\tc = %d\r\n", a, b, c);
    printf ("*p[0] = %d\t*p[1] = %d\t*p[2] = %d\r\n", *p[0], *p[1], *p[2]);

    printf ("Direcciones\r\n");
    printf ("%a = %p\t&b = %p\t&c = %p\r\n", &a, &b, &c);
    printf ("p[0] = %p\tp[1] = %p\tp[2] = %p\r\n", p[0], p[1], p[2]);

    printf ("Modifico los valores en la funcion\r\n");
    func (&p[0]);
    printf ("a = %d\tb = %d\tc = %d\r\n", a, b, c);
    printf ("*p[0] = %d\t*p[1] = %d\t*p[2] = %d\r\n", *p[0], *p[1], *p[2]);

    return (0);
}
```

```
jerome@linuxVm:~$ gcc ejemplo22_02.c -Wall -oejemplo22_02.out
jerome@linuxVm:~$ ./ejemplo22_02.out
Valores de las variables
    a = 11          b = 22          c = 33
*p[0] = 11         *p[1] = 22         *p[2] = 33

Direcciones
    &a = 0x7ffffac887434      &b = 0x7ffffac887438      &c = 0x7ffffac88743c
p[0] = 0x7ffffac887434      p[1] = 0x7ffffac887438      p[2] = 0x7ffffac88743c

Modifico los valores en la funcion
    a = 100          b = 200          c = 300
*p[0] = 100         *p[1] = 200         *p[2] = 300
```

### 3. Programa que genera dinámicamente un array de punteros a entero

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

int main (void)
{
    int **p;
    int a = 11, b = 22, c = 33;

    p = (int **)malloc (sizeof (*p) * 3);
    if (p == NULL) {
        printf ("Error malloc \r\n");
        return (-1);
    }

    *(p + 0) = &a;
    *(p + 1) = &b;
    *(p + 2) = &c;

    printf ("Valores de las variables\r\n");
    printf ("a = %d\tb = %d\tc = %d\r\n", a, b, c);
    printf ("*p[0] = %d\t*p[1] = %d\t*p[2] = %d\r\n", *p[0], *p[1], *p[2]);

    printf ("Direcciones\r\n");
    printf ("&a = %p\t&b = %p\t&c = %p\r\n", &a, &b, &c);
    printf ("*p[0] = %p\t*p[1] = %p\t*p[2] = %p\r\n", p[0], p[1], p[2]);

    return (0);
}
```

```
jerome@linuxVm:~$ gcc ejemplo22_03.c -Wall -oejemplo22_03.out
jerome@linuxVm:~$ ./ejemplo22_03.out
Valores de las variables
    a = 11          b = 22          c = 33
*p[0] = 11         *p[1] = 22         *p[2] = 33

Direcciones
    &a = 0x7fffac887434    &b = 0x7fffac887438    &c = 0x7fffac88743c
p[0] = 0x7fffac887434    p[1] = 0x7fffac887438    p[2] = 0x7fffac88743c
```

#### 4. Programa que genera dos vectores de punteros a char

- El primero lo inicializa con la dirección de 3 string en tiempo de compilación.
- El segundo lo inicializa con la dirección de 3 string en tiempo de ejecución.

```
#include <stdio.h>

int main (void)
{
    char *p[3];
    char *q[] = {
        "Hola",
        "Adios",
        "Info1"
    };

    char a[] = "Hola";
    char b[] = "Adios";
    char c[] = "Info1";

    p[0] = &a[0];
    p[1] = &b[0];
    p[2] = &c[0];

    printf ("Imprimo los strings\r\n");
    printf ("a = %s\tb = %s\tc = %s\r\n", a, b, c);
    printf ("p[0] = %s\tp[1] = %s\tp[2] = %s\r\n", p[0], p[1], p[2]);
    printf ("q[0] = %s\tq[1] = %s\tq[2] = %s\r\n", q[0], q[1], q[2]);

    printf ("Imprimo las direcciones de los strings\r\n");
    printf ("a = %p\tb = %p\tc = %p\r\n", &a[0], &b[0], &c[0]);
    printf ("p[0] = %p\tp[1] = %p\tp[2] = %p\r\n", p[0], p[1], p[2]);
    printf ("q[0] = %p\tq[1] = %p\tq[2] = %p\r\n", q[0], q[1], q[2]);

    return (0);
}
```

```
jerome@linuxVm:~$ gcc ejemplo22_04.c -Wall -oejemplo22_04.out
```

```
jerome@linuxVm:~$ ./ejemplo22_04.out
```

```
Imprimo los strings
```

```
    a = Hola        b = Adios        c = Info1
p[0] = Hola        p[1] = Adios        p[2] = Info1
q[0] = Hola        q[1] = Adios        q[2] = Info1
```

```
Imprimo las direcciones de los strings
```

```
    a = 0x7ffe54ac0290    b = 0x7ffe54ac02a0    c = 0x7ffe54ac02b0
p[0] = 0x7ffe54ac0290    p[1] = 0x7ffe54ac02a0    p[2] = 0x7ffe54ac02b0
q[0] = 0x4007b8          q[1] = 0x4007bd          q[2] = 0x4007c3
```



## 5. Programa que imprime todos los argumentos pasados por línea de comandos

```
#include <stdio.h>

int main (int argc, char *argv[])
{
    int i;

    for (i = 0; i < argc ; i++) {
        printf ("%d.%s\r\n", i, argv[i]);
    }
    return (0);
}
```

```
jerome@linuxVm:~$ gcc ejemplo22_05.c -Wall -oejemplo22_05.out
jerome@linuxVm:~$ ./ejemplo22_05.out Hola como te va
0../ejemplo22_05.out
1.hola
2.como
3.te
4.va
```

## 6. Programa demuestra el uso de un punteros a función

```
#include <stdio.h>

int suma (int a, int b)
{
    return(a + b);
}

int main (void)
{
    int a, b, r;
    int (*func) (int, int);

    //-- Asignacion del puntero a funcion --
    func = suma;

    //-- Ingreso de datos --
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &b);

    //-- Uso del puntero a funcion --
    r = func (a, b);
    printf ("El resultado es: %d\r\n", r);

    return (0);
}
```

## 7. Programa que muestra el uso de un array de punteros a función

```
#include <stdio.h>

#define SUMA ((int)0)
#define RESTA ((int)1)
#define DIVISION ((int)2)
#define MULTIPLICACION ((int)3)

int suma (int a, int b)
{
    return(a + b);
}

int resta (int a, int b)
{
    return(a - b);
}

int division (int a, int b)
{
    return(a / b);
}

int multiplicacion (int a, int b)
{
    return(a * b);
}

int main (void)
{
    int a, b, r;
    int (*func[4]) (int, int);
    int opIndex;

    //-- Asignacion del puntero a funcion --
    func[SUMA] = suma;
    func[RESTA] = resta;
    func[DIVISION] = division;
    func[MULTIPLICACION] = multiplicacion;

    //-- Ingreso de datos --
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &b);
    printf ("Ingrese operacion:\r\n");
    printf ("0. Suma\r\n");
    printf ("1. Resta\r\n");
    printf ("2. Division\r\n");
    printf ("3. Multiplicacion\r\n");
    scanf ("%d", &opIndex);
```

```

    if (opIndex < SUMA || opIndex > MULTIPLICACION ) {
        printf ("Operacion invalida\r\n");
        return (-1);
    }
    //-- Uso del puntero a funcion --
    r = func[opIndex] (a, b);
    printf ("El resultado es: %d\r\n", r);

    return (0);
}

```

#### 8. Programa que muestra el uso de un array de punteros a función.

```

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

#define SUMA ((int)0)
#define RESTA ((int)1)
#define DIVISION ((int)2)
#define MULTIPLICACION ((int)3)

int suma (int a, int b)
{
    return(a + b);
}

int resta (int a, int b)
{
    return(a - b);
}

int division (int a, int b)
{
    return(a / b);
}

int multiplicacion (int a, int b)
{
    return(a * b);
}

int main (void)
{
    int a, b, r;
    char op, *p;
    int (*func[4]) (int, int);
    char listOp[] = "+-/*";
    int opIndex;

    //-- Asignacion del puntero a funcion --
    func[SUMA] = suma;
    func[RESTA] = resta;

```

```

func[DIVISION] = division;
func[MULTIPLICACION] = multiplicacion;

//-- Ingreso de datos --
printf ("Ingrese un numero:\r\n");
scanf ("%d", &a);
printf ("Ingrese operacion\r\n");
scanf ("%*c%c", &op);
printf ("Ingrese un numero:\r\n");
scanf ("%d", &b);

p = strchr (listOp, op);
if (p == NULL) {
    printf ("Operacion invalida\r\n");
    return (0);
}

opIndex = (int) (p - listOp);
//-- Uso del puntero a funcion --
r = func[opIndex] (a, b);
printf ("El resultado es: %d\r\n", r);

return (0);
}

```

## Ejercicios

1. Implemente un programa que reciba por línea de comandos el nombre de un archivo e imprima en pantalla el tamaño en bytes del mismo.

```
jerome@linuxVm:~$ ./ejercicio22_01.out nombreArchivo
```

2. Implemente un programa que realice la copia de un archivo. El nombre del archivo origen y destino la reciba por línea de comandos. (Similar al comando cp)

```
jerome@linuxVm:~$ ./ejercicio22_02.out origen destino
```


3. Elabore un programa que por línea de comandos se pasan dos valores reales y el símbolo de la operación ('+', '-', '\*', '/'). Se deberá presentar por pantalla, los datos ingresados, la operación y el resultado. Si el símbolo utilizado no correspondiera a ninguna de las cuatro operaciones deberá presentar un mensaje de "Operación no válida".

```

jerome@linuxVm:~$ ./ejercicio22_03.out 2 + 3
2 + 3 = 5
jerome@linuxVm:~$ ./ejercicio22_03.out 2 * 3
2 * 3 = 6
jerome@linuxVm:~$ ./ejercicio22_03.out 2 ! 3
Operacion no valida

```

4. Modifique el programa anterior para usar un array de punteros a función.

- 
5. Implemente una función que lea de un archivo cuatro textos separados por "\r\n" y los coloque en cuatro string generados dinámicamente. Luego asigne estos cuatro punteros al vector de punteros pasado como parámetro en la función.

```
int leerTexto (char *nombre, char **textoPtr);
```

Donde:

nombre: Es el puntero al nombre del archivo

textoPtr: Es la cantidad de elementos del vector apuntado

Devuelve: La cantidad de líneas de texto leídas o un número negativo indicando el error.



