

## PRÁCTICA 3

**Tema: programación en C y C++ usando el terminal linux**

Estudiante	Escuela	Asignatura
Rodrigo Infanzón Acosta rinfanzona@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Sistemas Operativos

Informe	Tema	Duración
04	programación en C y C++ usando el terminal linux	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - B	20/09/24	27/09/24

### Índice

<b>1. Objetivos</b>	<b>2</b>
<b>2. Recursos y herramientas utilizados</b>	<b>2</b>
<b>3. URL de repositorio Github</b>	<b>2</b>
<b>4. GCC y G++</b>	<b>3</b>
4.1. Creación de un directorio general . . . . .	3
4.2. Instalar gedit en linux . . . . .	3
4.3. Crear, editar, compilar y ejecutar un código .c . . . . .	4
4.4. Crear, editar, compilar y ejecutar un código .cpp . . . . .	8
<b>5. Argumentos en la línea de comandos</b>	<b>10</b>
5.0.1. Ejemplo de uso . . . . .	11
<b>6. Comando make</b>	<b>12</b>
6.0.1. Ejemplo de uso . . . . .	12
<b>7. Ejercicios propuestos</b>	<b>15</b>
7.1. Código 1 . . . . .	15
7.2. Código 2 . . . . .	17
7.3. Código 3 . . . . .	18
7.4. Código 4 . . . . .	25
7.5. Código 5 . . . . .	27
<b>8. Cuestionario</b>	<b>31</b>

## 1. Objetivos

- El estudiante comprenderá el funcionamiento interno de los sistemas operativos desde la utilización de comandos.
- El estudiante deberá de probar, analizar y explicar el funcionamiento de las herramientas propuestas.

## 2. Recursos y herramientas utilizados

- Sistema operativo utilizado: Windows 10 pro 22H2 de 64 bits SO. 19045.4170.
- Hardware: Ryzen 7 5700X 4.0 GHz, RAM 32 GB DDR4 3200 MHz, RTX 4060 Asus Dual.
- Virtual Box 7.0.20-163906-Win
- Git 2.44.0.
- Sistema invitado utilizado: Linux Mint 22 Virginia Cinnamon
- Conocimientos básicos en Git.
- Conocimientos básicos en sistemas operativos.
- Conocimientos básicos en Linux.
- C y C++ (gcc y g++).

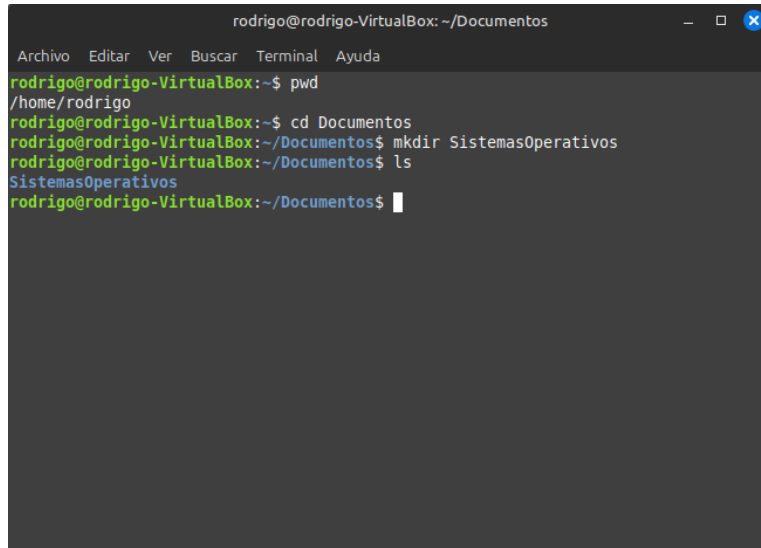
## 3. URL de repositorio Github

- URL para la práctica 3 en el repositorio GitHub:
- <https://github.com/RodrigoStranger/sistemas-operativos-24b/tree/main/Practica%203>

## 4. GCC y G++

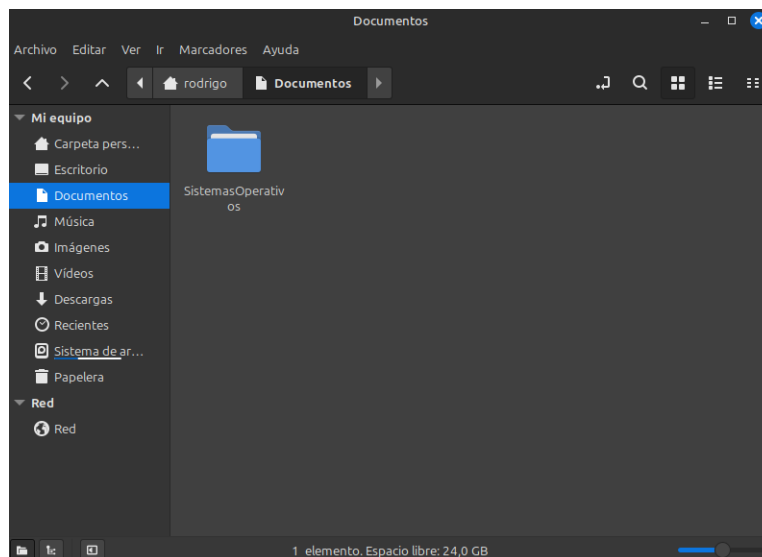
### 4.1. Creación de un directorio general

En primer lugar, creamos nuestro directorio en linux denominado SistemasOperativos, este estará situado dentro del directorio Documentos:



```
rodrigo@rodrigo-VirtualBox: ~/Documentos
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
rodrigo@rodrigo-VirtualBox:~$ pwd
/home/rodrigo
rodrigo@rodrigo-VirtualBox:~$ cd Documentos
rodrigo@rodrigo-VirtualBox:~/Documentos$ mkdir SistemasOperativos
rodrigo@rodrigo-VirtualBox:~/Documentos$ ls
SistemasOperativos
rodrigo@rodrigo-VirtualBox:~/Documentos$
```

Se puede apreciar con ayuda del visor de archivos que bajo la carpeta Documentos se ha creado la carpeta denominada SistemasOperativos:



### 4.2. Instalar gedit en linux

Opcionalmente, podemos usar el editor de texto gedit, nano, u otros editores de texto disponibles. Si no tenemos instalado gedit, podemos hacerlo mediante el comando en linux, **sudo apt install gedit**, ejemplo:

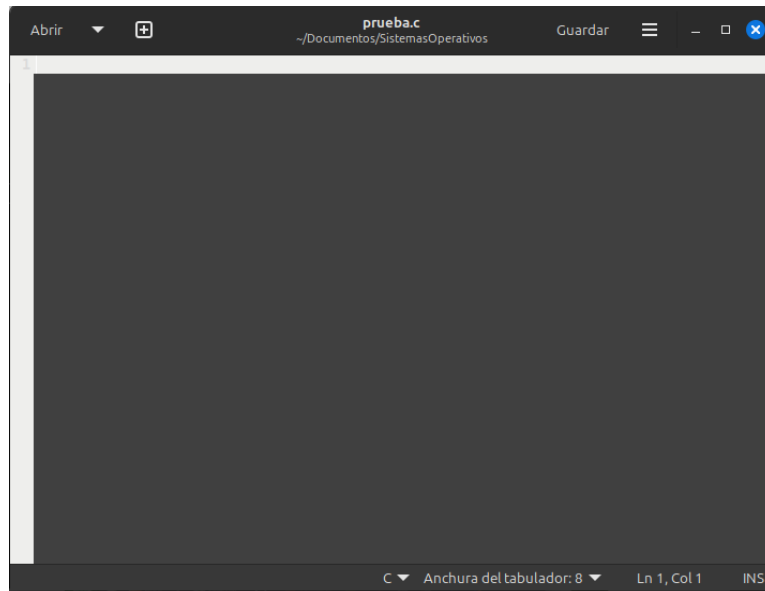
```
rodrigo@rodrigo-VirtualBox: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
rodrigo@rodrigo-VirtualBox:~$ sudo apt install gedit  
[sudo] contraseña para rodrigo:  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias... Hecho  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
  gedit-common gir1.2-amtk-5 gir1.2-gtksource-300 gir1.2-tepl-6  
  libgedit-amtk-5-0 libgedit-amtk-5-common libgedit-gtksourceview-300-0  
  libgedit-gtksourceview-300-common libtepl-6-4 libtepl-common  
Paquetes sugeridos:  
  gedit-plugins  
Se instalarán los siguientes paquetes NUEVOS:  
  gedit gedit-common gir1.2-amtk-5 gir1.2-gtksource-300 gir1.2-tepl-6  
  libgedit-amtk-5-0 libgedit-amtk-5-common libgedit-gtksourceview-300-0  
  libgedit-gtksourceview-300-common libtepl-6-4 libtepl-common  
0 actualizados, 11 nuevos se instalarán, 0 para eliminar y 0 no actualizados.  
Se necesita descargar 2.566 kB de archivos.  
Se utilizarán 17,4 MB de espacio de disco adicional después de esta operación.  
¿Desea continuar? [S/n] s  
Des:1 http://archive.ubuntu.com/ubuntu noble/universe amd64 gedit-common all 46.  
2-2 [1.480 kB]  
Des:2 http://archive.ubuntu.com/ubuntu noble/universe amd64 libgedit-gtksourcevi  
ew-300-common all 299.0.4-3build1 [282 kB]  
Des:3 http://archive.ubuntu.com/ubuntu noble/universe amd64 libgedit-gtksourcevi
```

### 4.3. Crear, editar, compilar y ejecutar un código .c

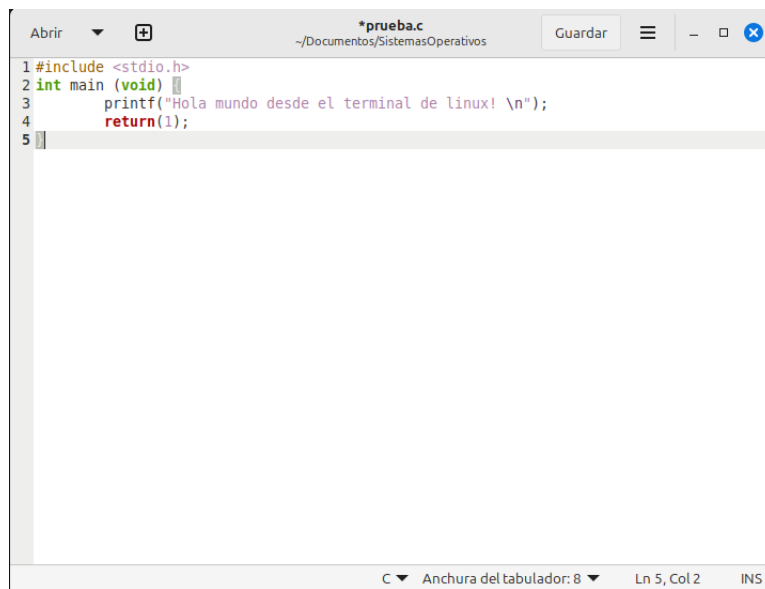
Estando dentro del directorio SistemasOperativos, crearemos nuestro código hecho para C:

```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos  
Archivo Editar Ver Buscar Terminal Ayuda  
rodrigo@rodrigo-VirtualBox:~/Documentos$ cd SistemasOperativos  
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gedit prueba.c
```