## 23. Matrices.

### Ejemplos

1. Programa que crea de forma estática una matriz de enteros bidimensional de 3 x 3 y la rellena con los números del 0 al 9

```
#include <stdio.h>

#define FILAS ((int)3)
#define COLUMNAS ((int)3)

int main (void)
{
    int m[FILAS][COLUMNAS];
    int i, j;
    int c = 0;

    //-- Inicializo la matriz --
    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS ; j++) {
            m[i][j] = c;
            c++;
        }
    }

    //-- Imprimo la matriz --
    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS ; j++) {
            printf ("%d\t",m[i][j]);
        }
        printf ("\r\n");
    }

    return (0);
}
```

2. Programa que crea de forma estática una matriz de enteros bidimensional de 3 x 3 y la rellena con los números del 0 al 9 utilizando punteros para direccionar como matriz.

```
#include <stdio.h>

#define FILAS ((int)3)
#define COLUMNAS ((int)3)

int main (void)
{
    int m[FILAS][COLUMNAS];
    int i, j;
    int c = 0;
    int *p;

    p = &m[0][0];
    //-- Inicializo la matriz --
```

```

    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS ; j++) {
            *(p + (i * COLUMNAS) + j) = c;
            c++;
        }
    }

    //-- Imprimo la matriz --
    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS ; j++) {
            printf ("%d\t", *(p + (i * COLUMNAS) + j));
        }
        printf ("\r\n");
    }

    return (0);
}

```

3. Programa que crea de forma estática una matriz de enteros bidimensional de 3 x 3 y la rellena con los números del 0 al 9 utilizando punteros.

```

#include <stdio.h>

#define FILAS ((int)3)
#define COLUMNAS ((int)3)

int main (void)
{
    int m[FILAS][COLUMNAS];
    int i, j;
    int *p;

    p = &m[0][0];
    //-- Inicializo la matriz --
    for (i = 0; i < FILAS * COLUMNAS; i++) {
        *(p + i) = i;
    }

    //-- Imprimo la matriz --
    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS ; j++) {
            printf ("%d\t", *(p + (i * COLUMNAS) + j));
        }
        printf ("\r\n");
    }

    return (0);
}

```

4. Programa que crea de forma dinámica una matriz de enteros bidimensional de 3 x 3 y la rellena con los números del 0 al 9 utilizando punteros.

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

#define FILAS ((int)3)
#define COLUMNAS ((int)3)

int main (void)
{
    int *m;
    int i, j;
    int c = 0;

    m = (int *)malloc (FILAS * COLUMNAS * sizeof (*m));
    if (m == NULL) {
        printf ("Error malloc \r\n");
        return (-1);
    }

    //-- Inicializo la matriz --
    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS ; j++) {
            *(m + (i * COLUMNAS) + j) = c;
            c++;
        }
    }

    //-- Imprimo la matriz --
    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS ; j++) {
            printf ("%d\t", *(m + (i * COLUMNAS) + j));
        }
        printf ("\r\n");
    }
    free (m)
    return (0);
}
```

## Ejercicios

1. Implemente una función que imprima en pantalla una matriz de filas por columnas, el prototipo de la función es

```
void matrizImprimir(int *p, int filas, int columnas);
```

2. Implemente una función que cree una matriz de filas por columnas y la inicializa en cero, el prototipo de la función es:

```
int * matrizCrear (int filas, int columnas);
```

3. Implemente una función que almacena una matriz pasada como parámetro en un archivo, el prototipo de la función es el siguiente



```
int matrizEscribir (char *nombre, int *p, int filas, int columnas);
```

4. Implemente una función que lea el archivo generado por la función matriz escribir y vuelva a almacenar la matriz en memoria.

```
int matrizLeer (char *nombre, int *p,int filas, int columnas);
```

5. Implemente un programa que le permita a dos usuarios jugar al ta-te-ti. La selección del casillero a jugar se realiza usando el número de casillero.

0	1	2
3	4	5
6	7	8

Por ejemplo si el usuario quiere colocar una X en el centro deberá ingresar 4.

El programa deberá realizar lo siguiente::

- El programa debe indicar si el usuario puede jugar en la posición que indica o si esa posición ya fue jugada. Deberá seguir pidiéndole al usuario que juegue hasta que ingrese una jugada válida
  - El programa debe indicar si algún jugador gana la partida o es un empate.
6. Modifique el programa anterior para ir almacenando la última partida en un archivo. Se almacena cada cambio en la matriz de juego.

■ ■ ■

## 24. Operaciones a nivel de bits, campos de bits. Uniones. Enum

### Operadores a nivel de bits

Operador	Descripción
&	Operación and bit a bit
	Operación or bit a bit
~	Operación not bit a bit
^	Operación xor bit a bit
<<	Desplazamiento a la derecha
>>	Desplazamiento a la izquierda

### Ejemplos

Todos los ejemplos de los operadores a nivel de bit se aplican sobre una variable tipo char para que sea más sencillo de entender, pero pueden aplicarse sobre cualquier tipo de datos. Por otra parte se define una función llamada `chartoBin` que imprime en binario el valor char pasado como parámetro.

#### 1. Programa que demuestra el uso de la función `toBin`

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

void chartoBin (unsigned char data, char *binPtr)
{
    int i;
    int sizeBit = sizeof (data) * 8;
    memset (binPtr, '\0', sizeof(*binPtr));
    for (i = 0; i < sizeBit; i++) {
        if ((data % 2) == 0) {
            *(binPtr + sizeBit - 1 - i) = '0';
        } else {
            *(binPtr + sizeBit - 1 - i) = '1';
        }
        data = data >> 1;
    }
    *(binPtr + i) = '\0';
}

int main (void)
{
    char vect[33];

    chartoBin ((unsigned char)0x01, &vect[0]);
    printf ("%s\r\n", &vect[0]);

    chartoBin ((unsigned char)0x80, &vect[0]);
    printf ("%s\r\n", &vect[0]);
}
```

```

    chartoBin ((unsigned char)0xA5, &vect[0]);
    printf ("%s\r\n", &vect[0]);

    return (0);
}

```

2. Programa que demuestra el desplazamiento a la izquierda y a la derecha. Imprime la operación realizada con los valores en binario.

```

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

//-- Agregar la funcion chartoBin --

int main (void)
{
    unsigned char entrada;
    unsigned char salida;
    char entradaV[9];
    char salidaV[9];

    entrada = 0x01;    salida = entrada << 1;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s << 1 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x01;    salida = entrada << 2;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s << 2 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x01;    salida = entrada << 4;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s << 4 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x01;    salida = entrada << 7;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s << 7 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x80;    salida = entrada >> 1;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s >> 1 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x80;    salida = entrada >> 2;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s >> 2 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x80;    salida = entrada >> 4;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s >> 4 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0xA0;    salida = entrada >> 7;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s >> 7 : %s\r\n", &entradaV[0], &salidaV[0]);
}

```

```
    return (0);  
}
```

### 3. Programa que demuestra uso del operador not

```
#include <stdio.h>  
#include <string.h>  
  
/-- Agregar la funcion chartoBin --  
  
int main (void)  
{  
    unsigned char entrada;  
    unsigned char salida;  
    char entradaV[9];  
    char salidaV[9];  
  
    entrada = 0x00;    salida = ~entrada;  
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);  
    printf ("~%s : %s\r\n", &entradaV[0], &salidaV[0]);  
  
    entrada = 0xFF;    salida = entrada << 2;  
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);  
    printf ("~%s : %s\r\n", &entradaV[0], &salidaV[0]);  
  
    entrada = 0x01;    salida = entrada << 4;  
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);  
    printf ("~%s : %s\r\n", &entradaV[0], &salidaV[0]);  
  
    entrada = 0xAA;    salida = entrada << 7;  
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);  
    printf ("~%s : %s\r\n", &entradaV[0], &salidaV[0]);  
  
    return (0);  
}
```

#### 4. Programa que demuestra uso del operador and

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

/-- Agregar la funcion chartoBin --

int main (void)
{
    unsigned char entrada;
    unsigned char salida;
    char entradaV[9];
    char salidaV[9];

    entrada = 0xFF;    salida = entrada & 0x01;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s & 0x01 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0xFF;    salida = entrada & 0x02;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s & 0x02 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x02;    salida = entrada & 0x01;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s & 0x01 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x02;    salida = entrada & 0x01;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s & 0x01 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x56;    salida = entrada & 0x0F;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s & 0x0F : %s\r\n", &entradaV[0], &salidaV[0]);

    return (0);
}
```

## 5. Programa que demuestra uso del operador or

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

/-- Agregar la funcion chartoBin --

int main (void)
{
    unsigned char entrada;
    unsigned char salida;
    char entradaV[9];
    char salidaV[9];

    entrada = 0x00;    salida = entrada | 0x01;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s | 0x01 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0xF0;    salida = entrada | 0x02;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s | 0x02 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x02;    salida = entrada | 0x01;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s | 0x01 : %s\r\n", &entradaV[0], &salidaV[0]);

    entrada = 0x50;    salida = entrada | 0x0F;
    chartoBin (entrada, entradaV); chartoBin (salida, salidaV);
    printf ("%s | 0x0F : %s\r\n", &entradaV[0], &salidaV[0]);

    return (0);
}
```

## 6. Programa que demuestra uso de una union

```
#include <stdio.h>

union intCharVect {
    int varInt;
    char vectChar[4];
};

int main (void)
{
    union intCharVect data;

    data.varInt = 0x12345678;

    printf("data.varInt = 0x%x\r\n", data.varInt);
    printf("data.vectChar[0] = 0x%x\r\n", data.vectChar[0]);
    printf("data.vectChar[1] = 0x%x\r\n", data.vectChar[1]);
    printf("data.vectChar[2] = 0x%x\r\n", data.vectChar[2]);
    printf("data.vectChar[3] = 0x%x\r\n", data.vectChar[3]);

    printf("sizeof (data) = %ld\r\n", sizeof (data));
    printf("sizeof (data.varInt) = %ld\r\n", sizeof (data.varInt));
    printf("sizeof (data.vectChar) = %ld\r\n", sizeof (data.vectChar));

    return (0);
}
```

## 7. Programa que demuestra uso de un enum

```
#include <stdio.h>
enum dias_E {
    DOMINGO = 0,
    LUNES,
    MARTES,
    MIERCOLES,
    JUEVES,
    VIERNES,
    SABADO,
};

int main (void)
{
    int dia;

    printf("Ingrese numero indicando el dia de la semana (0 - 6)\r\n")
    scanf("%d", &dia);

    switch (dia) {
        case 0: printf ("Domingo\r\n");    break;
        case 1: printf ("Lunes\r\n");      break;
        case 2: printf ("Martes\r\n");     break;
        case 3: printf ("Miercoles\r\n");  break;
        case 4: printf ("Jueves\r\n");     break;
        case 5: printf ("Viernes\r\n");    break;
        case 6: printf ("Sabado\r\n");     break;

        default:
            printf ("Ingreso invalido\r\n");
            break;
    }

    return (0);
}
```



## Ejercicios

1. Implemente una función que devuelva el valor de un bit indicado como parámetro

```
int leerBit (int palabra, int bit);
```

Donde:

- palabra: Es el valor al cual debe procesarse.
- bit: es el número de bit a leer de palabra

La función retorna un cero si el bit leído es cero o un uno en caso contrario.

2. Implemente una función que niegue el valor de un bit indicado como parámetro

```
int negarBit (int palabra, int bit);
```

Donde:

- palabra: Es el valor al cual debe procesarse.
- bit: es el número de bit a negar de palabra.

La función retorna el valor del bit negado.

3. Implemente las siguientes funciones que escriben un bit, nibble o byte de una palabra.

```
int escribirBit (int palabra, int bit, int datoBit);
```

```
int escribirNibble (int palabra, int nibble, int datoNibble);
```

```
int escribirByte (int palabra, int byte, int datoByte);
```

Donde:

- palabra: Es el valor al cual debe procesarse.
- bit: es el número de bit a setear de palabra
- datoBit: Es el valor a colocar en el bit indicado. Puede tomar el valor cero o uno.
- datoNibble: Es el valor a colocar en el nibble indicado. Puede tomar los valores desde 0x0 hasta 0xF
- datoByte: Es el valor a colocar en el byte indicado. Puede tomar los valores desde 0x00 hasta 0xFF

La función retorna la palabra modificada.

4. Implemente una función que devuelva el valor del nibble indicado como parámetro

```
int leerNibble (int palabra, int nibble);
```

Donde:

- palabra: Es el valor al cual debe procesarse.
- nibble: es el número de nibble a leer de palabra.

5. Implemente una función que devuelva el valor del byte indicado como parámetro

```
int leerByte (int palabra, int byte);
```

Donde:

- palabra: Es el valor al cual debe procesarse.
- byte: es el número de byte a setear de palabra.



## 25. Recursividad.

La recursividad consiste en construir funciones que se llamen a sí mismas.

### 1. Programa que cuenta de forma recursiva.

```
#include <stdio.h>

void contar (int n) {
    printf ("Numero %d\r\n", n);
    if (n > 0) {
        contar (n - 1);
    }
}

int main (void)
{
    int n;

    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &n);

    contar (n);

    return (0);
}
```

### 2. Programa que calcula de forma recursiva el factorial.

```
#include <stdio.h>

int factorial(int n)
{
    if (n == 0) {
        return (1);
    } else {
        return (n * factorial(n - 1));
    }
}

int main (void)
{
    int n, r;

    printf ("Ingrese numero\r\n");
    scanf ("%d", &n);
    r = factorial (n);
    printf ("El factorial de %d es %d\r\n", n, r);
    return (0);
}
```

## Ejercicios

1. Implemente una función que calcule de forma recursiva la potencia  $x^y$  El prototipo de la función es el siguiente:

```
int potencia(int x, int y)
```

2. Implemente una función que calcule la serie de Fibonacci de forma recursiva.
3. Implemente una función que pase a binario en un vector un dato int pasado como parámetro.



## 26. Uso de makefile: compilación y linkeo.

### Programas necesarios

Use apt para instalarlos

Paquete	Descripción
make	Herramienta para asistir en la compilación automática de código fuente.
gdb	Debugger

### Pasos de la compilación

1. Precompilador: En este paso se realizan las siguientes operaciones.
  - Resuelve las directivas que comienzan con # como los includes, ifdef y las macros.
  - Se eliminan los comentarios.Quando se compila con la opción `-save-temps` o `-E` se genera la salida luego de este paso. Suele ser un archivo con extensión `.i`
2. Compilación: Toma la salida del paso anterior y produce una salida con el código a nivel de assembly. Quando se compila con la opción `-save-temps` o `-S` se genera la salida luego de este paso. Suele ser un archivo con extensión `.s`
3. Ensamblado: Toma la salida del paso de compilación y lo traduce a lenguaje de máquina. Quando se compila con la opción `-save-temps` o `-c` se genera la salida luego de este paso. Suele ser un archivo con extensión `.o`
4. Linkeo: Este es el último paso para formar el archivo ejecutable, en esta fase se agrega código para el inicio y la finalización del programa. Además se unen las llamadas a función con su código y las `.`

El gcc es un wrapper que realiza estos pasos para compilar todo.

Guardamos el siguiente código en un archivo llamado `ejemplo.c`

```
#include <stdio.h>

#define CANT ((int)10)

int main (void)
{
    int i;

    for (i = 0; i < CANT; i++) {
        printf ("%d.Hola\r\n", i);
    }

    return (0);
}
```

Ejecutamos el siguiente comando que genera todos los archivos intermedios del proceso de compilación

```
jerome@linuxVm:~$ gcc ejemplo.c -Wall -save-temps -oejemplo.out
jerome@linuxVm:~$ ls
ejemplo.c  ejemplo.i  ejemplo.o  ejemplo.out  ejemplo.s
```

## Compilación de varios .c y .h

Transcriba el siguiente código en un archivo llamado funciones.c

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

int suma (int a, int b)
{
    return(a + b);
}

int resta (int a, int b)
{
    return(a - b);
}

int division (int a, int b)
{
    return(a / b);
}

int multiplicacion (int a, int b)
{
    return(a * b);
}
```

Transcriba el siguiente código en un archivo llamado funciones.h

```
#ifndef FUNCIONES_H
#define FUNCIONES_H

enum op_E {
    SUMA = 0,
    RESTA,
    DIVISION,
    MULTIPLICACION
};

int suma (int a, int b);
int resta (int a, int b);
int division (int a, int b);
int multiplicacion (int a, int b);

#endif
```

Transcriba el siguiente código en un archivo llamado main.c

```

#include <stdio.h>
#include <string.h>
#include "funciones.h"

int main (void)
{
    int a, b, r;
    char op, *p;
    int (*func[4]) (int, int);
    char listOp[] = "+-/*";
    int opIndex;

    //-- Asignacion del puntero a funcion --
    func[SUMA] = suma;
    func[RESTA] = resta;
    func[DIVISION] = division;
    func[MULTIPLICACION] = multiplicacion;

    //-- Ingreso de datos --
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &a);
    printf ("Ingrese operacion\r\n");
    scanf ("%c", &op);
    printf ("Ingrese un numero:\r\n");
    scanf ("%d", &b);

    p = strchr (listOp, op);
    if (p == NULL) {
        printf ("Operacion invalida\r\n");
        return (0);
    }

    opIndex = (int) (p - listOp);
    //-- Uso del puntero a funcion --
    r = func[opIndex] (a, b);
    printf ("El resultado es: %d\r\n", r);

    return (0);
}

```

Compile el código de la siguiente manera

```
jerome@linuxVm:~$ gcc main.c funciones.c -Wall -o ejemplo.out
```

## Makefile

Transcriba el siguiente código en un archivo llamado Makefile y colóquelo junto con los archivos del ejemplo anterior.