Se nos abrirá gedit, en la cual posee la siguiente interfaz gráfica:

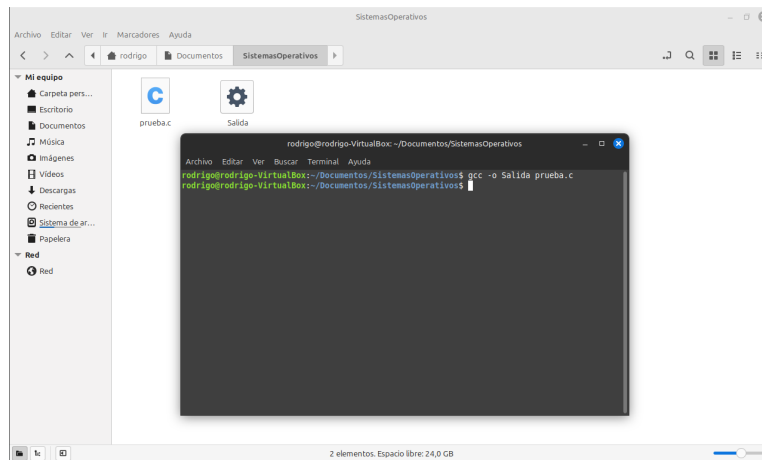


Dentro, escribimos el siguiente código:



Le damos en guardar y cerramos la interfaz gráfica.

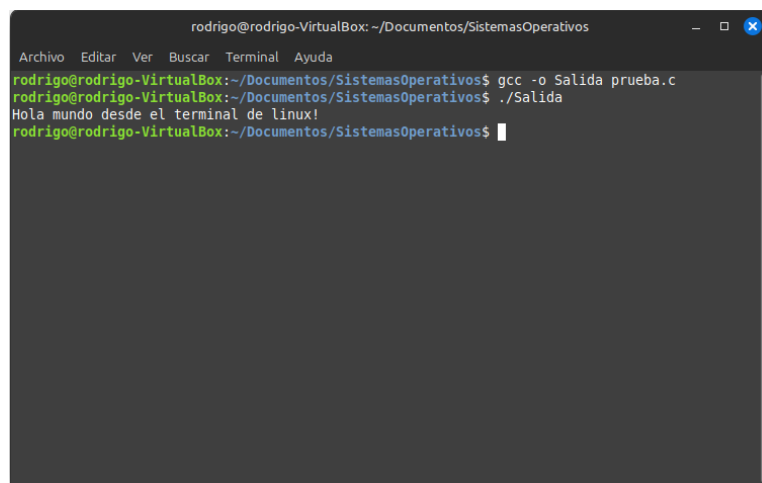
En la terminal de linux, para compilar el código, utilizamos la herramienta gcc. Para ello, se escribe el siguiente comando: **gcc -o Salida prueba.c.**



-o Salida le indica al compilador gcc que genere un archivo ejecutable con el nombre Salida a partir del código fuente en prueba.c

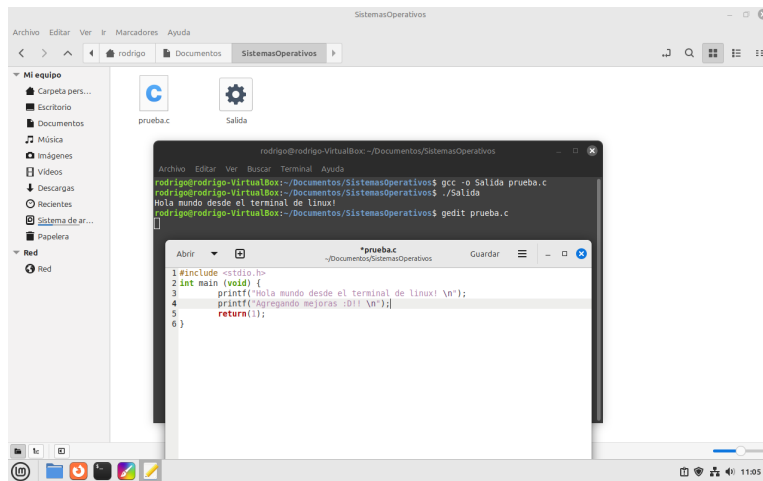
Si no se usa -o, el compilador crearía un ejecutable con el nombre predeterminado a.out.

Para poder ejecutar el programa.c, se escribe en la terminal el comando: **./Salida**



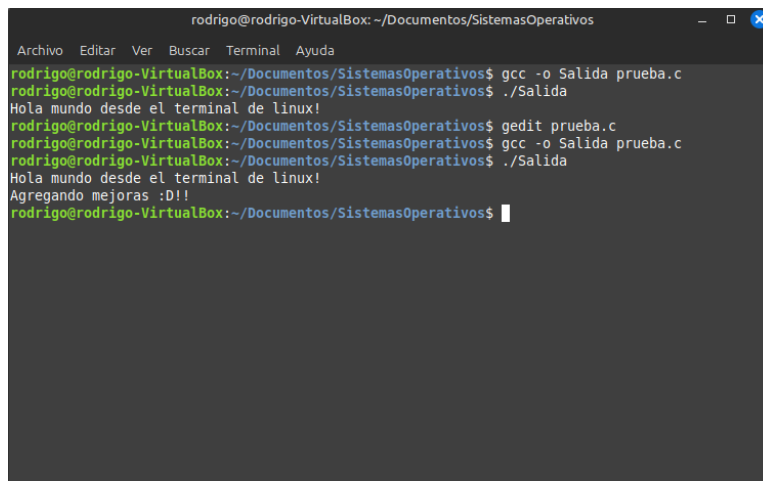
En este caso nos muestra un mensaje ya programado en prueba.c

A partir de este proceso ya se pueden hacer mejoras al código, solo hay que volver a compilar y ejecutar el archivo resultante nuevamente:

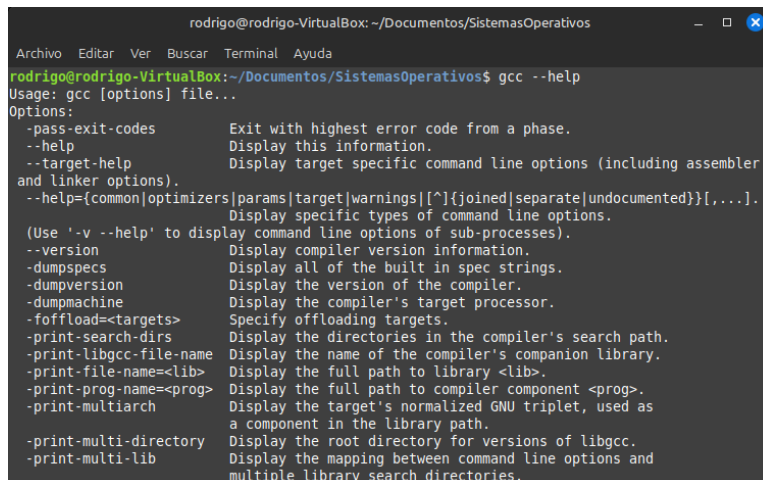


Le damos en guardar y cerramos la interfaz gráfica.

Ejecutamos de la misma manera el código prueba.c, nos muestra la siguiente salida:



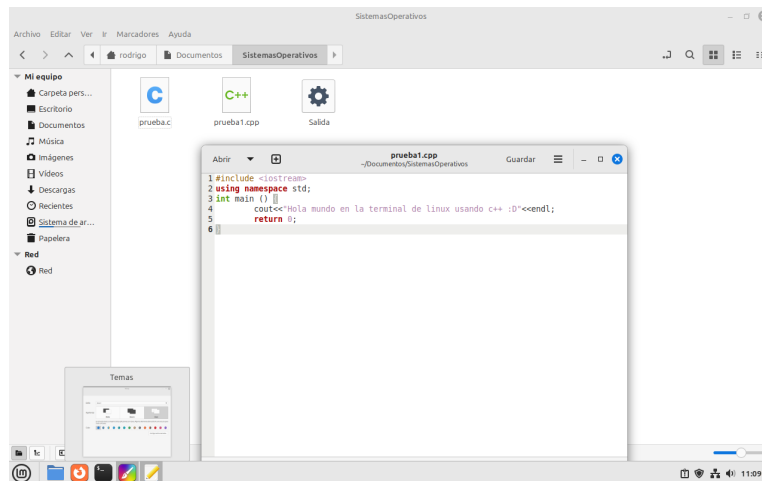
Para poder obtener más ayuda o consultas acerca del compilador gcc, usamos el comando: **gcc --help**



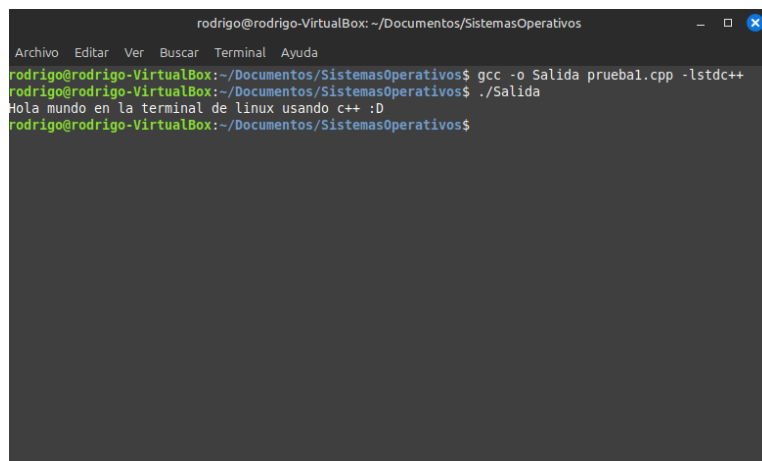
#### 4.4. Crear, editar, compilar y ejecutar un código .cpp

Para ejecutar códigos del lenguaje c++, usaremos el compilador gcc, para esto se tiene que crear un nuevo archivo, en esta ocasión con la extensión .cpp y el siguiente código:

Creamos el archivo prueba1.cpp y lo editamos en gedit:



Ejecutamos:





Se puede compilar código estándar de c++ con la herramienta gcc, pero se recomienda el uso de otro compilador más especializado. Para esto, eliminaremos el archivo generado del proceso de compilación tal como se muestra en la siguiente imagen:

```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -o Salida pruebal.cpp -lstdc++
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./Salida
Hola mundo en la terminal de linux usando c++ :D
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ rm Salida
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ g++ -o Salida pruebal.cpp
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./Salida
Hola mundo en la terminal de linux usando c++ :D
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$
```

Por último, de igual manera que la herramienta gcc, se puede solicitar todas las opciones disponibles de la herramienta g++ usando el comando **g++ --help**

```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ g++ --help
Usage: g++ [options] file...
Options:
  -pass-exit-codes      Exit with highest error code from a phase.
  --help                Display this information.
  --target-help          Display target specific command line options (including assembler and
linker options).
  --help={common|optimizers|params|target|warnings|[^]{}|joined|separate|undocumented}}[,...].
                        Display specific types of command line options.
                        (Use '-v --help' to display command line options of sub-processes).
  --version             Display compiler version information.
  -dumpspecs            Display all of the built in spec strings.
  -dumpversion          Display the version of the compiler.
  -dumpmachine          Display the compiler's target processor.
  -foffload=<targets>   Specify offloading targets.
  -print-search-dirs    Display the directories in the compiler's search path.
  -print-libgcc-file-name
                        Display the name of the compiler's companion library.
  -print-file-name=<lib>
                        Display the full path to library <lib>.
  -print-prog-name=<prog>
                        Display the full path to compiler component <prog>.
  -print-multiarch      Display the target's normalized GNU triplet, used as
                        a component in the library path.
  -print-multi-directory
                        Display the root directory for versions of libgcc.
  -print-multi-lib      Display the mapping between command line options and
                        multiple library search directories.
```

## 5. Argumentos en la línea de comandos

Muchos programas en Linux obtienen información a través de argumentos introducidos por la línea de comandos, y si los programas están preparados para aceptarlos, podrán tenerlos en cuenta.