```
CC=gcc
EJECUTABLE=main.out
HEADERS=./include

CFLAGS=-c -I$(HEADERS) -Wall
LFLAGS=
LIBS= -lm

all: main.o funciones.o
    $(CC) $(LFLAGS) main.o funciones.o $(LIBS) -o$(EJECUTABLE)

main.o: main.c
    $(CC) $(CFLAGS) main.c -omain.o

funciones.o: funciones.c
    $(CC) $(CFLAGS) funciones.c -ofunciones.o

doxy:
    doxygen Doxyfile
    firefox ./doxy/html/index.html &

clean:
    rm -f ./*.o
    rm -f ./*.out
    rm ./doxy -rf

ejecutar:
    ./ejemplo.out
```

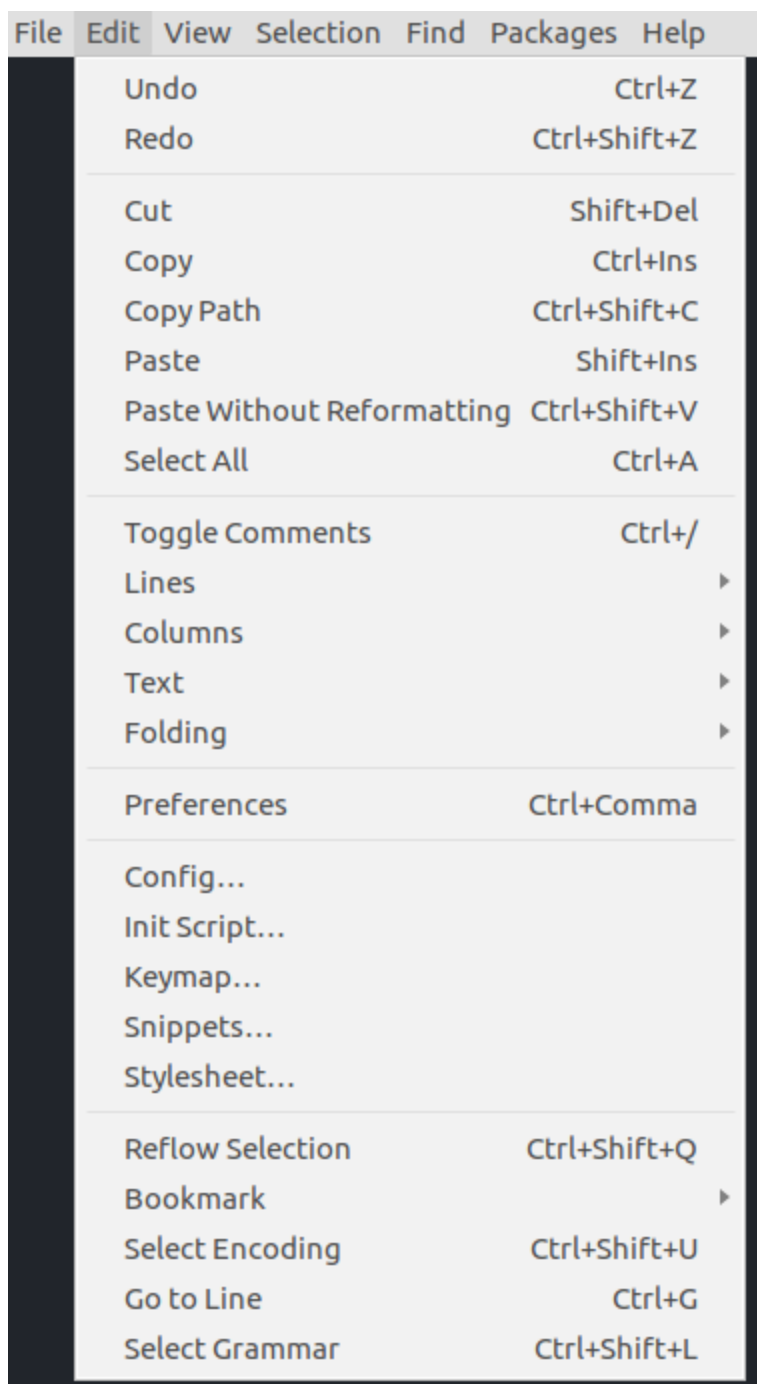
Compile el código de la siguiente manera

```
jerome@linuxVm:~$ Makefile
```

## Como instalar debugger en Atom

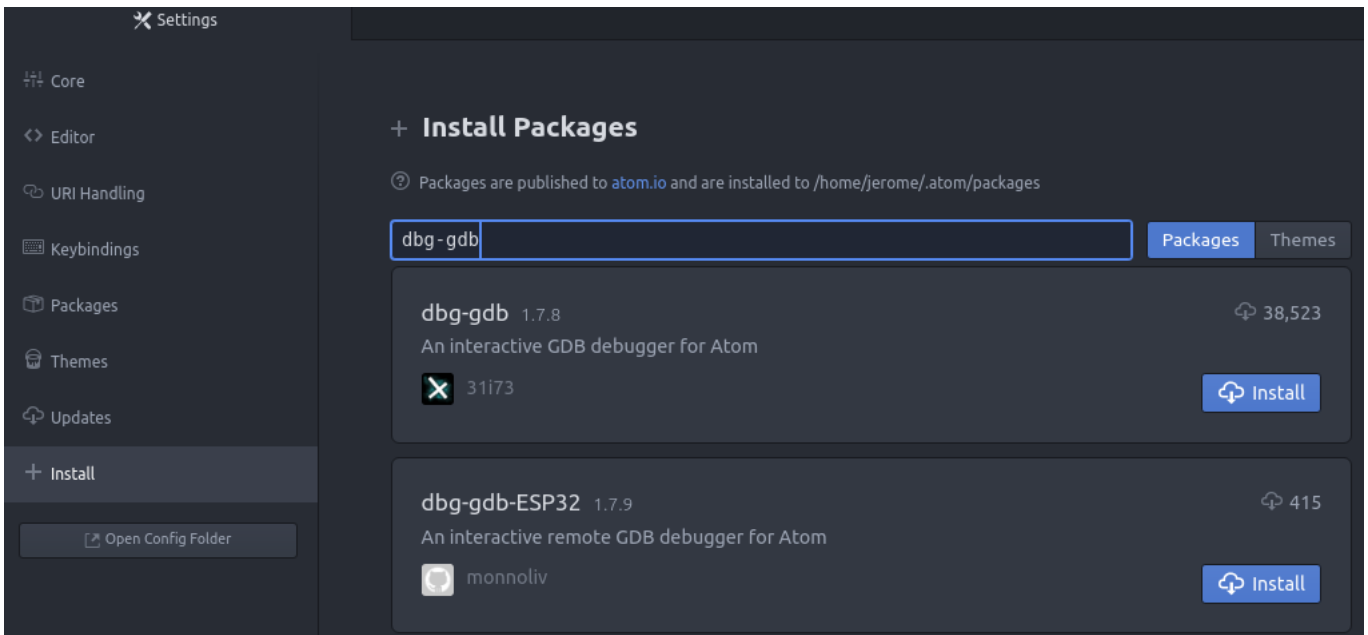
Instalar el paquete de Atom dbg-gdb

1. Entre a las preferencias de Atom Edit->Preferences

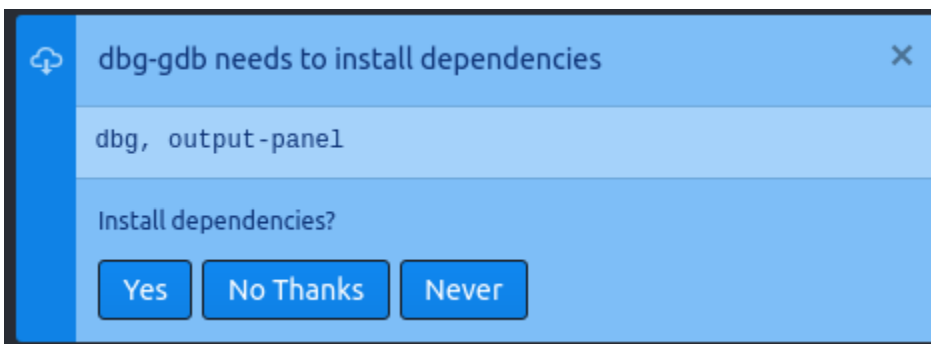




2. Seleccione el menú Install y busque el paquete `dbg-gdb`, luego instálelo.



3. Instale todas las dependencias que la instalación requiera.



## Cómo debuggear un programa en C

1. Copie el siguiente código en un archivo llamado `ejemplo.c`

```
#include <stdio.h>

#define CANT ((int)10)

int main (void)
{
    int i;

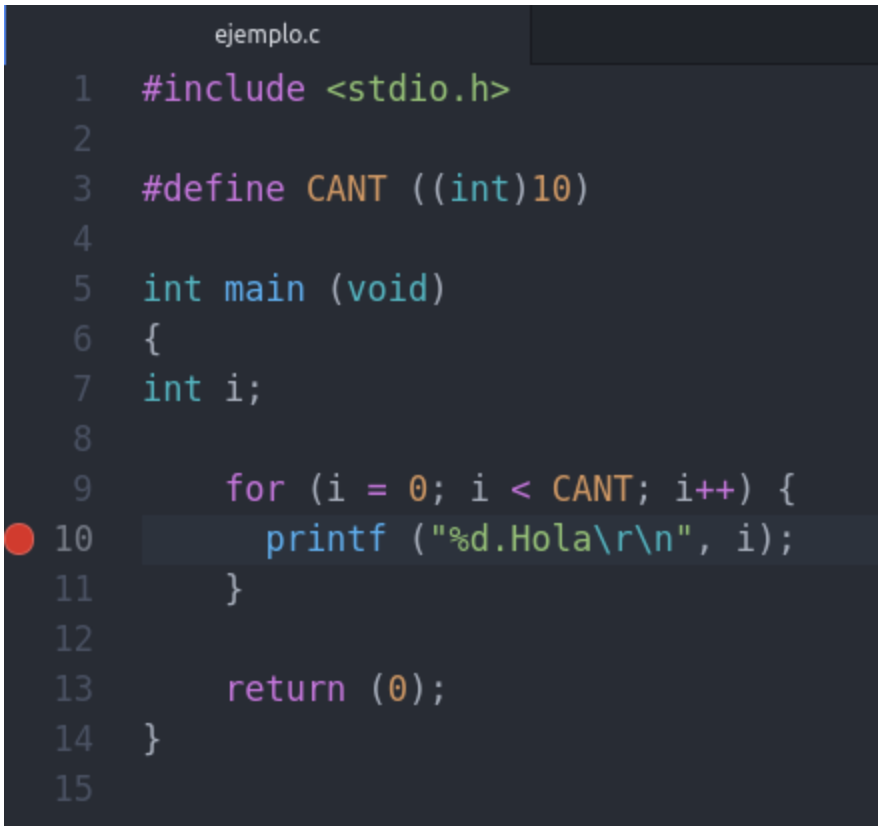
    for (i = 0; i < CANT; i++) {
        printf ("%d.Hola\r\n", i);
    }