Por ejemplo, en la siguiente orden `ls` se va a ejecutar con dos argumentos: `ls -l /tmp`

```

rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ls -l /tmp
total 48
-rw-r--r-- 1 rodrigo rodrigo 0 set 22 10:21 config-err-7XwiHp
drwxrwxrwx 2 rodrigo rodrigo 4096 set 22 10:21 initrd.img
-rw-r--r-- 1 rodrigo rodrigo 0 set 22 10:23 MozillaUpdateLock-4F96D1932A9F858E
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
colord.service-x7xUM6
drwx----- 3 root root 4096 set 22 10:52 systemd-private-dc70311bf12343119a8ead0a691af618-
fwupd.service-JgyMTU
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
ModemManager.service-v0Er3d
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
palkit.service-Bn46x5
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
switcheroo-control.service-FY06SL
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
systemd-logind.service-uxoLmk
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
systemd-resolved.service-E4KbxD
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
systemd-timesyncd.service-TSpb5H
drwx----- 3 root root 4096 set 22 10:21 systemd-private-dc70311bf12343119a8ead0a691af618-
upower.service-4LX1zw
drwx----- 2 rodrigo rodrigo 4096 set 22 10:23 Temp-4ee50873-ec83-4dcd-9816-138542a32f7e
drwxrwxrwt 2 root root 4096 set 22 10:21 VMwareDm0
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$

```

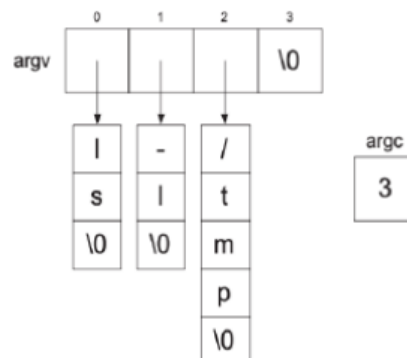
El comando `ls -l /tmp` le dice al programa `ls` que muestre el contenido del directorio `/tmp` con detalles adicionales, en formato de lista larga.

Para que estos argumentos puedan ser tenidos en cuenta en nuestro programa `C`, tenemos que invocar a la función `main` con dos argumentos: `argc` y `argv`:

```
void main(int argc, char *argv[] ) { ... }
```

En este caso, el sistema operativo pasa dos parámetros a la función principal del programa. `argc` es un entero que indica el número de argumentos en la línea de comandos, y `argv` es un array de punteros a carácter, en donde cada puntero apunta a uno de los argumentos almacenados como cadena en algún lugar de la memoria.

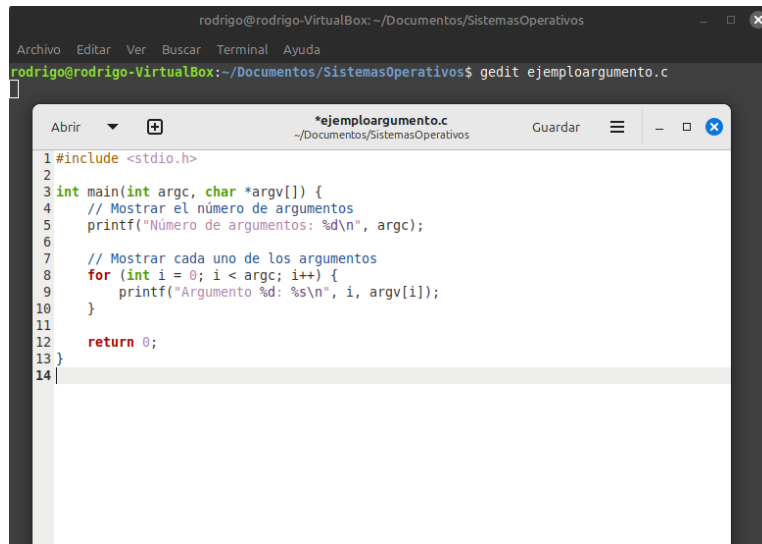
`argv[0]` apuntará a una cadena que contiene el nombre del programa. En la siguiente figura se muestra mediante una representación gráfica cuál sería el contenido de las variables `argc` y `argv` para el caso del ejemplo mostrado para la orden `ls` anterior.



Si el programa no ha sido escrito o preparado para recibir y tratar argumentos que se indiquen en la línea de comandos, no es necesario especificar nada en la declaración de la función `main()`.

### 5.0.1. Ejemplo de uso

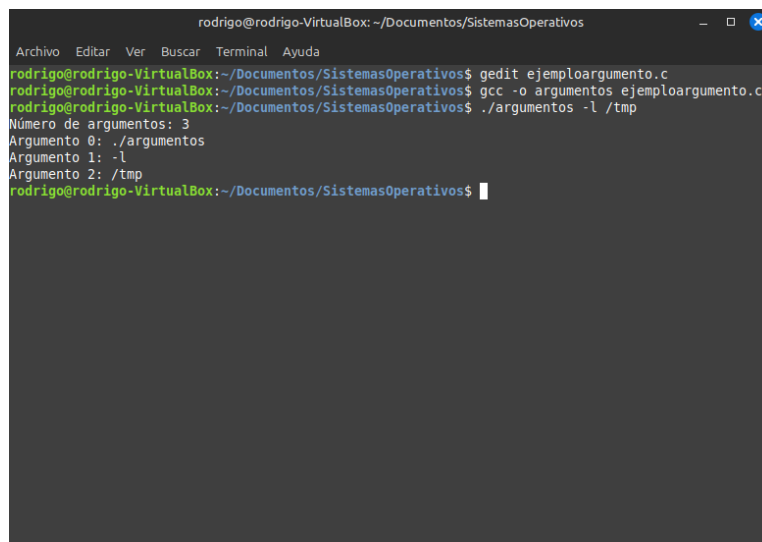
Crearemos un ejemplo de código en C que muestra cómo trabajar con los argumentos de la línea de comandos usando `argc` y `argv`:



```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[]) {
4     // Mostrar el número de argumentos
5     printf("Número de argumentos: %d\n", argc);
6
7     // Mostrar cada uno de los argumentos
8     for (int i = 0; i < argc; i++) {
9         printf("Argumento %d: %s\n", i, argv[i]);
10    }
11
12    return 0;
13 }
14
```

Le damos en guardar y cerramos la interfaz gráfica.

Compilamos el código, al ejecutar el programa con argumentos de línea de comandos `./argumentos -l /tmp` nos saldrá:



```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gedit ejemploargumento.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -o argumentos ejemploargumento.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./argumentos -l /tmp
Número de argumentos: 3
Argumento 0: ./argumentos
Argumento 1: -l
Argumento 2: /tmp
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$
```

Aquí, `argc` indica que hay 3 argumentos, y `argv` es un arreglo donde:

- `argv[0]` contiene el nombre del programa (`./argumentos`).
- `argv[1]` contiene `-l`.

- `argv[2]` contiene `/tmp`.

## 6. Comando `make`

Resulta realmente útil utilizar el programa `make` a la hora de trabajar con proyectos donde no tenemos uno o dos ficheros fuente, sino una amplia colección de ellos repartidos muchas veces en varios directorios.

El funcionamiento es sencillo: cuando llamamos a `make`, este busca un fichero de configuración en el directorio actual, que se debe llamar `Makefile` o `makefile`. Este fichero de configuración contiene lo que se conocen como dependencias, y le va a servir a `make` para comprobar que los ficheros de los que depende el proceso en cuestión están actualizados. Si no lo están, probablemente llevará a cabo alguna acción (como, por ejemplo, compilar una fuente para obtener el objeto).

Si alguna dependencia falla, es decir, algún fichero no existe y no tiene forma de crearlo, el programa dará un error y se detendrá. Si no, ejecutará los comandos indicados hasta el final.

Un fichero `Makefile` básico está formado por una lista de reglas. Las reglas constan de tres partes:

1. Una etiqueta, que usaremos en la llamada al programa `make`. Cada etiqueta distingue un grupo de acciones a realizar.
2. Un conjunto de dependencias, esto es, aquellos ficheros que son necesarios para llevar a cabo las acciones que corresponden a la etiqueta.
3. Los comandos a ejecutar.