    return (0);
}
```

2. Compile el código utilizando la opción -g

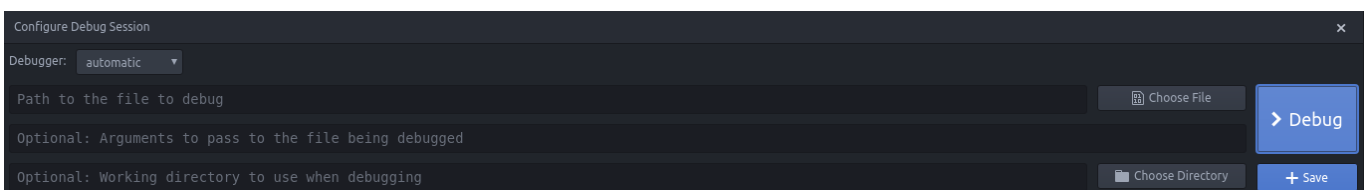
```
jerome@linuxVm:~$ gcc ejemplo.c -Wall -g -oejemplo.out
```

3. Puede colocar breakpoints (puntos de detención) haciendo click a la izquierda del número de línea aparecerá un círculo rojo indicando que el breakpoint pudo colocarse.



```
ejemplo.c
1  #include <stdio.h>
2
3  #define CANT ((int)10)
4
5  int main (void)
6  {
7  int i;
8
9      for (i = 0; i < CANT; i++) {
10     printf ("%d.Hola\r\n", i);
11     }
12
13     return (0);
14 }
15
```

4. Presione la tecla F5 para configurar el debugger. En el panel que aparece seleccione el archivo ejemplo.out que genere en el punto 2



5. Presione el botón Debug para comenzar la ejecución del código
6. Para continuar la ejecución puede presionar la tecla F5

## Library

Es un conjunto de funciones compiladas juntas pensadas para ser reutilizadas.

En linux tenemos de 3 tipos

- Static library: La library se copia en el programa al cual se linkea.
- Dynamic library: Sólo se incluye una referencia a la library, por lo cual se puede modificar la Library sin recompilar todo el programa.

### Static library

Transcriba el siguiente código en un archivo llamado Makefile y colóquelo junto con los archivos del ejemplo anterior.

```
CC=gcc
EJECUTABLE=main.out
HEADERS=./include

CFLAGS=-c -I$(HEADERS) -Wall
LFLAGS=
ARFLAGS=rsc
LIBS= -lm

all: main.o funciones.o
    $(CC) $(LFLAGS) main.o funciones.o $(LIBS) -o$(EJECUTABLE)

main.o: main.c
    $(CC) $(CFLAGS) main.c -omain.o

funciones.o: funciones.c
    $(CC) $(CFLAGS) funciones.c -ofunciones.o

#-- Creo la lib estatica --
libStatic: funciones.o
    $(AR) $(ARFLAGS) libfunciones.a funciones.o

#-- Compilo con la lib estatica --
compileLibStatic: main.o
    $(CC) -o$(EJECUTABLE) main.o -L. -lfunciones

doxy:
    doxygen Doxyfile
    firefox ./doxy/html/index.html &

clean:
    rm -f ./*.o
    rm -f ./*.out
    rm -f ./*.a
    rm ./doxy -rf

ejecutar:
    ./main.out
```

Compile el código de la siguiente manera

```
jerome@linuxVm:~$ Makefile
```

## Dynamic library

```
CC=gcc
EJECUTABLE=main.out
HEADERS=./include

CFLAGS=-c -I$(HEADERS) -Wall
LFLAGS=
ARFLAGS=
LIBS= -lm

all: main.o funciones.o
    $(CC) $(LFLAGS) main.o funciones.o $(LIBS) -o$(EJECUTABLE)

main.o: main.c
    $(CC) $(CFLAGS) main.c -omain.o

funciones.o: funciones.c
    $(CC) $(CFLAGS) funciones.c -ofunciones.o

#-- Creo la lib shared --
libShared: funciones.o
    $(CC) $(CFLAGS) -Werror -fPIC funciones.c
    $(CC) -shared -o libfunciones.so funciones.o

#-- Compilo con la lib shared --
compileLibShared: main.o
    $(CC) -L./ -Wall main.o -o$(EJECUTABLE) -lfunciones
    #export LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH

doxy:
    doxygen Doxyfile
    firefox ./doxy/html/index.html &

clean:
    rm -f ./*.o
    rm -f ./*.out
    rm -f ./*.so
    rm ./doxy -rf
    #unset LD_LIBRARY_PATH

ejecutar:
    ./main.out
```

Compile el código de la siguiente manera

```
jerome@linuxVm:~$ Makefile
```

Agregue el path donde esta la library en la variable de entorno LD\_LIBRARY\_PATH

```
jerome@linuxVm:~$ export LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH
```

Utilizando este comando puede eliminar la configuracion anterior.

```
jerome@linuxVm:~$ unset LD_LIBRARY_PATH
```

■ ■ ■

## 27. Documentación del código: Doxygen e indentación

### Programas necesarios

Use apt para instalarlos

Paquete	Descripción
doxygen	Programa para documentar código fuente en c, c++m etc
doxygen-gui	Interfaz gráfica de doxygen
graphviz	Herramientas para generar gráficos.

### Doxygen

Es un generador de documentación para C, C++, Python, VHDL, etc el cual nos permite contener dentro de los mismos archivos de código fuente la documentación facilitando mantener la documentación actualizada.

### Documentando con doxygen

La documentación en doxygen se realiza directamente sobre los archivos fuente como comentarios utilizando comentarios de bloque y/o en línea especialmente identificados para ser tomados como arte de la documentación y no como un comentario del código.

- Comentario de bloque

```
/**  
    Comentario de bloque  
*/
```

- Comentario en línea

```
/*! Comentario en línea
```

- Comentario en línea para documentar en detalle un define o un campo de una estructura.

```
/*!< Comentario en línea
```

Dentro de estos comentarios se colocan tags para realizar la documentación entre ellos podemos mencionar

- **\file** Para documentar un archivo
- **\fn** Para documentar una función
- **\struct** Para documentar una estructura
- **\enum** Para documentar un enum
- **\brief** Descripción breve del elemento documentado.
- **\detail** Descripción detallada del elemento documentado.
- **\author** Autor del elemento documentado.
- **\date** Fecha del elemento documentado.
- **\version** Versión del elemento documentado.
- **\param** Parámetro de la función documentada
- **\return** Valor de retorno de la

- \todo Agrega el texto a la lista de todo de la documentación

## Ejemplo de documentación de archivo con función main.

```
/**
    \file main.c
    \brief Programa ejemplo que llama a una funcion alojada en otro archivo .c
    \author Ing. Jerónimo F. Atencio (jerome5416@gmail.com)
    \date 2021.05.05
    \version 1
*/
//-----
//-- Includes --
//-----
#include <stdio.h>
#include "oper.h"

/**
    \fn int main (void)
    \brief Pide el ingreso de dos numeros enteros y realiza su suma utilizando
    la funcion operSuma, si los dos operandos son cero sale del programa.
    \author Ing. Jerónimo F. Atencio (jerome5416@gmail.com)
    \date 2021.05.05
    \return Retorna siempre cero.
    \bug No verifica el rango de los números ingresados.
*/
int main(void)
{
    int a, b;
    int resultadoS;
    int resultadoM;

    do {
        //-- Ingreso de datos --
        printf ("Ingrese numero:\r\n");
        scanf ("%d", &a);
        printf ("Ingrese numero:\r\n");
        scanf ("%d", &b);

        //-- llamo a la funcion ---
        resultadoS = operSuma (a, b);
        resultadoM = operMult (a, b);
        //-- Imprimo el resultado de la operacion --
        printf ("\r\n%d + %d = %d\r\n", a,b, resultadoS);
        printf ("\r\n%d * %d = %d\r\n", a,b, resultadoM);

    } while ((a != 0) || (b != 0));

    return (0);
}
```

Guarde este código en un archivo llamado main.c

## Ejemplo de documentación de archivo con funciones varias.

```
/**
    \file oper.c
    \brief Contiene funciones que realizan operaciones matematicas
    \author Ing. Jerónimo F. Atencio (jerome5416@gmail.com)
    \date 2021.05.05
    \version 1
    \todo Agregar mas operaciones.
*/
//-----
//-- Includes --
//-----
#include "oper.h"

/**
    \fn int operSuma (int a, int b)
    \brief Suma dos numeros enteros y retorna el resultado
    \author Ing. Jerónimo F. Atencio (jerome5416@gmail.com)
    \date 2021.05.05
    \param a Primer operando a sumar
    \param b Segundo operando a sumar
    \return Retorna la suma de los operandos pasados como parametros
    \todo Modificar para que verificar overflow.
*/
int operSuma (int a, int b)
{
    return (a + b);
}

/**
    \fn int operMult (int a, int b)
    \brief Multiplica dos numeros enteros y retorna el resultado
    \author Ing. Jerónimo F. Atencio (jerome5416@gmail.com)
    \date 2021.05.05
    \param a Primer operando a sumar
    \param b Segundo operando a sumar
    \return Retorna la suma de los operandos pasados como parametros
*/
int operMult (int a, int b)
{
    return (a * b);
}
```

Guarde este código en un archivo llamado oper.c



## Ejemplo de documentación de archivo .h

```
/**
    \file oper.h
    \brief Prototipos de funciones de oper.c
    \author Ing. Jerónimo F. Atencio (jerome5416@gmail.com)
    \date 2021.05.05
    \version 1.0.0
*/
#ifndef OPER_H
#define OPER_H

//-----
//-- Prototipos --
//-----
int operSuma (int a, int b);
int operMult (int a, int b);

#endif
```

Compilamos el código

```
jerome@linuxVm:~$ gcc -Wall main.c oper.c -osuma.out
```

Generamos el archivo de configuración doxygen.

```
jerome@linuxVm:~$ doxygen -g
Configuration file `Doxyfile' created.

Now edit the configuration file and enter

    doxygen Doxyfile

to generate the documentation for your project
jerome@linuxVm:~$
```

Generamos la documentación

```
jerome@linuxVm:~$ doxygen
```

Abrimos la documentación utilizando el firefox o cualquier browser que utilice.

```
jerome@linuxVm:~$ firefox ./html/index.html
```

■ ■ ■

## 28. Lista simple enlazadas I

Las funciones que aparecen a continuación son las necesarias para operar una lista simple enlazada de un nodo como el siguiente.

```
struct nodo_S {
    int data;
    struct nodo_S *sig;
};
```

### Insertar un nodo al inicio (pila)

Inserta un nuevo nodo al inicio de la lista simple enlazada, el primer parámetro de la función es un puntero a la cabeza de la lista, en el caso de que la lista está vacía este puntero debe ser NULL. El segundo parámetro es el dato a insertar en la lista. La función retorna un puntero a la cabeza de la lista.

```
struct nodo_S * lseInsertarInicio (struct nodo_S *h, int data)
{
    struct nodo_S *p;

    *err = 0;
    //-- Pido memoria --
    p = (struct nodo_S*)malloc (sizeof (struct nodo_S));
    if (p == NULL) {
        //-- No hay memoria --
        return (NULL);
    } else {
        //-- Relleno la estructura --
        p->data = data;
        //-- Pongo el nodo al principio --
        p->sig = h; //-- Si es el primer nodo de la lista h es NULL --
        //-- Apunto el puntero a la cabeza de la lista al nuevo nodo
        h = p;
        return (h);
    }
}
```

## Imprime todos los nodos

Imprime los datos de la lista simple enlazada, incluyendo la dirección del nodo y la dirección del siguiente nodo. El parámetro de la función es el puntero a la cabeza de la lista.

```
void lseImprimir (struct nodo_S *h)
{
    struct nodo_S *p;
    int i = 0;
    //-- Voy recorriendo la lista e imprimiendo nodos --
    puts ("");
    printf("Index\tData\t\tDirNodo\t\tDirNodoSig\r\n");
    for (p = h; p != NULL; p = p->sig) {
        printf("%02d\t%08d\t%p\t%p\r\n", i++, p->data, p, p->sig);
    }
    puts ("");
}
```

## Libera todos los nodos

Libera la memoria que fue asignada dinámicamente de todos los nodos de la lista. El parámetro de la función es el puntero a la cabeza de la lista.

```
struct nodo_S * lseLiberar (struct nodo_S *h)
{
    struct nodo_S *p, *aux;

    if (h == NULL) {
        //-- Lista vacia --
        return (h);
    } else {
        //-- Voy recorriendo la lista y eliminando nodos --
        for (p = h; p != NULL;) {
            /** Debug **
            //printf ("%p\r\n", p);
            /** Debug **
            aux = p;    //-- Salvo el puntero al nodo a eliminar --
            p = p->sig;    //-- Me muevo al siguiente nodo --
            free (aux);    //-- libero el nodo actual --
        }
        h = p; //-- Pongo el puntero a la cabeza de la lista en NULL --
    }
    return (h);
}
```

## Cuenta la cantidad de nodos de la lista simple enlazada

El parámetro de la función es el puntero a la cabeza de la lista y la misma retorna la cantidad de nodos que tiene la lista.

```
int lseContar (struct nodo_S *h)
{
    struct nodo_S *p;
    int i;

    for (p = h, i = 0; p != NULL; p = p->sig, i++);
    return (i);
}
```

## Inserta un nodo al final

Inserta un nuevo nodo al final de la lista simple enlazada, el primer parámetro de la función es un puntero a la cabeza de la lista, en el caso de que la lista está vacía este puntero debe ser NULL. El segundo parámetro es el dato a insertar en la lista. La función retorna un puntero a la cabeza de la lista.

```
struct nodo_S * lseInsertarFinal (struct nodo_S *h, int data)
{
    struct nodo_S *p, *aux;

    //-- Pido memoria --
    p = (struct nodo_S*)malloc (sizeof (struct nodo_S));
    if (p == NULL) {
        //-- No hay memoria --
        return (NULL);
    } else {
        //-- Relleno la estructura --
        p->data = data;
        p->sig = NULL; //-- Como es el ultimo nodo apunto el sig a NULL

        //-- Me fijo donde insertarlo --
        if (h == NULL) {
            //-- La lista esta vacia --
            h = p;
        } else {
            //-- Busco el último nodo --
            for (aux = h; aux->sig != NULL; aux = aux->sig);
            //-- Inserto el nodo al final --
            aux->sig = p;
        }
        return (h);
    }
}
```

## Busca un nodo

El primer parámetro de la función es un puntero a la cabeza de la lista donde se buscará el dato que se encuentra en el segundo parámetro de la función. La función retorna un puntero al nodo que contiene el dato buscado o NULL en caso de que no se encuentre el dato buscado.

```

struct nodo_S * lseBuscar (struct nodo_S *h, int data)
{
    struct nodo_S *p;

    //-- Recorro la lista --
    for (p = h; p != NULL; p = p->sig) {
        if (p->data == data) {
            //-- Encontre el dato --
            return (p);
        }
    }
    //-- Si sali del for es que no encontre nada --
    return (NULL);
}

```

## Remueve un nodo

El primer parámetro de la función es un puntero a la cabeza de la lista donde se buscará el dato que se encuentra en el segundo parámetro de la función, para posteriormente eliminar el nodo. La función retorna un puntero a la cabeza de la lista o NULL en caso de que la lista este vacia.

```

struct nodo_S * lseRemover (struct nodo_S *h, int data)
{
    struct nodo_S *p, *aux;

    //-- Me fijo si hay algo en la lista --
    if (h != NULL) {
        //-- Busco el nodo a eliminar --
        for (p = h, aux = NULL; p != NULL; aux = p, p = p->sig){
            if (p->data == data) {
                //-- Encontre el dato --
                if (aux == NULL) {
                    //-- Es el primer nodo --
                    h = p->sig;
                } else {
                    aux->sig = p->sig;
                }
                free (p);

                return (h);
            }
        }
    } else {
        //-- Lista vacia --
        return (h);
    }
    return (h);
}

```

## Inserta un nodo de forma ordenada

La función inserta el dato pasado como parámetro en una lista simple enlazada ordenada de forma ascendente. Si la lista está creada previamente y está desordenada, esta función no la ordena lo único que hace es ir insertando nuevos nodos de forma ordenada.

```
struct nodo_S * lseInsertarOrdenado (struct nodo_S *h, int data)
{
    struct nodo_S *p, *q, *r;

    //-- Me fijo donde va --
    for (p = h, q = NULL; p != NULL && (data < p->data); q = p, p = p->sig);

    r = (struct nodo_S*)malloc (sizeof (struct nodo_S));
    if (r == NULL) {
        //-- No hay memoria --
        return (NULL);
    } else {
        //-- Relleno la estructura --
        r->data = data;
        r->sig = NULL;

        //-- Inserto el nodo donde va --
        if (q == NULL) {
            //-- Es el primer nodo de la lista --
            r->sig = h;
            h = r;
        } else {
            //-- Va antes del primer nodo --
            r->sig = p;
            q->sig = r;
        }
    }

    return (h);
}
```

## Función main que demuestra el uso de las funciones anteriormente descritas.