La forma en la que se escriben las reglas es la siguiente:

```
etiqueta : [dependencia1 ...] [;comandos] [# comentarios]
      (tabulación) comandos  [# comentarios]
```

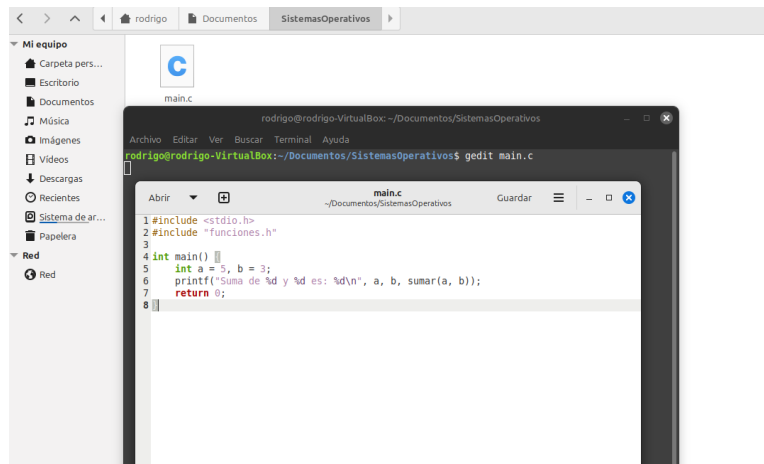
### 6.0.1. Ejemplo de uso

Crearemos un ejemplo práctico de un archivo `Makefile` que sigue una estructura de suma de números, con dependencias y comandos a ejecutar:

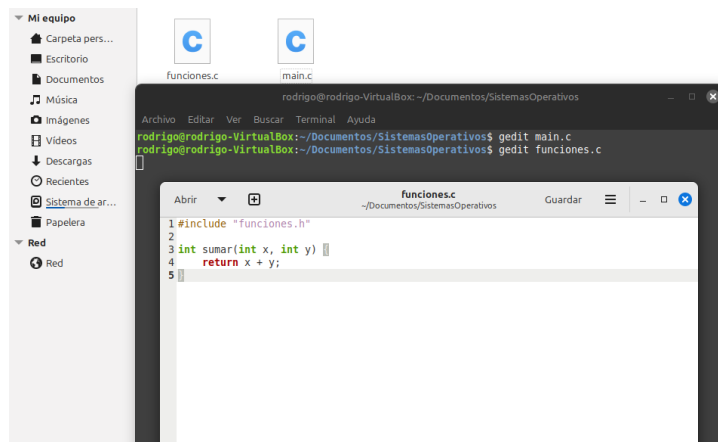
Estructura:

- `main.c`
- `funciones.c`
- `funciones.h`

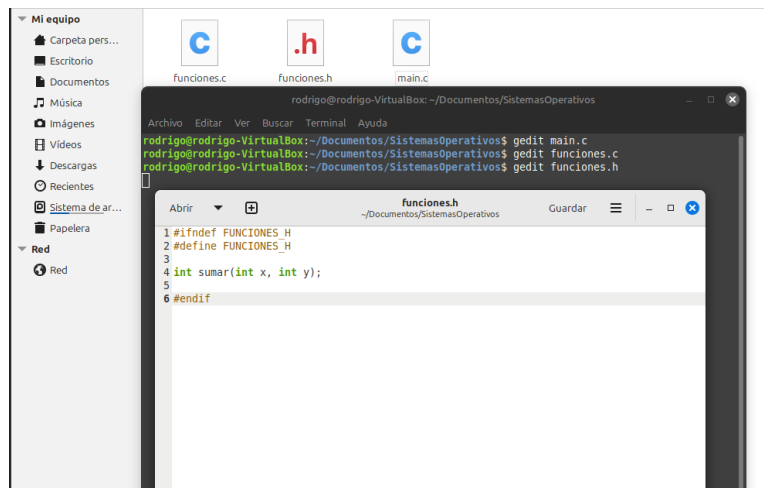
Creación del main.c:



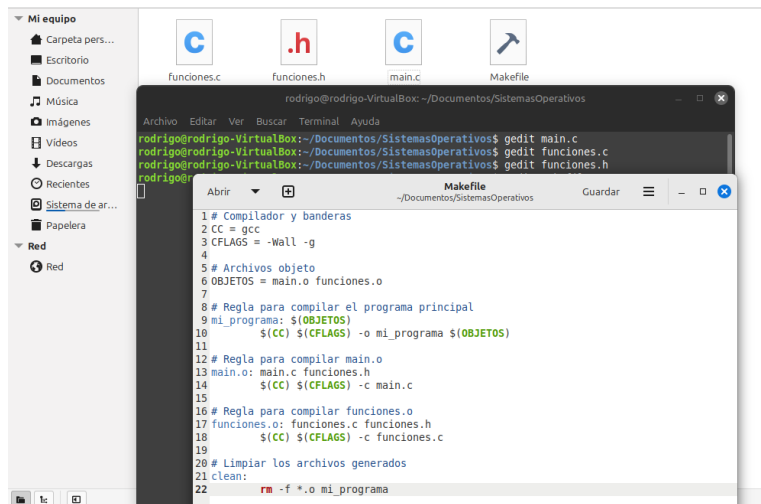
Creación de funciones.c:



Creación de funciones.h:



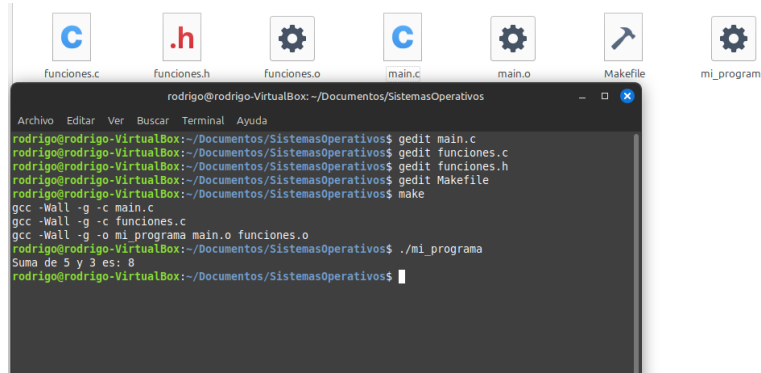
Creación del Makefile:



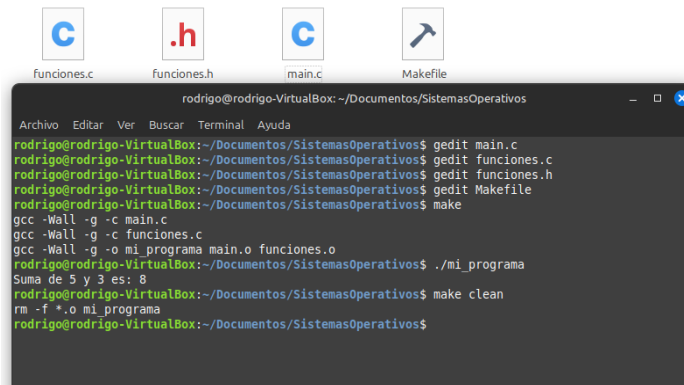
Explicación:

- **Compilador y banderas:**
  - CC = gcc: usa el compilador gcc.
  - CFLAGS = -Wall -g: activa todas las advertencias (-Wall) y genera información de depuración (-g).
- **Archivos objeto:**
  - OBJETOS = main.o funciones.o: define los archivos objeto que se deben generar a partir de los archivos .c.
- **Reglas:**
  - mi\_programa: es la etiqueta principal que genera el archivo ejecutable mi\_programa a partir de los objetos main.o y funciones.o. Si alguno de los archivos objeto está desactualizado, se recompila.
  - main.o: genera el objeto main.o si main.c o funciones.h han cambiado.
  - funciones.o: genera el objeto funciones.o si funciones.c o funciones.h han cambiado.
- **Comandos:** cada regla tiene un conjunto de comandos que se ejecutan si las dependencias están desactualizadas. Estos comandos están precedidos por una tabulación, no por espacios.
- **Limpieza:** la regla clean elimina los archivos objeto (\*.o) y el ejecutable mi\_programa para dejar el directorio limpio. Puedes ejecutarla con make clean.

Esto compilará el proyecto utilizando el Makefile que se creó con gedit. Si todo está correcto, se generará el ejecutable llamado `mi_programa`. Finalmente llamamos: `./mi_programa`



Podemos limpiar el make con el siguiente comando: `make clean`



## 7. Ejercicios propuestos

Se deberá de probar, compilar y ejecutar los siguientes códigos:

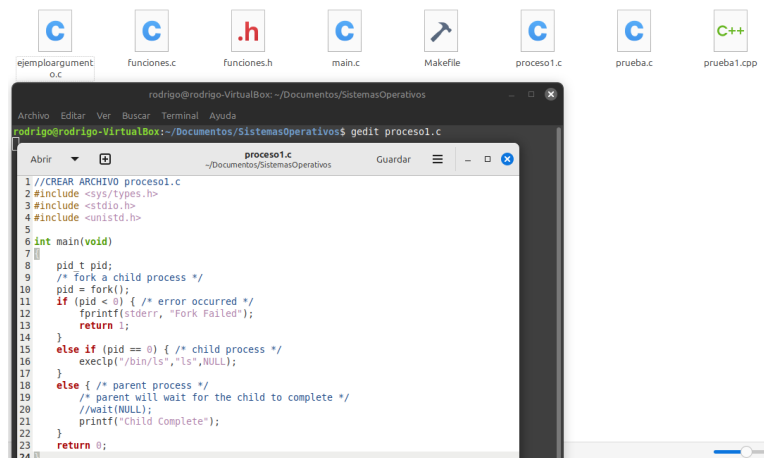
### 7.1. Código 1

```
1 //CREAR ARCHIVO proceso1.c
2 #include <sys/types.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 int main(void)
7 {
8     pid_t pid;
9     /* fork a child process */
10    pid = fork();
11    if (pid < 0) { /* error occurred */
12        fprintf(stderr, "Fork Failed");
13        return 1;
14    }
15    else if (pid == 0) { /* child process */
```

```
16     execlp("/bin/ls", "ls", NULL);
17 }
18 else { /* parent process */
19     /* parent will wait for the child to complete */
20     //wait(NULL);
21     printf("Child Complete");
22 }
23 return 0;
24 }
```

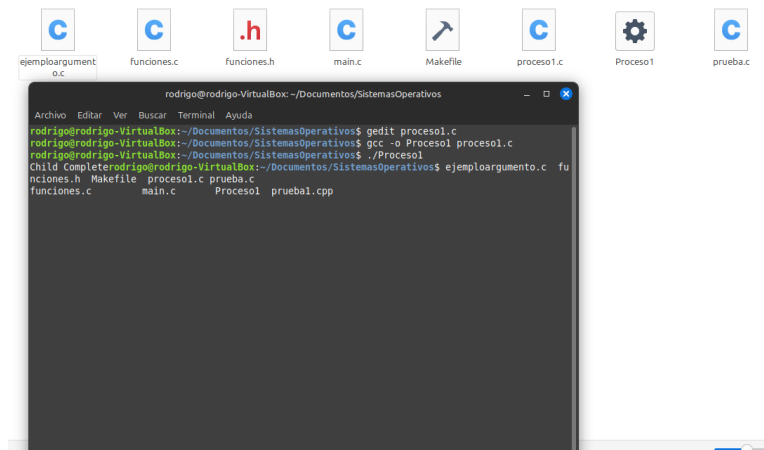
### Resolución:

Escribimos el código:



```
1 //CREAR ARCHIVO procesol.c
2 #include <sys/types.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 int main(void)
7 {
8     pid_t pid;
9     /* fork a child process */
10    pid = fork();
11    if (pid < 0) { /* error occurred */
12        fprintf(stderr, "Fork Failed");
13        return 1;
14    }
15    else if (pid == 0) { /* child process */
16        execlp("/bin/ls", "ls", NULL);
17    }
18    else { /* parent process */
19        /* parent will wait for the child to complete */
20        //wait(NULL);
21        printf("Child Complete");
22    }
23    return 0;
24 }
```