```
int main (void)
{
int data[] = {1, 2, 3, 4}; //-- Datos a insertar en la lista --
struct nodo_S *h = NULL;    //-- Puntero a la cabeza de la lista --
struct nodo_S *aux;         //-- Puntero auxiliar --
int i;

    //-- Imprimo los sizeof para poder entender las direcciones impresas --
    printf ("sizeof (struct nodo_S) = %ld\r\n", sizeof (struct nodo_S)
    printf ("sizeof (struct nodo_S*) = %ld\r\n\r\n",sizeof (struct nodo_S*));

    //-- Inserto en la lista --
    for (i = 0; i < 4; i++) {
        aux = lseInsertarInicio (h, data[i]);
        if (aux != NULL) {
            h = aux;
        } else {
            printf ("Error al insertar nodo\r\n");
        }
    }
    //-- Imprimo los datos --
    lseImprimir (h);

    //-- Cuenta la cantidad de nodos --
    printf ("Nodos cant: %d\r\n", lseContar (h));

    //-- Inserto en la lista --
    for (i = 0; i < 4; i++) {
        aux = lseInsertarFinal (h, data[i] + 10);
        if (aux != NULL) {
            h = aux;
        } else {
            printf ("Error al insertar nodo\r\n");
        }
    }

    //-- Imprimo los datos --
    lseImprimir (h);

    //-- Cuenta la cantidad de nodos --
    printf ("Nodos cant: %d\r\n", lseContar (h));

    //-- Busco algo que esta en la lista --
    puts ("\r\nBusco en la lista el número 12:");
    aux = lseBuscar (h, 12);
    if (aux != NULL) {
        printf ("Lo encontré\r\n");
    } else {
        printf ("No lo encontré\r\n");
    }
}
```

```

    //-- Busco algo que no esta en la lista --
    puts ("\r\nBusco en la lista el número 2:");
    aux = lseBuscar (h, 2);
    if (aux != NULL) {
        printf ("Lo encontré\r\n");
    } else {
        printf ("No lo encontré\r\n");
    }

    //-- Remuevo un nodo de la lista --
    puts ("\r\n");
    puts ("Remuevo un nodo de la lista que tenga el numero 3 como dato:");
    h = lseRemover (h, 3);
    lseImprimir (h);

    //-- Liberar los datos --
    puts ("\r\nLibero la memoria pedida");
    h = lseLiberar (h);

    //-- Cuenta la cantidad de nodos --
    printf ("Nodos cant: %d\r\n", lseContar (h));

    return (0);
}

```

## Ejercicios

1. Tome todas las funciones para operar sobre una lista simple enlazada y coloquelas en un archivo llamado lsd.c y cree su archivo de cabecera correspondiente lse.h Luego cree un makefile que compile lse.c como una librería estática. Finalmente agregue una regla en el makefile para compilar una función main que demuestre el uso de las funciones de la lista simple enlazada linkeando la librería antes creada.
2. Implemente una función que lea un archivo de texto y lo inserte de forma ordenada por nombre en una lista simple enlazada.

El formato del archivo es el siguiente:

```

Nombre,edad\r\n
Jose,33\r\n

```

El prototipo de la función es:

```
struct nodo_S * leerArchivo (struct nodo_S *h, char *nombreArchivo)
```

Donde:

- h: Es el puntero a la cabeza de la lista.
- nombreArchivo: Nombre del archivo origen.

La función retorna un número negativo indicando error o la cantidad de nodos de la lista que se pudieron crear.





El nodo de la lista simple enlazada es:

```
#define NOMBRE_SIZE      (32)
struct nodo_S {
    char nombre[NOMBRE_SIZE];
    int edad;
    struct nodo_S *sig;
};
```

3. Implemente una función con el siguiente prototipo que almacene los datos de una lista simple enlazada en un archivo formado por estructuras como la siguiente.

```
#define NOMBRE_SIZE      (32)
struct dato_S {
    char nombre[NOMBRE_SIZE];
    int edad;
};
```

El prototipo de la función es:

```
int leerArchivo (struct nodo_S *h, char *nombreArchivo)
```

Donde:

- h: Es el puntero a la cabeza de la lista.
- nombreArchivo: Nombre del archivo destino

La función retorna un número negativo indicando error o la cantidad de nodos de la lista que se pudieron escribir en el archivo.



## 29. Signals

### Funciones utilizadas

Función	Descripción	Include
getpid	Retorna el PID del proceso que lo ejecuta	sys/types.h unistd.h
getppid	Retorna el PID del padre del proceso que lo ejecuta	sys/types.h unistd.h
sigaction	Es una system call que cambia la acción tomada por un proceso al recibir una signal determinada.	signal.h
sleep	Duerme al thread por una determinada cantidad de segundos	unistd.h
kill	Es una system call que envía una signal a cualquier proceso o grupo de procesos.	sys/types.h signal.h
alarm	Configura SIGALRM para que sea enviada en una cantidad determinada de segundos.	unistd.h
exit	Termina un proceso de forma normal	stdlib.h
pause	Detiene un proceso hasta que llegue una signal	unistd.h

### Comandos

Comando	Descripción
ps	Process Status. Permite visualizar el estado de los procesos.
kill	Sirve para enviar signals a un proceso en ejecución
grep	Toma una expresión regular de la línea de comandos, lee la entrada estándar o una lista de archivo
time	Nos da información acerca de los recursos del sistema usados.

### Programa, proceso

- **Programa:** Secuencia de instrucciones que ejecuta el procesador para realizar alguna operación o procesamiento de datos.
- **Proceso:** Es una instancia de un programa en ejecución. Cada proceso es identificado por el sistema operativo por un número llamado PID (Process ID)

## Ejemplos

1. Programa imprime en pantalla su propio PID (Process ID) y el PID del proceso padre (PPID, Parent Process ID)

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main (void)
{
    pid_t pid, ppid;

    /*-- Obtengo mi pid y el pid del padre --
    pid = getpid ();
    printf ("PID: %d\r\n", pid);
    ppid = getppid ();
    printf ("PPID: %d\r\n", ppid);

    return (0);
}
```

2. Programa en el que se imprime la leyenda "Hola signal" al recibir la signal SIGUSR1.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

void handler_SIGUSR1 (int mySignal)
{
    printf ("Hola signal %d\r\n", mySignal);
}

int main (void)
{
    int r;
    struct sigaction s;

    /*-- Relleno la estructura sigaction --
    s.sa_handler = handler_SIGUSR1;
    s.sa_flags = 0;
    sigemptyset(&s.sa_mask);

    r = sigaction (SIGUSR1, &s, NULL);
    if (r < 0) {
```

```

        //-- Error --
        printf ("sigaction error: %d> %s\r\n", errno, strerror(errno));
        return (-1);
    } else {
        printf ("Espero a SIGUSR1\r\n");
        while (1) {
            sleep (1);
        }
    }

    return (0);
}

```

Escriba y guarde el código anterior en un archivo llamado ejemplo27\_02.c

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo27_02.c -Wall -oejemplo27_02.out
jerome@linuxVm:~$ ./ejemplo27_02.out

```

Utilice otra terminal para enviarle SIGUSR1 al programa.

```

jerome@linuxVm:~$ ps -e | grep ejemplo27_02.out
PPPP pts/0      00:00:00 a.out
jerome@linuxVm:~$ kill -SIGUSR1 PPPP

```

En este ejemplo el PID está representado por **PPPP**. Para enviar la señal SIGUSR1 el pid devuelto por el comando ps es el usado con el comando kill

### 3. Programa que genera una violacion de segmento y es capturada la signal SIGSEGV.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdlib.h>

void handler_SIGSEGV (int mySignal)
{
    printf ("Violacion de segmento %d\r\n", mySignal);
    exit (0);
}

int main (void)
{
    int r;
    struct sigaction s;
    int *p = NULL;

    //-- Relleno la estructura sigaction --
    s.sa_handler = handler_SIGSEGV;
    s.sa_flags = 0;
    sigemptyset(&s.sa_mask);

    r = sigaction (SIGSEGV, &s, NULL);
    if (r < 0) {
        //-- Error --
        printf ("sigaction error: %d> %s\r\n", errno, strerror(errno));
        return (-1);
    } else {
        printf ("Apunto de generar una violacion de segmento\r\n");
        *p = 0;
    }
    return (0);
}
```

#### 4. Programa que genera una excepción de división por cero y es capturada la signal SIGFPE.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdlib.h>

void handler_SIGFPE (int mySignal)
{
    printf ("Excepcion de punto flotante %d\r\n", mySignal);
    exit (0);
}

int main (void)
{
    int r;
    struct sigaction s;

    //-- Relleno la estructura sigaction --
    s.sa_handler = handler_SIGFPE;
    s.sa_flags = 0;
    sigemptyset(&s.sa_mask);

    r = sigaction (SIGFPE, &s, NULL);
    if (r < 0) {
        //-- Error --
        printf ("sigaction error: %d> %s\r\n", errno, strerror(errno));
        return (-1);
    } else {
        printf ("Calculo 1 / 0\r\n");
        //-- Division por cero en enteros genera excepcion de punto flotante --
        r = 1 / 0;

        printf ("%d\r\n", r);
    }

    return (0);
}
```

5. Programa que configura la función alarma para que se ejecute el handler de SIGALRM cada 5 segundos

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdlib.h>

void handler_SIGALRM (int mySignal)
{
    printf ("Alarma %d\r\n", mySignal);
    alarm (5);  //-- Configuro la alarma en 5 segundos --
}

int main (void)
{
    int r;
    struct sigaction s;

    //-- Relleno la estructura sigaction --
    s.sa_handler = handler_SIGALRM;
    s.sa_flags = 0;
    sigemptyset(&s.sa_mask);
    r = sigaction (SIGALRM, &s, NULL);
    if (r < 0) {
        //-- Error --
        printf ("sigaction error: %d> %s\r\n", errno, strerror(errno));
        return (-1);
    } else {
        //-- Configuro la alarma para dentro de 5 segundos --
        alarm (5);
        do {
            pause ();
        } while (1);
    }
    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo27\_05.c

Compilando y ejecutando el programa.

```
jerome@linuxVm:~$ gcc ejemplo27_05.c -Wall -oejemplo27_05.out
jerome@linuxVm:~$ ./ejemplo27_05.out
```

Puede ejecutar el programa utilizando el comando time previamente y ver los recursos de sistema utilizados

```
jerome@linuxVm:~$ time ./ejemplo27_05.out
```

## 6. Programa que captura la presión de las teclas CTRL-C del teclado

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdlib.h>

void handler_SIGINT (int mySignal)
{
    printf ("Presionaron CTRL-C %d\r\n", mySignal);
}

int main (void)
{
    int r;
    struct sigaction s;

    //-- Relleno la estructura sigaction --
    s.sa_handler = handler_SIGINT;
    s.sa_flags = 0;
    sigemptyset(&s.sa_mask);

    r = sigaction (SIGINT, &s, NULL);
    if (r < 0) {
        //-- Error --
        printf ("sigaction error: %d> %s\r\n", errno, strerror(errno));
        return (-1);
    } else {
        //-- Configuro la alarma para dentro de 5 segundos --
        do {
            sleep (1);
        } while (1);
    }
    return (0);
}
```



## 7. Programa que ejemplifica el uso de kill para enviar la SIGUSR1

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>

int main (int argc, char *argv[])
{
    int r;
    //-- Verifico los argumentos --
    if (argc < 2) {
        printf ("Uso ./enviarSignal PID\r\n");
        return (-1);
    }

    r = kill(atoi(argv[1]), SIGUSR1);
    if (r != 0) {
        printf ("Error kill\r\n");
        return (-2);
    }

    printf ("Se pudo enviar la SIGUSR1\r\n");

    return (0);
}
```

Escriba y guarde el código anterior en un archivo llamado ejemplo28\_11.c

En una terminal ejecute el ejemplo28\_11.out

```
jerome@linuxVm:~$ ./ejemplo28_11.out
```

Utilice otra terminal compile el programa

```
jerome@linuxVm:~$ gcc -Wall ejemplo28_11.c -oejemplo28_11.out
jerome@linuxVm:~$ ps -e | grep ejemplo28_11.out
PPPP pts/0    00:00:00 a.out
jerome@linuxVm:~$ ./ejemplo28_12.out PPPP
```

En este ejemplo el PID está representado por **PPPP**.

## Ejercicios

1. Implemente un programa que imprima la leyenda "Hola Mundo" cinco segundos después de que se presione CTRL-C. (use alarm)
2. Implemente un programa imprima una leyenda cada cierto tiempo. La cantidad de segundos que transcurren entre cada impresión es ingresada por línea de comandos en segundos.



## 30. Threads

### Funciones utilizadas

Función	Descripción	Include
pthread_create	Crea un thread	pthread.h
pthread_join	Espera que un thread específico termine	pthread.h
pthread_exit	Finaliza un thread	pthread.h
pthread_cancel	Cancela un thread	pthread.h
pthread_mutex_lock		pthread.h
pthread_mutex_unlock		pthread.h

### Ejemplos

#### 1. Crea un hilo de ejecución nuevo

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

void * threadFunc1 (void *p)
{
    int i;
    for (i = 0; i < 6; i++) {
        sleep (1);
        printf ("Th1: Paso 1 segundo\r\n");
    }

    return (NULL);
}

void * threadFunc2 (void *p)
{
    int i;

    for (i = 0; i < 2; i++) {
        sleep (1);
        printf ("Th2: Paso 1 segundo\r\n");
    }
    return (NULL);
}

int main (void)
{
    pthread_t t1;
    void *t1sRet;
    int rVal;
```

```

    rVal = pthread_create(&t1, NULL, threadFunc1, NULL);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }

    threadFunc2 (NULL);

    rVal = pthread_join (t1, &tlsRet);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }
    return (0);
}

```

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo29_01.c -Wall -lpthread -oejemplo29_01.out
jerome@linuxVm:~$ ./ejemplo29_01.out

```

Se agrega la directiva de linker -l junto con el nombre de la librería a incluir en este caso pthread. Por ello se observa -lpthread que debe ser agregado para linkear las funciones de la librería de threads.

**Advertencia:**  
No olvide agregar las library necesarias para linkear su código. Use la directiva -l

## 2. Muestra como pasar y recibir datos de un thread

```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

void * threadFunc (void *p)
{
    int i;
    int *val;
    int *q;

    val = (int*)p;
    printf ("Th: val = %d\r\n", *val);
    (*val)++;

    q = (int*)malloc (1 * sizeof(int));
    if (q != NULL) {
        *q = 27;
    }
    pthread_exit (q);
}

int main (void)

```

```

{
pthread_t t;
void *tRet;
int rVal;
int paramVal;

    paramVal = 100;
    printf ("main: paramVal = %d\r\n",    paramVal);

    rVal = pthread_create(&t, NULL, threadFunc, &paramVal);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }

    rVal = pthread_join (t, &tRet);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }

    printf ("main: *tRet = %d\r\n", *((int*)tRet));
    printf ("main: paramVal = %d\r\n", paramVal);

    return (0);
}

```

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo29_02.c -Wall -lpthread -oejemplo29_02.out
jerome@linuxVm:~$ ./ejemplo29_02.out

```

### 3. Muestra el uso de mutex

```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

int c = 0;
pthread_mutex_t lock;

void * threadFunc (void *p)
{
    int i;
    pthread_mutex_lock(&lock);
    for (i = 0; i < 1e8; i++) {
        c++;
    }
    pthread_mutex_unlock(&lock);
}

```

```

int main (void)
{
pthread_t t0, t1;
void *tRet;
int rVal;

    rVal = pthread_create(&t0, NULL, threadFunc, NULL);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }

    rVal = pthread_create(&t1, NULL, threadFunc, NULL);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }

    rVal = pthread_join (t0, &tRet);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }

    rVal = pthread_join (t1, &tRet);
    if (rVal != 0) {
        printf ("Error pthread_create\r\n");
        return (-1);
    }

    printf ("c: %d\r\n", c);
    return (0);
}

```

Compilando y ejecutando el programa.

```

jerome@linuxVm:~$ gcc ejemplo29_03.c -Wall -lpthread -oejemplo29_03.out
jerome@linuxVm:~$ ./ejemplo29_03.out

```

■ ■ ■

## 31. Sockets

### Funciones utilizadas

Función	Descripción	Include
gethostbyname	Asigna dinámicamente memoria.	#include <netdb.h> #include <sys/socket.h>
inet_ntoa	Función para manipulación de direcciones de internet	#include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h>
inet_ntop	Convierte una dirección IPv4 o IPv6 de binario a texto.	#include <arpa/inet.h>
getaddrinfo	Traducción de servicios y direcciones de red	#include <sys/types.h> #include <sys/socket.h> #include <netdb.h>
socket	Crea un endpoint de una comunicación	#include <sys/types.h> #include <sys/socket.h>
setsockopt	Obtiene y configura opciones en los sockets	#include <sys/types.h> #include <sys/socket.h>
bind	Enlaza el nombre a un socket	#include <sys/types.h> #include <sys/socket.h>
listen	Espera conexiones de un socket	#include <sys/types.h> #include <sys/socket.h>
accept	Acepta una conexión a un socket	#include <sys/types.h> #include <sys/socket.h>
connect	Inicia una conexión a un socket	#include <sys/types.h> #include <sys/socket.h>
send	Envía un mensaje por un socket	#include <sys/types.h> #include <sys/socket.h>
recv	Recibe un mensaje en un socket	#include <sys/types.h> #include <sys/socket.h>

### Ejemplos

1. Junto con esta guía encontrará una carpeta con los siguientes archivos

- Makefile: Makefile del ejemplo
- TCP\_clientMain.c : Ejemplo del uso del cliente.
- TCP\_serverMain.c: Ejemplo del uso del servidor.
- TCP\_clientServer.c: Código del cliente-servidor TCP
- TCP\_clientServer.h: Definiciones necesarias para el cliente-servidor TCP

En este ejemplo una vez que el servidor está en ejecución, se puede iniciar un cliente el cual quedará a la espera de que el usuario ingrese una palabra. La palabra ingresada será enviada al servidor el cual la

pasará a mayúscula y la devolverá al cliente. Finalmente el cliente la imprime en pantalla y espera el ingreso de una nueva palabra.

Todo el proyecto está configurado para conectarse por la interfaz de loopback en el puerto 5000 así que no es necesario dos computadoras para la prueba, basta con tener dos terminales abiertas. El servidor puede aceptar más de una conexión simultánea.

Compilando el cliente y el servidor

```
jerome@linuxVm:~$ make
```

En una terminal ejecutamos el servidor

```
jerome@linuxVm:~$ ./server.out
```

En una terminal ejecutamos el cliente

```
jerome@linuxVm:~$ ./client.out
```

## Ejercicios

1. Implemente dos programas uno cliente TCP y otro servidor TCP
  - a. El cliente se conectará al servidor y recibirá un número entero que imprimirá en pantalla antes de finalizar su ejecución.
  - b. El servidor cada vez que se conecte un cliente deberá devolverle un número aleatorio y se quedará esperando una nueva conexión.
2. Implemente dos programas uno cliente TCP y otro servidor TCP
  - a. El cliente esperará el ingreso de un texto por teclado el que enviará al servidor. Luego se quedará esperando la respuesta del mismo para imprimir lo devuelto en pantalla.
  - b. El servidor recibirá un texto el cual deberá convertirlo a morse y enviarlo de regreso al cliente.
3. Implemente dos programas uno cliente TCP y otro servidor TCP
  - a. El cliente leerá un archivo de texto el cual será enviado al servidor y luego se quedará esperando la devolución de los datos por parte del main y los almacenará en otro archivo.
  - b. El servidor esperará conexiones de clientes y convertirá todos los caracteres a mayúscula y se los devolverá al cliente.



## 32. Interfaces visuales: Qt

### Programas necesarios

Use apt para instalarlos

Paquete	Descripción
qt5-default	QT 5
qtcreator	Paquete de instalación de qt creator
qtbase5-examples	Ejemplos de QT
qt5-doc	Documentación
qt5-doc-html qtbase5-doc-html	Documentación en html

### Ejecución

qtcreator