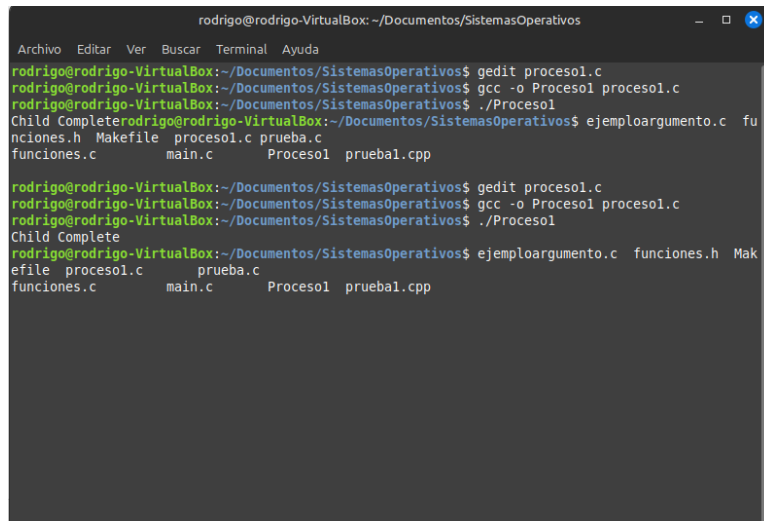
Compilamos y ejecutamos el código:



```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gedit procesol.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -o Procesol procesol.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./Procesol
Child Complete
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ejemplargumento.c fu
ncciones.h Makefile procesol.c prueba.c
funciones.c main.c Procesol prueba1.cpp
```



Ponemos salto de línea, compilamos y ejecutamos el código :



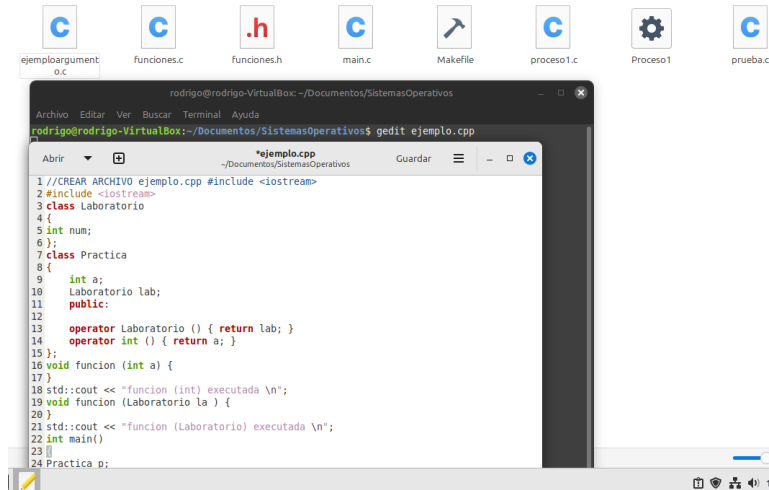
```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gedit procesol.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -o Procesol procesol.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./Procesol
Child Complete
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ejemploargumento.c fu
nciones.h Makefile procesol.c prueba.c
funciones.c main.c Procesol prueba1.cpp
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gedit procesol.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -o Procesol procesol.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./Procesol
Child Complete
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ejemploargumento.c funciones.h Mak
efile procesol.c prueba.c
funciones.c main.c Procesol prueba1.cpp
```

## 7.2. Código 2

```
1 //CREAR ARCHIVO ejemplo.cpp
2 #include <iostream>
3 class Laboratorio
4 {
5     int num;
6 };
7 class Practica
8 {
9     int a;
10    Laboratorio lab;
11 public:
12    operator Laboratorio() { return lab;}
13    operator int() {return a;}
14 };
15 void funcion(int a)
16 {
17     std::cout << "funcion (int) ejecutada \n";
18 }
19 void funcion(Laboratorio la)
20 {
21     std::cout << "funcion (Laboratorio) ejecutada \n";
22 }
23 int main()
24 {
25     Practica p;
26     funcion(static_cast<int>(p)); // Conversin explcita a int
27     return 0;
28 }
```

## Resolución:

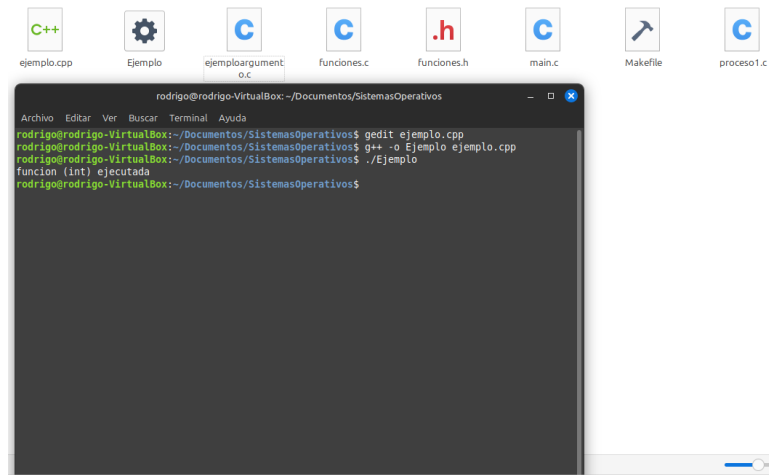
Escribimos el código:



```

rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gedit ejemplo.cpp
Abrir Guardar
1 //CREAR ARCHIVO ejemplo.cpp #include <iostream>
2 #include <iostream>
3 class Laboratorio
4 {
5     int num;
6 };
7 class Practica
8 {
9     int a;
10    Laboratorio lab;
11    public:
12
13    operator Laboratorio () { return lab; }
14    operator int () { return a; }
15 };
16 void funcion (int a) {
17 }
18 std::cout << "funcion (int) ejecutada \n";
19 void funcion (Laboratorio la) {
20 }
21 std::cout << "funcion (Laboratorio) ejecutada \n";
22 int main()
23 {
24     Practica p;
  
```

Compilamos y ejecutamos el código:



```

rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gedit ejemplo.cpp
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ g++ -o Ejemplo ejemplo.cpp
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./Ejemplo
funcion (int) ejecutada
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$
  
```

## 7.3. Código 3

```

1 //CREAR ARCHIVO LinkedList.h
2 #pragma once
3
4 #include "ListNode.h"
5 #include <iostream>
6
7 class LinkedList {
8     private:
9         ListNode* _phead;
10
11     public:
12         LinkedList();
  
```

```
13     void insert(int n);
14     void print(void);
15
16     void deleteAll(void);
17     void deleteNodes(ListNode* pn);
18 };
```

```
1  //CREAR ARCHIVO LinkedList.cpp
2  #include "LinkedList.h"
3
4  LinkedList::LinkedList() {
5      _phead = nullptr;
6  }
7  void LinkedList::insert(int n) {
8      if (_phead == nullptr) {
9          _phead = new ListNode();
10         _phead->_value = n;
11         return;
12     } else {
13         ListNode* pn = new ListNode();
14         pn->_value = n;
15         ListNode* pnode = _phead;
16         while (pnode->_pNext != nullptr && pnode->_pNext->_value < n) {
17             pnode = pnode->_pNext;
18         }
19         if (pnode->_pNext != nullptr && pnode->_value > n) {
20             pn->_pNext = _phead;
21             _phead = pn;
22         }
23         if (pnode->_pNext == nullptr) {
24             pnode->_pNext = pn;
25         } else {
26             pn->_pNext = pnode->_pNext;
27             pnode->_pNext = pn;
28         }
29     }
30 }
31 void LinkedList::print(void) {
32     ListNode* pnode = _phead;
33     while (pnode->_pNext != nullptr) {
34         std::cout << pnode->_value << " ";
35         pnode = pnode->_pNext;
36     }
37 }
38 void LinkedList::deleteAll(void) {
39     if (_phead != nullptr) {
40         deleteNodes(_phead);
41     }
42 }
43 void LinkedList::deleteNodes(ListNode* pn) {
44     if (pn->_pNext != nullptr) {
45         deleteNodes(pn->_pNext);
46     }
47     delete(pn);
48 }
```

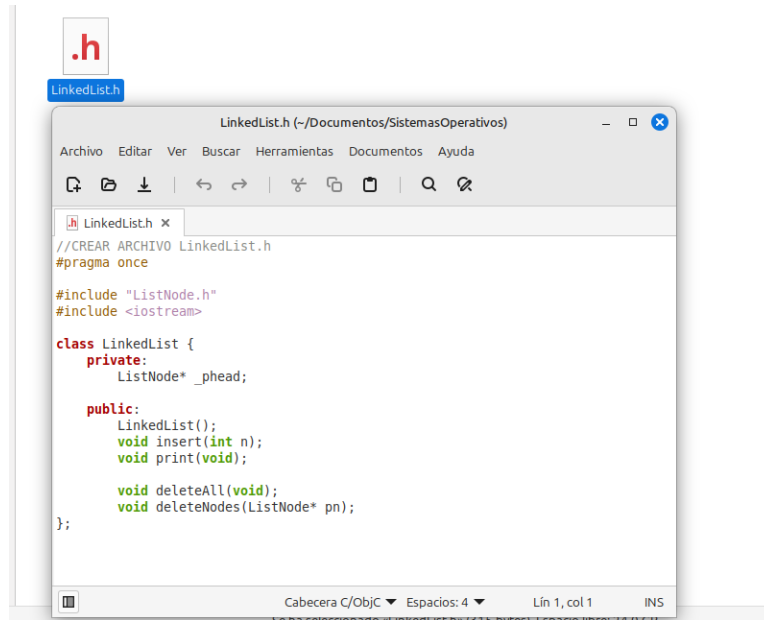
```
1 //CREAR ARCHIVO ListNode.cpp
2 #include "ListNode.h"
3 ListNode::ListNode() {
4     _value = -1;
5     _pNext = nullptr;
6 }
7 ListNode::~ListNode() {
8     //delete _pNext;
9 }
```

```
1 //CREAR ARCHIVO ListNode.h
2 #pragma once
3 class ListNode {
4     public:
5         int _value;
6         ListNode* _pNext;
7
8         ListNode();
9         ~ListNode();
10 };
```

```
1 //CREAR ARCHIVO main.cpp
2 #include <iostream>
3 #include <stdlib.h>
4 #include "LinkedList.h"
5
6 int main() {
7     int max = 10;
8     LinkedList* plist = new LinkedList();
9
10    for (int i = 0; i < max; i++) {
11        int num = rand() % max;
12        plist->insert(num);
13    }
14    plist->print();
15    plist->deleteAll();
16    delete(plist);
17
18    return 0;
19 }
```

## Resolución:

Creación del archivo LinkedList.h:



The screenshot shows a code editor window titled "LinkedList.h (~/Documentos/SistemasOperativos)". The editor contains the following C++ code:

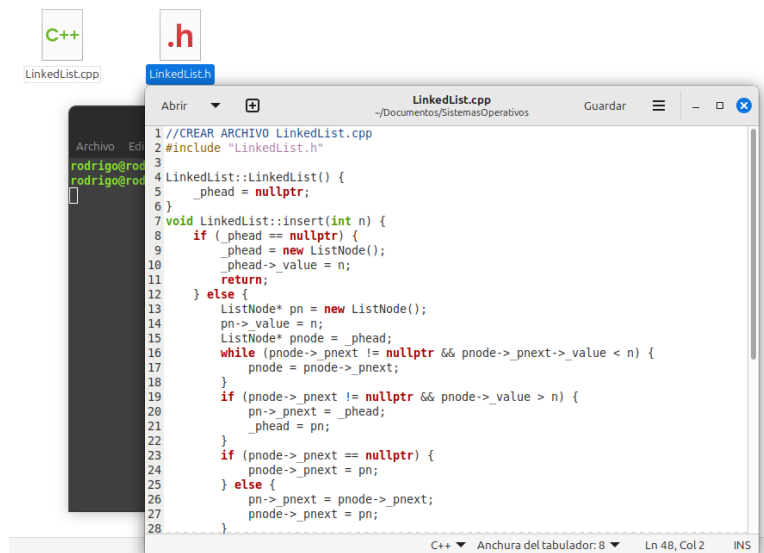
```
//CREAR ARCHIVO LinkedList.h
#pragma once

#include "ListNode.h"
#include <iostream>

class LinkedList {
private:
    ListNode* _phead;

public:
    LinkedList();
    void insert(int n);
    void print(void);
    void deleteAll(void);
    void deleteNodes(ListNode* pn);
};
```

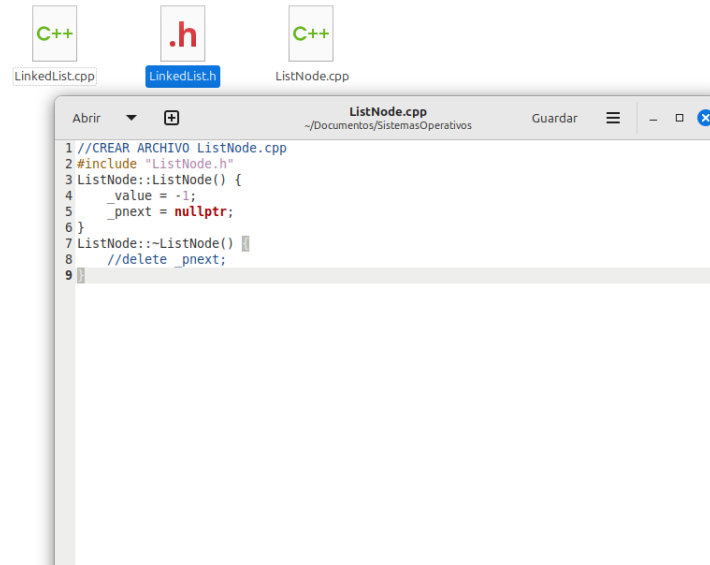
Creación del archivo LinkedList.cpp:



The screenshot shows a code editor window titled "LinkedList.cpp (~/Documentos/SistemasOperativos)". The editor contains the following C++ code:

```
1 //CREAR ARCHIVO LinkedList.cpp
2 #include "LinkedList.h"
3
4 LinkedList::LinkedList() {
5     _phead = nullptr;
6 }
7 void LinkedList::insert(int n) {
8     if (_phead == nullptr) {
9         _phead = new ListNode();
10        _phead->_value = n;
11        return;
12    } else {
13        ListNode* pn = new ListNode();
14        pn->_value = n;
15        ListNode* pnode = _phead;
16        while (pnode->_pNext != nullptr && pnode->_pNext->_value < n) {
17            pnode = pnode->_pNext;
18        }
19        if (pnode->_pNext != nullptr && pnode->_pNext->_value > n) {
20            pn->_pNext = _phead;
21            _phead = pn;
22        }
23        if (pnode->_pNext == nullptr) {
24            pn->_pNext = pnode;
25        } else {
26            pn->_pNext = pnode->_pNext;
27            pnode->_pNext = pn;
28        }
29    }
```

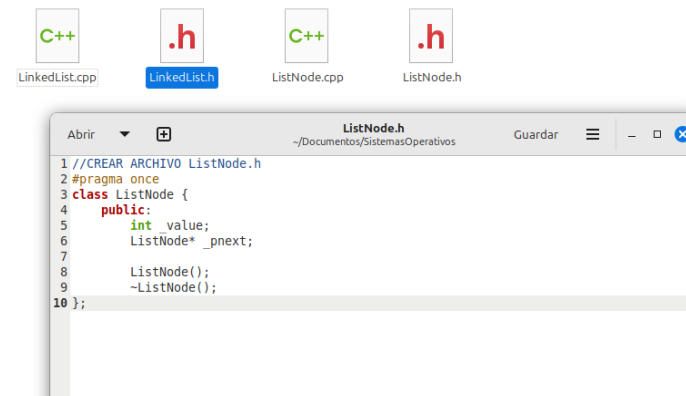
Creación del archivo ListNode.cpp:



The screenshot shows a code editor window titled "ListNode.cpp" with the following code:

```
1 //CREAR ARCHIVO ListNode.cpp
2 #include "ListNode.h"
3 ListNode::ListNode() {
4     _value = -1;
5     _pNext = nullptr;
6 }
7 ListNode::~ListNode() {}
8 //delete _pNext;
9
```

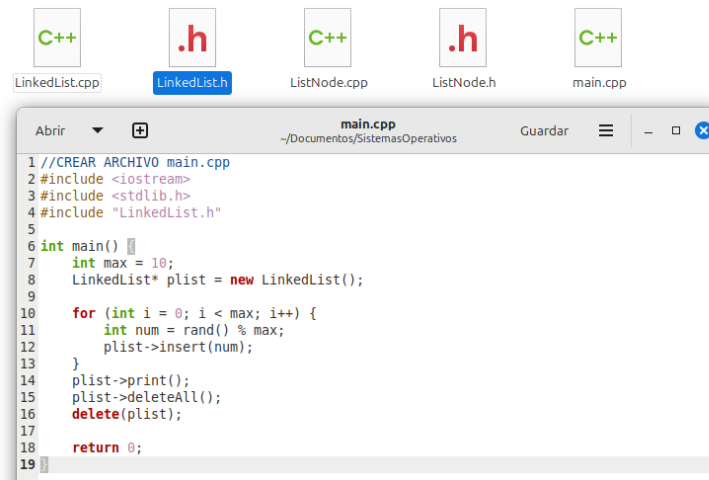
Creación del archivo ListNode.h:



The screenshot shows a code editor window titled "ListNode.h" with the following code:

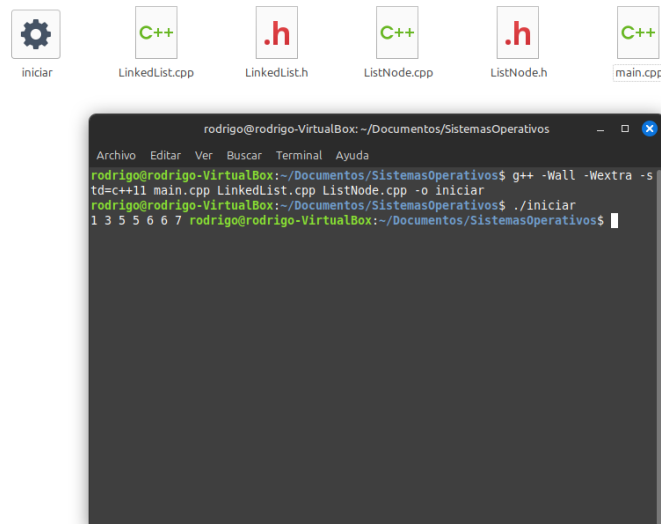
```
1 //CREAR ARCHIVO ListNode.h
2 #pragma once
3 class ListNode {
4 public:
5     int _value;
6     ListNode* _pNext;
7     ListNode();
8     ~ListNode();
9 };
10
```

Creación del archivo main.cpp:

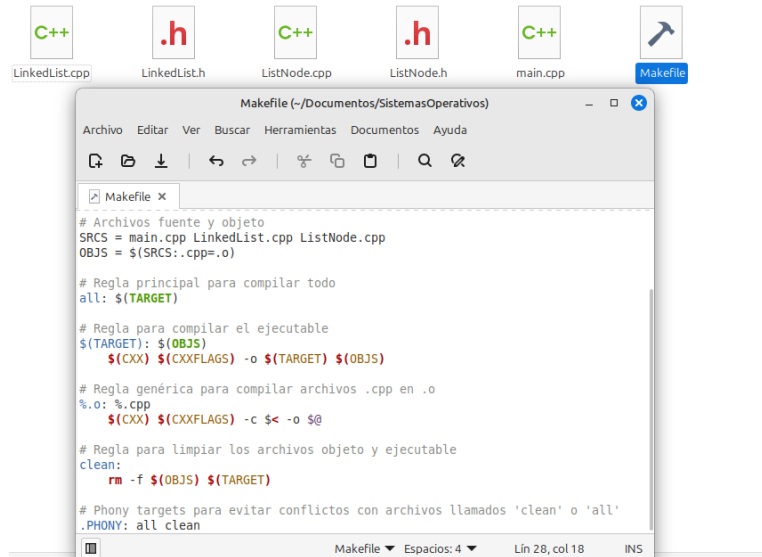


Ejecución del código:

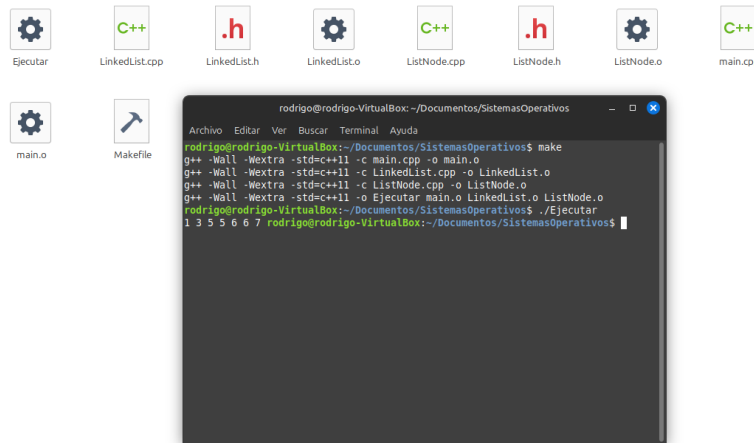
Se puede usar el comando: `g++ -Wall -Wextra -std=c++11 main.cpp LinkedList.cpp ListNode.cpp -o iniciar`



O de otro modo, crear un Makefile:

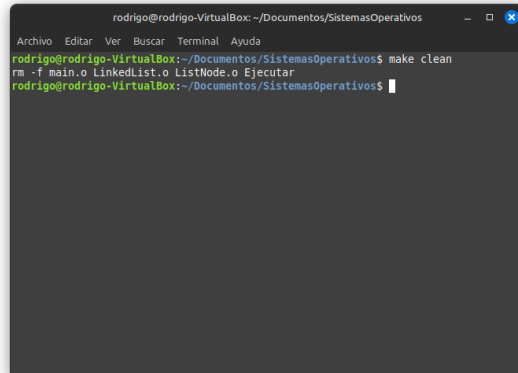


Ejecutamos:





Limpiamos el make:



## 7.4. Código 4

Teniendo en cuenta lo descrito en el capítulo: de línea de comandos, escriba, compile y ejecute el siguiente programa:

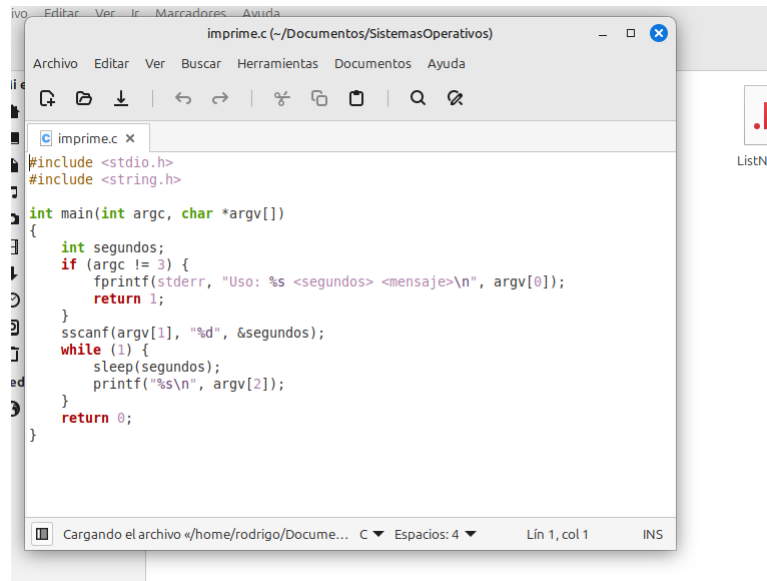
```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(int argc, char *argv[])
5 {
6     int segundos;
7     if (argc != 3) {
8         fprintf(stderr, "Uso: %s <segundos> <mensaje>\n", argv[0]);
9         return 1;
10    }
11    sscanf(argv[1], "%d", &segundos);
12    while (1) {
13        sleep(segundos);
14        printf("%s\n", argv[2]);
15    }
16    return 0;
17 }
```

Para ejecutar el programa de hacerlo desde la línea de comandos a través de una orden como la siguiente: \$ imprime <segundo><mensaje>

¿Qué problemas podemos tener al tratar la línea de argumentos? Razone la respuesta

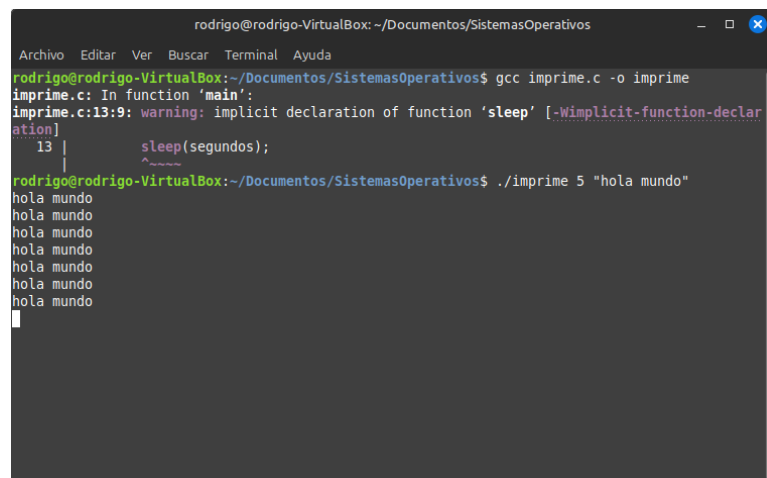
### Resolución:

Escribimos el código:



```
imprime.c (~/Documentos/SistemasOperativos)
Archivo  Editar  Ver  Buscar  Herramientas  Documentos  Ayuda
imprime.c x
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
    int segundos;
    if (argc != 3) {
        fprintf(stderr, "Uso: %s <segundos> <mensaje>\n", argv[0]);
        return 1;
    }
    sscanf(argv[1], "%d", &segundos);
    while (1) {
        sleep(segundos);
        printf("%s\n", argv[2]);
    }
    return 0;
}
```

Ejecutamos como se nos indica, segundos y el mensaje:



```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc imprime.c -o imprime
imprime.c: In function 'main':
imprime.c:13:9: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
   13 |         sleep(segundos);
      |         ^~~~~
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./imprime 5 "hola mundo"
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
hola mundo
```

### Resolución de la pregunta:

En primer lugar, al ejecutar el código nos aparece el siguiente mensaje:

```
warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
```

El compilador no encuentra la declaración de la función `sleep()` antes de que la intente utilizar en el código. En C, todas las funciones que no son nativas del lenguaje deben ser declaradas o incluidas mediante bibliotecas adecuadas antes de poder usarlas.

La función `sleep()` pertenece a la biblioteca estándar de Unix, que se encuentra en el archivo de cabecera `<unistd.h>`. Sin la inclusión de este archivo de cabecera, el compilador no sabe que `sleep()` existe, lo que genera una advertencia de "declaración implícita".

Ahora, los posibles problemas principales que podrían surgir al manejar la línea de comandos está relacionado con la cantidad y el tipo de argumentos pasados, tales como:

- **Número incorrecto de argumentos:** el código verifica si argc es distinto de 3 (if (argc != 3)), lo que implica que debe haber 3 argumentos: el nombre del programa (argv[0]), los segundos (argv[1]), y el mensaje (argv[2]).
- **Conversión incorrecta del argumento segundos:** El argumento que representa los segundos (argv[1]) se convierte a un entero usando sscanf(). Si se introduce un valor que no sea un número entero válido, la conversión puede fallar. Esto puede llevar a un valor indefinido para los segundos, lo que podría causar un comportamiento impredecible en el ciclo while.
- **Errores con el mensaje:** el tercer argumento (argv[2]) se imprime repetidamente en el ciclo while. Si este argumento no se proporciona correctamente, es vacío o contiene caracteres no deseados, el programa seguirá ejecutándose indefinidamente, pero el mensaje impreso puede ser inesperado o incorrecto.

## 7.5. Código 5

En el siguiente ejercicio configurará un archivo Makefile que incluya las instrucciones para ejecutar los siguientes archivos los mismos que deberá escribirlos en primera instancia:

```
1 //Archivo salida_alt.h
2 #ifndef __SALIDA_ALT__
3 #define __SALIDA_ALT__
4
5 void muestra(char *);
6
7 #endif
```

```
1 //Archivo salida_alt.c
2 #include <stdio.h>
3
4 void muestra(char *msg) {
5     printf("Mensaje: %s\n", msg);
6 }
```

```
1 //Archivo mensaje.c
2 #include "salida_alt.h"
3
4 int main() {
5     muestra("Muestra este bonito mensaje por la salida estndar");
6     return 0;
7 }
```

Ahora para compilar estos archivos va a escribir lo siguiente en un archivo de nombre Makefile, después ejecutará make y observará los resultados:

Makefile a escribir:

```
mensaje: mensaje.o salida_alt.o
    gcc -o mensaje mensaje.o salida_alt.o

mensaje.o: mensaje.c salida_alt.h
```

```
gcc -c -g mensaje.c
```

```
salida_alt.o: salida_alt.c salida_alt.h
```

```
gcc -c -g salida_alt.c
```

### Resolución:

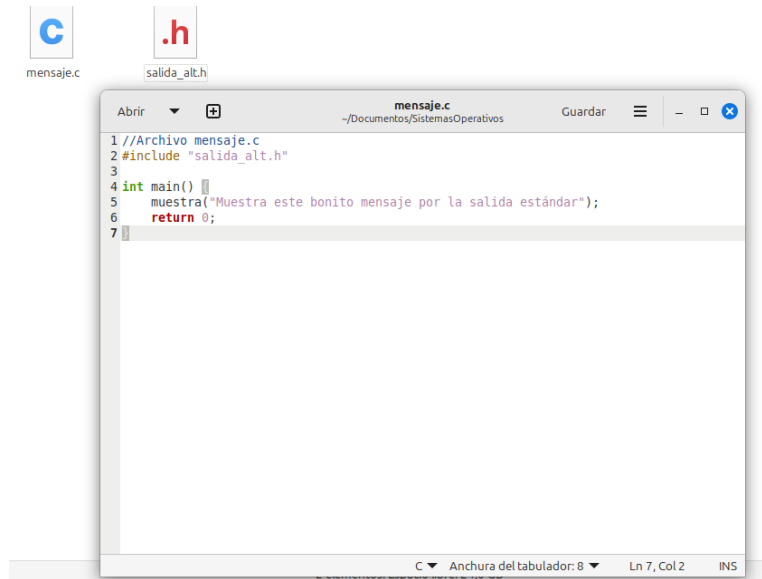
Escribimos el código salida\_alt.h:



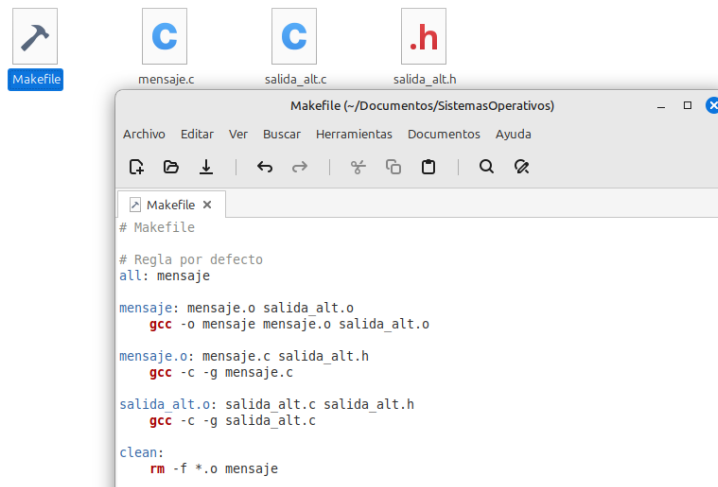
Escribimos el código salida\_alt.c:



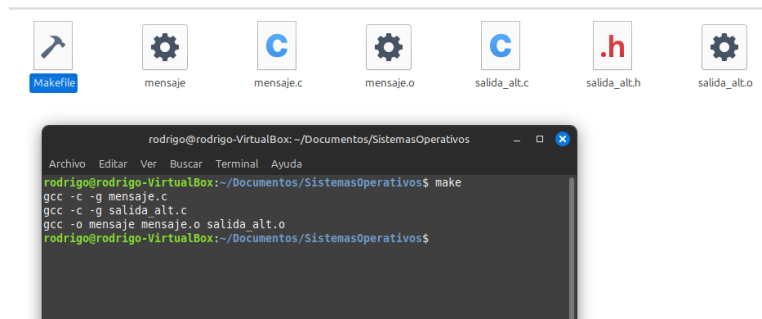
Escribimos el código mensaje.c:



Escribimos el Makefile:

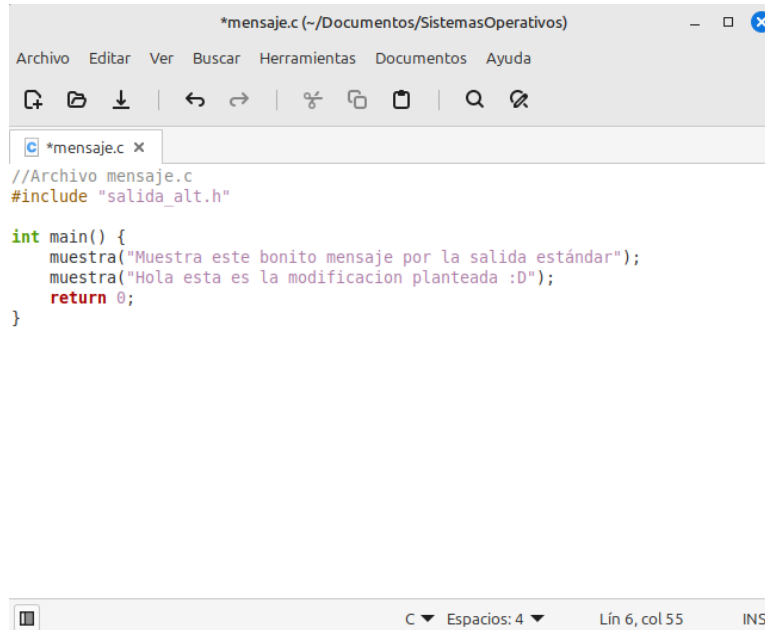


Ejecutamos el Makefile:



Ahora realice alguna modificación (las que usted considere) al archivo mensaje.c y ejecute nuevamente make:

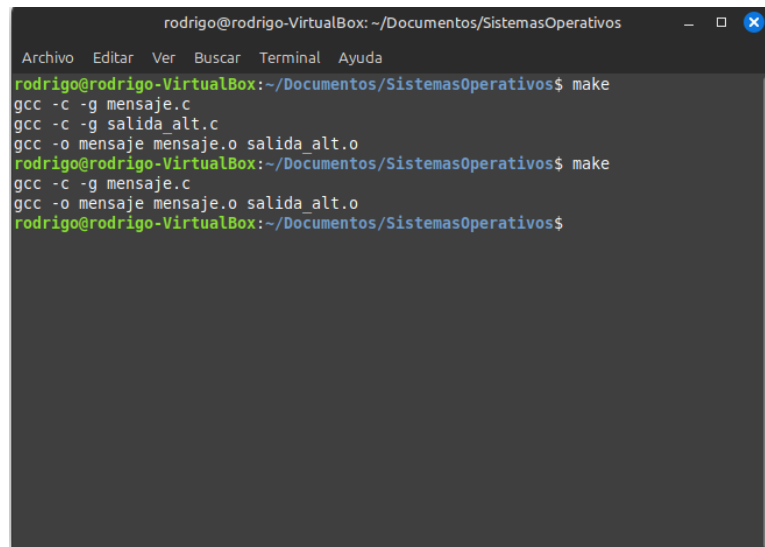
Editamos mensaje.c:



```
*mensaje.c (~/Documentos/SistemasOperativos)
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
//Archivo mensaje.c
#include "salida_alt.h"

int main() {
    muestra("Muestra este bonito mensaje por la salida estándar");
    muestra("Hola esta es la modificacion planteada :D");
    return 0;
}
```

Ejecutamos nuevamente el Makefile:



```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ make
gcc -c -g mensaje.c
gcc -c -g salida_alt.c
gcc -o mensaje.o mensaje.o salida.o
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ make
gcc -c -g mensaje.c
gcc -o mensaje.o mensaje.o salida.o
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$
```

Describe lo que hace make en relación a las modificaciones realizadas en el archivo mensaje.c

**Respuesta:**

Lo que hace make es verificar las dependencias en el Makefile, compila los archivos fuente que han sido modificados desde la última compilación y enlazarlos para generar el ejecutable final de manera más automática y eficiente posible.

## 8. Cuestionario

### 1.- ¿Cuál es la diferencia entre compilar con gcc y g++?

El **gcc** compila programas escritos en C. Utiliza un enlazador adecuado para C y trata los archivos con extensión `.c` como archivos de C por defecto.

En cambio, **g++** compila programas escritos en C++. También puede compilar código C, pero trata todo el código como C++ por defecto, incluso los archivos con extensión `.c`. Además, g++ enlaza automáticamente las bibliotecas estándar de C++.

### 2.- ¿En qué se diferencia el archivo generado `.o` contra un `.exe` ?

Por un lado, `.o` es un archivo de código objeto generado tras la compilación de un archivo fuente (`.c` o `.cpp`). Contiene el código binario traducido por el compilador, pero no está listo para ejecutarse directamente porque aún no ha sido enlazado con otras posibles dependencias y bibliotecas.

Por otro lado, `.exe` es el archivo final que se genera tras el proceso de enlazado. Un archivo `.exe` contiene todos los archivos objeto y bibliotecas necesarias, ya combinados, y es completamente ejecutable por el sistema operativo.

### 3.- Explique cuál es la diferencia entre el proceso de compilación y enlazado . Proponga un ejemplo en que haga uso de más de un archivo donde se evidencie ambos procesos.

Primeramente, **la compilación** traduce el código fuente (C o C++) a código objeto (`.o`), que contiene instrucciones en lenguaje máquina pero no puede ejecutarse directamente. Este proceso se hace archivo por archivo.

De otro modo, **el enlazado** une todos los archivos objeto (`.o`) generados durante la compilación y resuelve todas las referencias a funciones o variables externas, creando el archivo ejecutable final (`.exe`) en Windows.

#### Propuesta:

Supongamos que tenemos funciones de cálculo de áreas:

Trapezio:

```
1 //Archivo: calculo_trapezio.c
2 #include <stdio.h>
3
4 float area_trapezio(float base1, float base2, float altura) {
5     return ((base1 + base2) / 2) * altura;
6 }
```

Circulo:

```
1 //Archivo: calculo_circulo.c
2 #include <stdio.h>
3 #define PI 3.1416
4
5 float area_circulo(float radio) {
6     return PI * radio * radio;
7 }
```

Triangulo:

```
1 //Archivo: calculo_triangulo.c
2 #include <stdio.h>
3
4 float area_triangulo(float base, float altura) {
```

```
5     return (base * altura) / 2;  
6 }
```

Rombo:

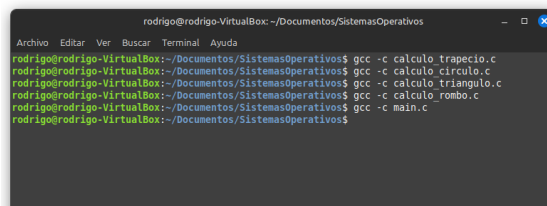
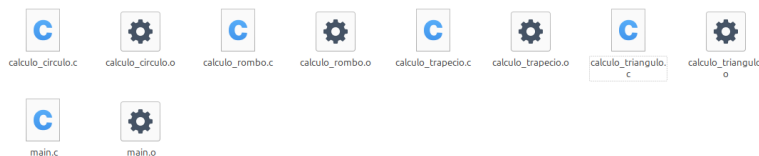
```
1 //Archivo: calculo_rombo.c  
2 #include <stdio.h>  
3  
4 float area_rombo(float diagonalMayor, float diagonalMenor) {  
5     return (diagonalMayor * diagonalMenor) / 2;  
6 }
```

Main:

```
1 //Archivo: main.c  
2 #include <stdio.h>  
3  
4 // Prototipos de las funciones  
5 float area_trapezio(float base1, float base2, float altura);  
6 float area_circulo(float radio);  
7 float area_triangulo(float base, float altura);  
8 float area_rombo(float diagonalMayor, float diagonalMenor);  
9  
10 int main() {  
11     float base1 = 5, base2 = 7, altura = 4;  
12     float radio = 3;  
13     float base = 6;  
14     float diagonalMayor = 8, diagonalMenor = 6;  
15  
16     printf("rea del trapezio: %.2f\n", area_trapezio(base1, base2, altura));  
17     printf("rea del circulo: %.2f\n", area_circulo(radio));  
18     printf("rea del triangulo: %.2f\n", area_triangulo(base, altura));  
19     printf("rea del rombo: %.2f\n", area_rombo(diagonalMayor, diagonalMenor));  
20  
21     return 0;  
22 }
```

Por compilación:

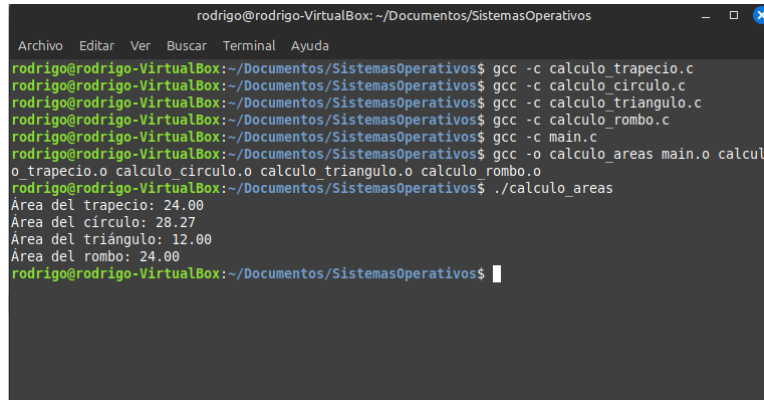
Una vez creado los archivos, compilaremos cada archivo .c de manera individual. El comando gcc -c crea archivos objeto (.o) sin enlazarlos aún:





**Por enlace:**

Ahora que tenemos todos los archivos .o (objetos) generados por la compilación, el siguiente paso es enlazarlos para crear el ejecutable final.



```
rodrigo@rodrigo-VirtualBox: ~/Documentos/SistemasOperativos
Archivo Editar Ver Buscar Terminal Ayuda
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -c calculo_trapezio.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -c calculo_circulo.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -c calculo_triangulo.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -c calculo_rombo.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -c main.c
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ gcc -o calculo_areas main.o calcul
o trapezio.o calculo_circulo.o calculo_triangulo.o calculo_rombo.o
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$ ./calculo_areas
Area del trapezio: 24.00
Area del círculo: 28.27
Area del triángulo: 12.00
Area del rombo: 24.00
rodrigo@rodrigo-VirtualBox:~/Documentos/SistemasOperativos$
```

**Resumen del proceso:**

**Compilación:** el comando `gcc -c` compila cada archivo .c y genera un archivo objeto .o. Este proceso convierte el código fuente en código de máquina, pero los archivos no están listos para ejecutarse aún.

**Enlazado:** el comando `gcc -o` toma los archivos objeto (.o) y los enlaza, resolviendo referencias entre los diferentes archivos (como las llamadas a funciones) para generar un ejecutable completo (calculo\_areas).